The R Journal

Volume 9/2, December 2017

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

Contents

Editorial 4

Contributed Research Articles

anchoredDistr: a Package for the Bayesian Inversion of Geostatistical Parameters with Multi-type and Multi-scale Data 6

dGAselID: An R Package for Selecting a Variable Number of Features in High Dimensional Data 18

Allele Imputation and Haplotype Determination from Databases Composed of Nuclear Families 35

Visualization of Regression Models Using **visreg** 56

fourierin: An R package to compute Fourier integrals 72

Discrete Time Markov Chains with R 84

CRTgeeDR: an R Package for Doubly Robust Generalized Estimating Equations Estimations in Cluster Randomized Trials with Missing Data 105

queueing: A Package For Analysis Of Queueing Networks and Models in R 116

ctmcd: An R Package for Estimating the Parameters of a Continuous-Time Markov Chain from Discrete-Time Data 127

Furniture for Quantitative Scientists 142

BayesBD: An R Package for Bayesian Inference on Image Boundaries 149

arulesViz: Interactive Visualization of Association Rules with R 163

ManlyMix: An R Package for Manly Mixture Modeling 176

adegraphics: An S4 Lattice-Based Package for the Representation of Multivariate Data 198

carx: an R Package to Estimate Censored Autoregressive Time Series with Exogenous Covariates 213

liureg: A Comprehensive R Package for the Liu Estimation of Linear Regression Model with Collinear Regressors 232

A Tidy Data Model for Natural Language Processing using **cleanNLP** 248

mle.tools: An R Package for Maximum Likelihood Bias Correction 268

afmToolkit: an R Package for Automated AFM Force-Distance Curves Analysis . . . 291

The **welchADF** Package for Robust Hypothesis Testing in Unbalanced Multivariate Mixed Models with Heteroscedastic and Non-normal Data 309

ider: Intrinsic Dimension Estimation with R 329
rpsftm: An R Package for Rank Preserving Structural Failure Time Models 342
anomalyDetection: Implementation of Augmented Network Log Anomaly Detection Procedures 354
Simulating Noisy, Nonparametric, and Multivariate Discrete Patterns 366
glmmTMB Balances Speed and Flexibility Among Packages for Zero-inflated Generalized Linear Mixed Modeling 378
Simulating Probabilistic Long-Term Effects in Models with Temporal Dependence . . 401
RQGIS: Integrating R with QGIS for Statistical Geocomputing 409
Partial Rank Data with the **hyper2** Package: Likelihood Functions for Generalized Bradley-Terry Models 429
riskRegression: Predicting the Risk of an Event using Cox Regression Models 440
LeArEst: Length and Area Estimation from Data Measured with Additive Error . . . 461
Splitting It Up: The **spduration** Split-Population Duration Regression Package for Time-Varying Covariates 474
Bayesian Regression Models for Interval-censored Data in R 487
openEBGM: An R Implementation of the Gamma-Poisson Shrinker Data Mining Model 499
rentrez: An R package for the NCBI eUtils API 520
An Introduction to Rocker: Docker Containers for R 527

News and Notes

Conference Report: useR!2017 537
Conference Report: R in Insurance 2017 539
Forwards Column 541
R Teaching Column 553
R Foundation News 563
Changes on CRAN 564
News from the Bioconductor Project 567
Changes in R 568

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

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

Editorial

by Roger Bivand

In my editorial for the 2017–1 issue, I concentrated on tabulating the status of this journal with respect to its authors and reviewers (updated tables below). This time, I was prompted by an interesting blog posting by Jan Wijffels of BNOSAC, describing the use of the **udpipe** package to apply natural language processing (NLP) to the CRAN package database available from `tools::CRAN_package_db()` since the release of R 3.4. The interactive NLP searcher is a dashboard permitting exploration of annotated CRAN package title and description NLP data.

It struck me that an analysis of abstracts of contributed research articles published in the R Journal would now be possible since the introduction of article landing pages earlier this year, because the website configuration file containing the abstracts can be read using the **yaml** package. Jan Wijffels kindly and rapidly responded, providing an R Journal NLP search tool analogous to the CRAN NLP search tool.

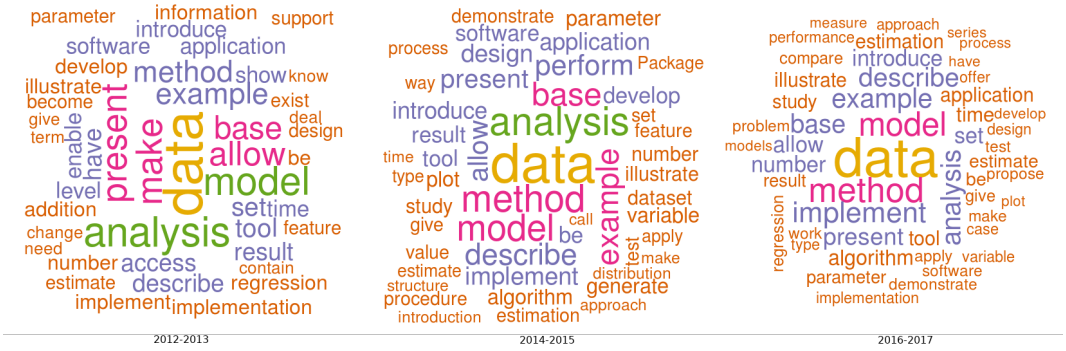


Figure 1: Wordcloud for abstracts of contributed research articles: left panel 53 articles 2012–2013, centre panel: 69 articles 2014–2015, right panel: 115 articles 2016–2017.

Figure 1 shows total cumulative wordclouds for the last six years in two-year slices, and indicates that we are, broadly, maintaining topical consistency with a sustained focus on data. The search tool permits much more detailed exploration as well, such as term search to supplement web searches on `site:journal.r-project.org`.

While we do not have on-site indexing or searching, a tab has been added for news and notes contributions by issue. In this issue, two new columns are initiated, one for news and notes from Forwards, starting with a report on the useR! 2016 survey (see also **forwards**). The second new column covers teaching R and teaching with R, and kicks off with a note on linking teaching and reproducible research (see also the **revisit** package). Progress in answering Heather Turner's appeal to help useRs navigate their way through the R world (editorial, 2011–1) is at best incremental, but progress none the less. A year later, Martyn Plummer pointed out (editorial, 2012–1) that "it is worth spending some time browsing these sections in order to catch up on changes you may have missed."

From the publication of this issue, fuller benefits from the introduction of landing pages may be realised through the addition of citation page metadata tags, permitting search engines to index contributed research articles more efficiently, thanks to a suggestion by Carl Boettiger. We expect to begin providing DOI for published contributed research articles during 2018 as a further step towards increasing the visibility of the valuable work published here.

We have already added links for supplementary matter (typically reproduction code) on article landing pages, for articles published in this issue. Since the beginning of 2017, submissions were expected to provide scripts permitting reviewers to run code without copying from the manuscript, but previously this was only exceptionally the case, so it may not be practical to provide supplementary matter for articles published in earlier issues.

The wisdom of the editors in choosing to consolidate, and become a listed journal (see Peter Dalgaard's editorial in 2010–1) is manifest in our current standing in Journal Citation Reports, with a 2016 impact factor of 1.075, and a five-year score of 2.114. The steps being taken by the editors and the R Foundation should enhance the discoverability and impact of work published here. It is fair to repeat from the 2010–1 editorial that "we need to show that we have a solid scientific standing with good editorial standards, giving submissions fair treatment and being able to publish on time."

	2009	2010	2011	2012	2013	2014	2015	2016	2017
Published	26	26	26	22	31	36	51	74	24
Rejected	11	14	11	24	29	32	53	68	55
Under review	0	0	0	0	0	0	0	2	65
Total	37	40	37	46	60	68	104	144	144

Table 1: Submission outcomes 2009–2017, by year of submission.

	2009	2010	2011	2012	2013	2014	2015	2016	2017
Page count	109	123	123	136	362	358	479	895	1023
Article count	18	18	20	18	35	33	36	62	68
Average length	6.1	6.8	6.2	7.6	10.3	10.8	13.3	14.4	15.0

Table 2: Published contributed articles 2009–2017, by year of publication.

	2013	2014	2015	2016	2017
Median	347.0	225.5	212.5	212.0	244.0

Table 3: Median day count from acknowledgement to acceptance and online publication 2013–2017, by year of publication.

Roger Bivand
Roger.Bivand@r-project.org

anchoredDistr: a Package for the Bayesian Inversion of Geostatistical Parameters with Multi-type and Multi-scale Data

by Heather Savoy, Falk Heße, and Yoram Rubin

Abstract The Method of Anchored Distributions (MAD) is a method for Bayesian inversion designed for inferring both local (e.g. point values) and global properties (e.g. mean and variogram parameters) of spatially heterogenous fields using multi-type and multi-scale data. Software implementations of MAD exist in C++ and C# to import data, execute an ensemble of forward model simulations, and perform basic post-processing of calculating likelihood and posterior distributions for a given application. This article describes the R package **anchoredDistr** that has been built to provide an R-based environment for this method. In particular, **anchoredDistr** provides a range of post-processing capabilities for MAD software by taking advantage of the statistical capabilities and wide use of the R language. Two examples from stochastic hydrogeology are provided to highlight the features of the package for MAD applications in inferring anchored distributions of local parameters (e.g. point values of transmissivity) as well as global parameters (e.g. the mean of the spatial random function for hydraulic conductivity).

Introduction

The field of geostatistics originated in the 1950s with the pioneering work of Krige (1951) and Matheron (1962) who tried to estimate the characteristics of subsurface properties with the limited measurements typically available in this field. This scarcity, caused by the high explorations costs, is exacerbated by the strong heterogeneity that many such subsurface properties exhibit. Both these factors combined make it impossible to describe any subsurface process with certainty, therefore necessitating the application of statistical tools. Today, geostatistics is used in many fields of earth science such as geology (Hohn, 1962), hydrogeology (Kitanidis, 2008), plus hydrology and soil science (Goovaerts, 1999). To meet this demand, many software packages have been developed that provide practitioners and scientists alike with the much needed tools to apply geostatistics. In R, the best collection of such tools is arguably found in the **gstat** package (Pebesma, 2004) developed and maintained by Pebesma and colleagues. With **gstat**, it is possible to estimate (Kriging) and simulate (Gaussian process generation) heterogenous fields in one, two or three dimensions, therefore providing necessary tools for geostatistical analysis.

Any such statistical analysis should draw on all available data that are connected to the variable of interest to infer, i.e. to learn about, its spatial distribution as much as possible. Examples for such spatially distributed variables in earth sciences would be, e.g. the hydraulic conductivity of an aquifer, evapotranspiration rates of different land surface areas, and soil moisture. In classical statistics, such information may consist of measurements of the variable itself or so-called local variables. Here, local means that a point-by-point relationship between both variables exists. However, many data are non-local, which means they are connected to the variable of interest via a complicated forward model. For instance, hydraulic conductivity may be connected by a solute transport model to break-through curves of said solutes and soil moisture may be connected by a hydraulic catchment model to river discharge. To learn about the input from the output of such forward models means to invert them, hence the name inversion for such techniques.

The Method of Anchored Distributions (MAD) provides a Bayesian framework for the geostatistical inversion of spatially heterogeneous variables. MAD solves the aforementioned problem by converting non-local data into equivalent local data using the tools of Bayesian inference. The result of such a conversion is the consistent representation of all data (local and non-local) as local data only, which is then amendable to further geostatistical analysis (Rubin et al., 2010). So far, applications of MAD have been focused on hydrogeology (Murakami et al., 2010; Chen et al., 2012; Heße et al., 2015) as well as soil science (Over et al., 2015). However, given the explanations above, MAD is in no way limited to these fields and can be employed wherever non-local data need to be incorporated into a geostatistical framework. This generality also extends to the spatial model being inferred. While there are R packages utilizing Bayesian inference for spatial models such as **spBayes** (Finley et al., 2015), **spTimer** (Bakar and Sahu, 2015), and **INLA** (Lindgren and Rue, 2015, software available from `http://www.r-inla.org/`), these packages have several constraints compared to **anchoredDistr**. First, each method assumes a Gaussian process for the spatial variability. MAD has no inherent distributional assumptions, which allows its application to a wide variety of scenarios where, for

example, Gaussian fields are not justified. In addition, these packages are either geared toward large data sets (**spBayes** and **spTimer**) or applied to only local data (**spBayes**, **spTimer**, and **INLA**) while MAD focuses on addressing uncertainty due to sparse data sets by incorporating non-local data. Finally, MAD employs a non-parameteric likelihood estimation, which allows for great flexibility, in particular for non-linear forward models. The presented R package **anchoredDistr** provides an interface to the C# implementation of MAD. It allows post-processing of calculating likelihood and posterior distributions as well as visualization of the data.

The Method of Anchored Distributions

Equation 1 displays the general procedure of Bayesian inference where θ represents the parameters of the variable being inferred (e.g. hydraulic conductivity) and z represents the data informing the inference:

$$p(\theta|z) \propto p(\theta)\, p(z|\theta)\,. \tag{1}$$

An important element of MAD is a strict classification of all data into local z_a and non-local data z_b, with the latter being the target of inversion. MAD employs Bayesian inference in the realm of geostatistics by expanding the supported parameters into θ for global parameters (describing overall trend and spatial correlation) and ϑ for local parameters. Since MAD is a Bayesian scheme, these θ and ϑ both have probability distributions. As mentioned above, MAD turns non-local data into equivalent local data ϑ by inverting the forward model that connects both. The non-local data therefore become anchored in space, hence the name Method of Anchored Distributions. Equation 2 displays the general form of MAD:

$$p(\theta, \vartheta|z_a, z_b) \propto p(\theta)\, p(\vartheta|\theta, z_a)\, p(z_b|\theta, \vartheta, z_a)\,. \tag{2}$$

Open-source software implementations for applying the entirety of MAD are available both with a graphical interface and a command-line interface to guide users through connecting their forward models and random field generators and to execute the ensemble of forward simulations (a. Osorio-Murillo et al., 2015). This software (available at `http://mad.codeplex.com`) was inspired by the claim that inverse modeling will be widely applied in hydrogeology only if user-friendly software tools are available (Carrera et al., 2005).

The package **anchoredDistr** described here focuses on extending the post-processing capabilities of MAD software, particularly the calculation of the likelihood distribution $p(z_b|\theta, \vartheta, z_a)$ and the posterior distribution $p(\theta, \vartheta|, z_b, z_a)$ after the ensemble of forward model simulations is already complete. The MAD# software has basic post-processing capabilities, but does not offer the degree of flexibility as R for the post-processing analysis. For example, when handling z_b in the form of time series, dimension reduction techniques are necessary for calculating the likelihood values. By having the R package **anchoredDistr**, users have the support to attach whichever applicable technique for their data.

General workflow

In the current version of **anchoredDistr**, which only handles the post-processing of a MAD application, it is assumed that prior distributions of local and global parameters, $p(\vartheta|\theta, z_a)$ and $p(\theta)$ respectively, have already been defined and sampled and that forward model simulations based on those samples have been executed either within the MAD# software or by other means of batch execution. If the MAD# software is used, this data is stored by MAD# in databases (extensions .xresult for project metadata and .xdata for each sample). The package **anchoredDistr** primarily consists of methods for the S4 class `"MADproject"` that extract and analyze data from these databases, i.e. handling information regarding the samples from the prior distributions and the resulting ensemble of simulated z_b data. If MAD# is not used, the information can be formatted into a `"MADproject"` manually. The usage of **anchoredDistr** will generally follow the workflow below (also see Figure 1):

1. Create `"MADproject"` object with `new()` function (passing slot information if manually filling data)
2. Read data from MAD# databases, if being used, into `"MADproject"` object with `readMAD()`
3. View the observations and realizations with `plotMAD()`
4. Apply any necessary dimension reduction techniques to z_b with `reduceData()`
5. Test the convergence of the likelihood distribution with respect to the number of realizations with `testConvergence()` (return to MAD software to run additional realizations if unsatisfactory)
6. Calculate likelihood and posterior distributions with `calcLikelihood()` and `calcPosterior()`, respectively

7. View the posterior distribution with `plotMAD()`.

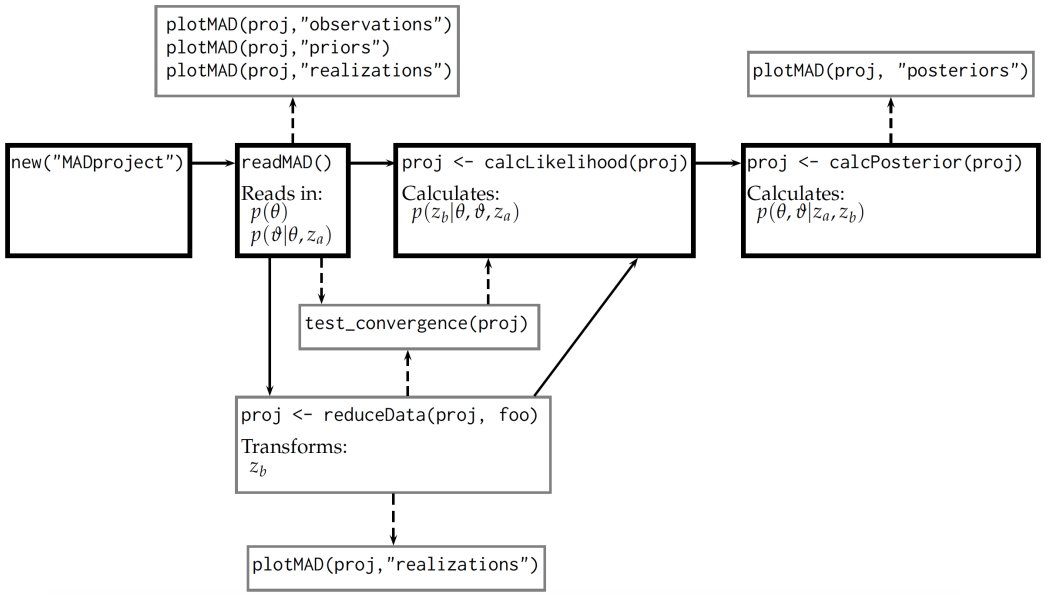


Figure 1: Schematic of utilizing **anchoredDistr** for MAD post-processing if the MAD# is used. Solid arrow lines indicate the fundamental workflow while dashed arrow lines are optional.

To install the **anchoredDistr** package, the release version is available from CRAN:

```
install.packages("anchoredDistr")
library(anchoredDistr)
```

Alternatively, the development version can be obtained by using the **devtools** package (Wickham and Chang, 2016) to download the necessary files from GitHub:

```
library(devtools)
install_github("hsavoy/anchoredDistr")
library(anchoredDistr)
```

Other packages used by **anchoredDistr** include **RSQLite** (Wickham et al., 2014) for reading from MAD databases, **np** (Hayfield and Racine, 2008) for estimating non-parametric density distributions, **plyr** (Wickham, 2011) and **dplyr** (Wickham and Francois, 2016) for efficient data manipulation, and **ggplot2** (Wickham, 2009) for plotting. The methods included in **anchoredDistr** are listed in Table 1 and two examples utilizing these methods are provided next.

Method	Description
`readMAD()`	Reads data from databases generated by MAD software
`reduceData()`	Applies dimension reduction to z_b time series
`testConvergence()`	Tests for convergence of likelihood values for increasing number of realizations
`calcLikelihood()`	Calculates the likelihood values for the samples
`calcPosterior()`	Calculates the posterior values for the samples
`plotMAD()`	Plots the observations, realizations, reduced data, and/or posteriors

Table 1: The methods for the `"MADproject"` S4 class provided by **anchoredDistr**.

Example 1: aquifer characterization with steady-state hydraulic head from multiple wells

Scenario setup

In this first example, we will use the tutorial example available from the MAD website `http://mad.codeplex.com`. Within the **anchoredDistr** package, this tutorial example is available as MAD# databases, as well as a `"MADproject"` object accessed by `data(tutorial)`. The variable of interest is transmissivity T, an aquifer property that represents how much water can be transmitted horizontally through an aquifer. We will use the one-dimensional heterogenous field of the decimal log transform of T (see Figure 2) as our baseline field from which we can generate virtual measurements and validate our resulting posterior distributions. The field was generated as a Gaussian process by the **gstat** package in R with a mean $\mu_{\log_{10} T} = -2$ and an exponential covariance function with a variance $\sigma^2_{\log_{10} T} = 0.4$ and length scale $l_{\log_{10} T} = 3$ m. Within the scope of this example, we assume these global parameter values to be known. Furthermore, we assume that we have local data in the form of measurements of T at three different locations. In addition, non-local data are available in the form of head measurements (indication of water pressure) at the same locations. The forward model used to solve the groundwater flow equation and relate T to head is the software MODFLOW-96 (Harbaugh and Mcdonald, 1996), part of the open source MODFLOW series that is the industry standard for groundwater modeling. To convert the non-local data into equivalent local data of T, we will place four anchors at selected unmeasured locations. The number of anchors needs to be justified by the data content of the measurements such that the complexity of the model does not become disproportionate to the information available. The locations of these anchors reflect locations where there is no other local data available but there is non-local data nearby for conversion (see Yang et al. (2012) for more discussion on anchor placement). The locations of the measurements and anchors are depicted in Figure 2. The prior distributions for these anchors are based on simple kriging with the local data z_a for conditioning and the known Gaussian process for the covariance function:

$$p\left(\vartheta_i|\theta, z_a\right) = \mathcal{N}\left(\mu = \hat{Z}\left(y_i\right), \sigma^2 = Var\left(Z\left(y_i\right) - \hat{Z}\left(y_i\right)\right)\right), \tag{3}$$

where Z generally represents $\log_{10} T$, y_i is the y-coordinate of the i^{th} anchor, $\hat{Z}\left(y_i\right)$ is the kriging estimate at the i^{th} anchor , and $Var\left(Z\left(y_i\right) - \hat{Z}\left(y_i\right)\right)$ is the kriging variance at the i^{th} anchor. The goal of the example is to compare the posterior distributions of the four anchors resulting from the inversion to their prior distributions which will indicate the information gain from the inclusion of the non-local data z_b.

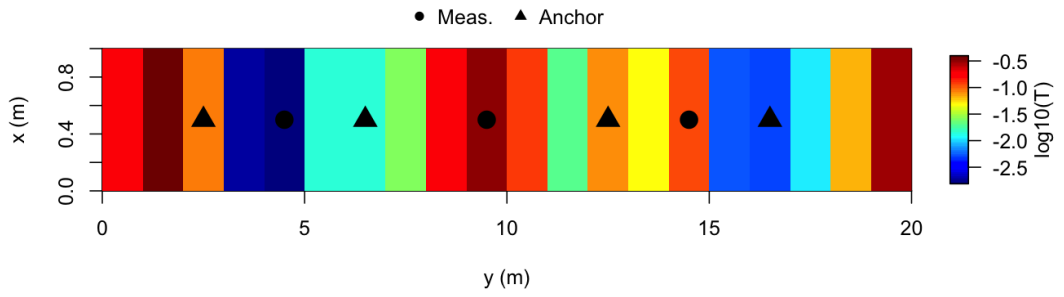


Figure 2: The one-dimensional baseline field of $\log_{10} T$ used in Example 1 with locations of measurements (co-located z_a and z_b) marked along with the anchors to be inferred.

Reading and viewing data

In the first step, a `"MADproject"` object is created with the `new()` function. Three arguments must be provided to read the MAD databases: `madname` (the name of the MAD project, e.g. the filename for the .xmad database), `resultname` (the name of the result from MAD, e.g. the result folder name), and `xpath` (the path to where the .xresult database and result folder are located). These three arguments ensure the MAD databases can be read by the method `readMAD()`, which will read in the prior distribution samples for the global and local parameters plus the observations and forward model predictions for the z_b. Note that **anchoredDistr** could be used independently of the MAD software, if desired, as long

as the slots filled in by `readMAD()` (see Table 2) are provided manually (see next example). To create a `"MADproject"` object for this tutorial example, the code below will read the MAD# databases stored in the **anchoredDistr** package files.

```
tutorial <- new("MADproject", madname="Tutorial", resultname="example1",
           xpath=paste0(system.file("extdata", package = "anchoredDistr"),"/"))
tutorial <- readMAD(tutorial, 1:3)
```

Slot	Description	Source
`madname`	MAD project name	user provided
`resultname`	MAD result name	user provided
`xpath`	Path to .xresult database	user provided
`numLocations`	Number of z_b locations	`readMAD()`
`numTimesteps`	Number of time steps measured at each z_b locations	`readMAD()`
`numSamples`	Number of samples drawn from prior distributions	`readMAD()`
`numAnchors`	Number of local parameters / anchors placed in field	`readMAD()`
`numTheta`	Number of random global parameters to infer	`readMAD()`
`truevalues`	True values for the parameters to infer, if known	`readMAD()`
`observations`	Observed values of the z_b locations and time steps	`readMAD()`
`realizations`	Simulated values of the z_b locations and time steps	`readMAD()`
`priors`	Samples from the prior distributions of each parameter	`readMAD()`
`likelihoods`	Likelihood values for each sample	`calcLikelihoods()`
`posteriors`	Posterior values for each sample of each parameter	`calcPosteriors()`

Table 2: The slots for the `"MADproject"` S4 class provided by **anchoredDistr**.

The prior distributions can be viewed by calling the `plotMAD()` function with the `"MADproject"` object and the string "priors" (see below). Figure 3 shows the prior distributions for the four anchors in Example 1. The distributions roughly follow a Gaussian distribution due to the baseline field being a Gaussian field and the prior distributions based on the kriging mean and variance at these four locations from the z_a data and the known spatial random function. The x-axis labels are pulled from the `"MADproject"` object's `priors` slot, which contains the random parameter names as provided in the MAD software.

```
plotMAD(tutorial, "priors")
```

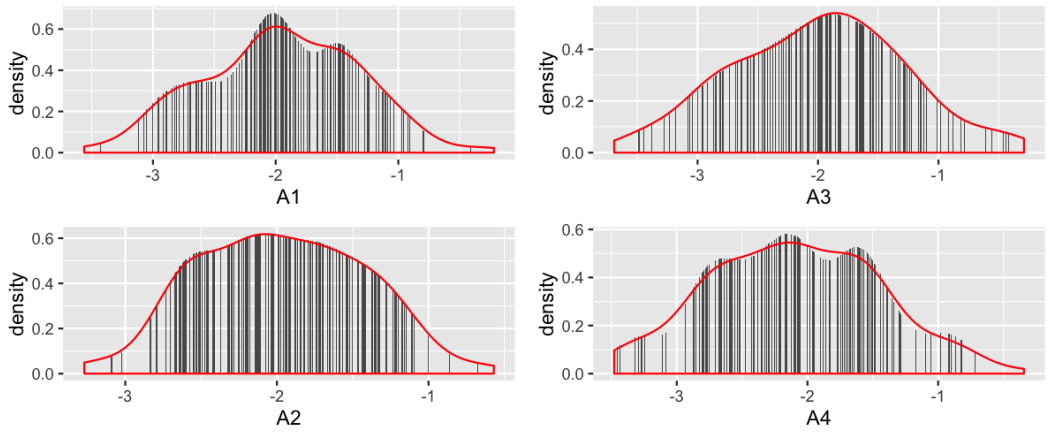


Figure 3: The relative frequency (gray bars) and estimated density (red line) of the prior distributions for the four anchor locations based on samples supplied in Example 1.

Calculating likelihoods and posteriors

After the information contained in the MAD databases has been read into the `"MADproject"` object, the likelihood and posterior distributions can be calculated by `calcLikelihood()` and `calcPosterior()`, respectively. The method `calcLikelihood()` uses non-parametric kernel density estimation (from the

package **np**) to estimate the probability density of measured inversion data from the probability density function of inversion data simulated from the realizations per sample. The method `calcPosterior()` multiplies the resulting likelihood distribution across the samples and the provided prior distribution to calculate the posterior.

First, we can call the `testConvergence()` method to visually inspect if we have enough realizations for the likelihood values of samples to converge (this method calls the `calcLikelihood()` internally to perform this test). Figure 4 depicts this qualitative convergence test for Example 1 by plotting the likelihood values of a sample with increasing number of realizations. In order to prevent cluttering, the default number of samples to display is set to seven samples randomly selected from those available in the project. Convergence is achieved when the likelihood stabilizes with increasing realizations. For this example, it appears that the log likelihood of the samples have started to stabilize by 50 realizations, but more realizations may be warranted.

The posterior distributions for each random parameter can be seen by calling `plotMAD()` with the `"MADproject"` object and the string "posteriors". Figure 5 shows the posteriors for Example 1 along with the prior distribution and the true values for each of the four anchors. The posterior distributions for Anchors 2 and 3, which were surrounded by z_b measurements, show an increase in probability near the true value, indicating a successful information transfer from the non-local z_b into equivalent local data.

```
testConvergence(tutorial)
tutorial <- calcLikelihood(tutorial)
tutorial <- calcPosterior(tutorial)
plotMAD(tutorial, "posteriors")
```

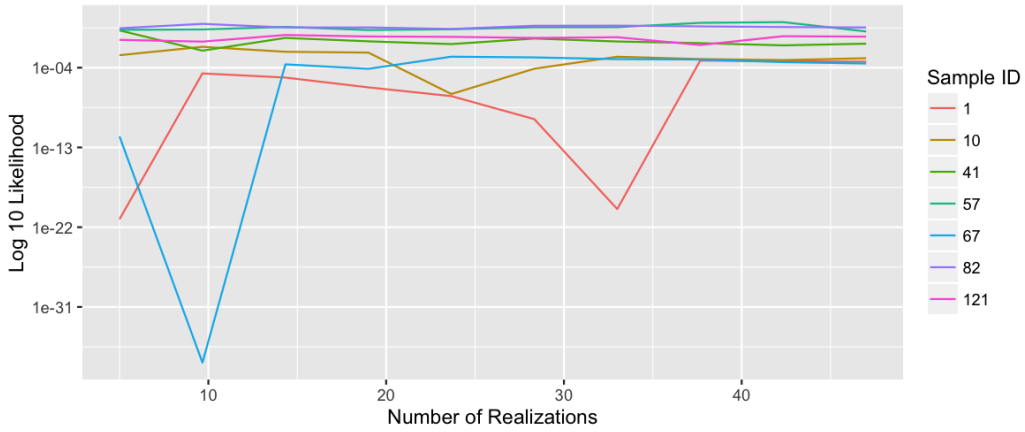


Figure 4: Convergence testing for Example 1 by plotting the decimal log of likelihood of a collection of randomly selected samples wth increasing number of realizations.

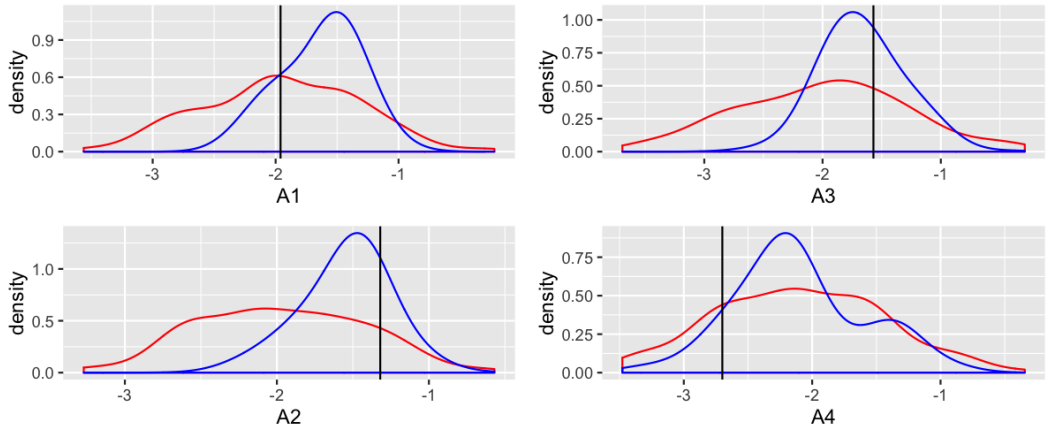


Figure 5: The prior (red) and posterior (blue) distributions with the true value (black) for the four anchor locations in Example 1.

Example 2: aquifer characterization with one pumping drawdown curve

Scenario setup

The second example depicts a different aquifer characterization scenario for a two-dimensional field where the natural log transform of hydraulic conductivity (K) is assumed to be an isotropic Gaussian field with variance $\sigma^2_{\ln K} = 1$ and length scale $l_{\ln K} = 10$m but unknown mean $\mu_{\ln K}$ (Figure 6). There are no anchors placed in this example, leaving the mean as the only parameter to infer. Unlike Example 1, Example 2 is therefore a demonstration of how MAD can be employed as a regular Bayesian inversion scheme, too. The prior distribution for global parameters ideally come from previous knowledge of similar sites, e.g. the distribution of mean $\ln K$ observed at other aquifers with the same geological setting. For this example, we will compare three equally spaced samples for $\ln K$ to represent a uniform prior distribution for the mean. The data include four local data z_a (K) at four different locations and one non-local data series z_b (hydraulic head drawdown) at a single location (see Figure 6). The z_b location provides 100 time steps, i.e. data points, of drawdown measurements (Figure 7). The forward model used to solve the groundwater flow equation and relate K to drawdown is OpenGeoSys (Kolditz et al., 2012), an open source software that simulates a variety of subsurface processes. This second example uses a different forward model than the first example to showcase the MAD software's modular design, which does not assume or rely on specific forward models. The observation, realizations, and prior sample data for this example is provided within the package as external data that can be created with `new()` as shown below, as well as a pre-made `"MADproject"` object accessed with `data(pumping)`.

```
load(system.file("extdata", "pumpingInput.RData", package = "anchoredDistr"))
pumping <- new("MADproject",
           numLocations = 1,
           numTimesteps = 100,
           numSamples   = 50,
           numAnchors = 0,
           numTheta = 1,
           observations = obs,
           realizations = realizations,
           priors = priors)
```

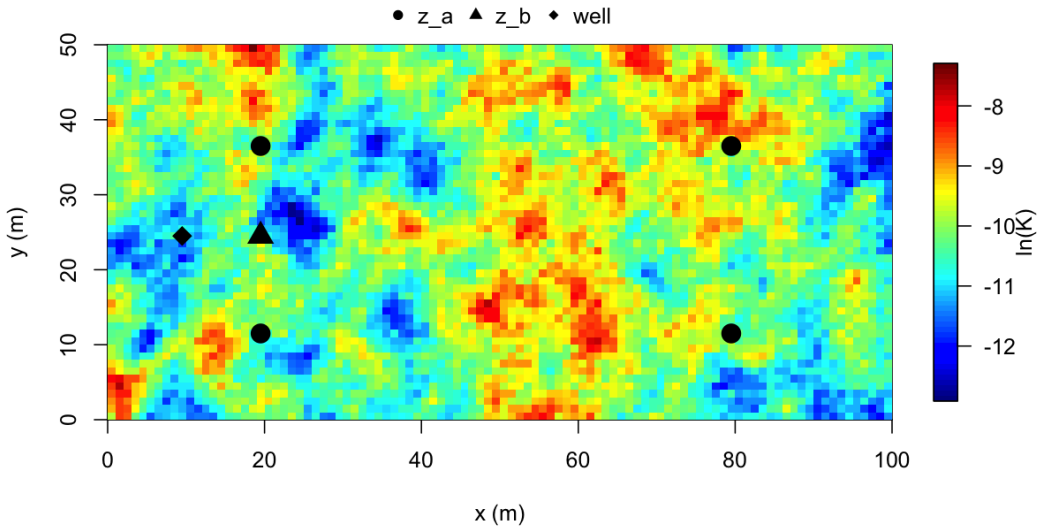


Figure 6: The two-dimensional baseline field of $\ln K$ used in Example 2 with the location of measurements marked.

When the `pumping` dataset is initially loaded, we can view the observation of z_b, i.e. drawdown time series (Figure 7), the prior distribution of the three samples (Figure 8), and the interquartile range of the time series simulated by the forward model for the samples (Figure 9).

```
plotMAD(pumping, "observations")
```

```
plotMAD(pumping, "priors")
plotMAD(pumping, "realizations")
```

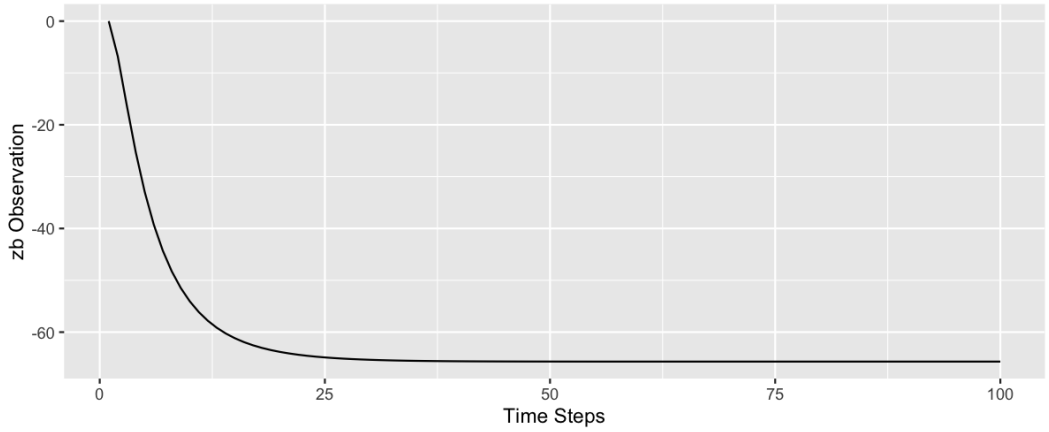


Figure 7: The observed time series of hydraulic head drawdown to be used as non-local data z_b in Example 2.

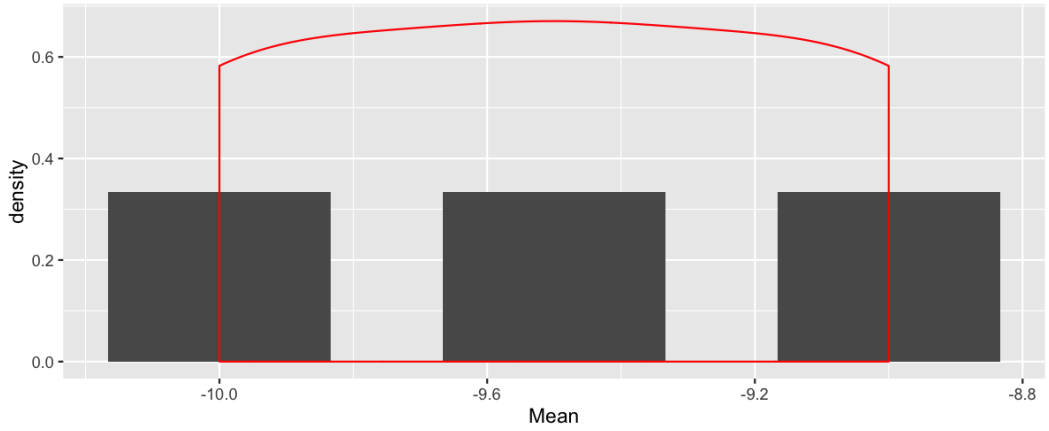


Figure 8: The histogram (gray bars) and estimated density (red line) of the prior distributions for the mean ln K Example 2.

Applying dimension reduction to time series

Even though we have the time series of drawdown, we cannot use these 100 individual values to calculate the likelihood because they are correlated and the multivariate likelihood distribution would be 100-dimensional. Such dimensionality would require an unrealistic number of realizations to resolve, known as "the curse of dimensionality." To overcome this obstacle, dimension reduction is needed and the method to use depends on the type of non-local data z_b. For this example, we will simply use the `min()` function to collect the minimum head value in the time series since the observed head reduces and converges to a stable head value with time (Figure 7). The **anchoredDistr** package can handle any non-parameterized function, such as `min()`, or a parameterized function if initial values for each parameter are given and the `nls()` function (R Core Team, 2016) can perform the fitting (see the package vignette for an example). The `reduceData()` function is used to perform the dimension reduction on the time series:

```
pumping.min <- reduceData(pumping, min)
plotMAD(pumping.min, "realizations")
```

The `reduceData()` function returns a `"MADproject"` object with a `realizations` slot with reduced dimensions. The reduced data can be viewed by calling `plotMAD()` with the string "realizations". The plot shows the distributions of each parameter for each sample. In this case, Figure 10 shows the

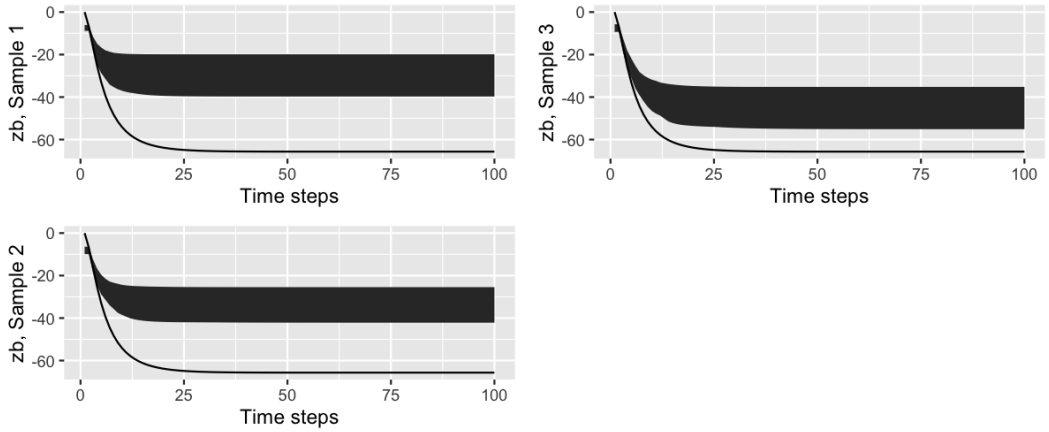


Figure 9: The observed time series of drawdown at the z_b location along with the inter-quartile range of simulated values for each time step for the three samples.

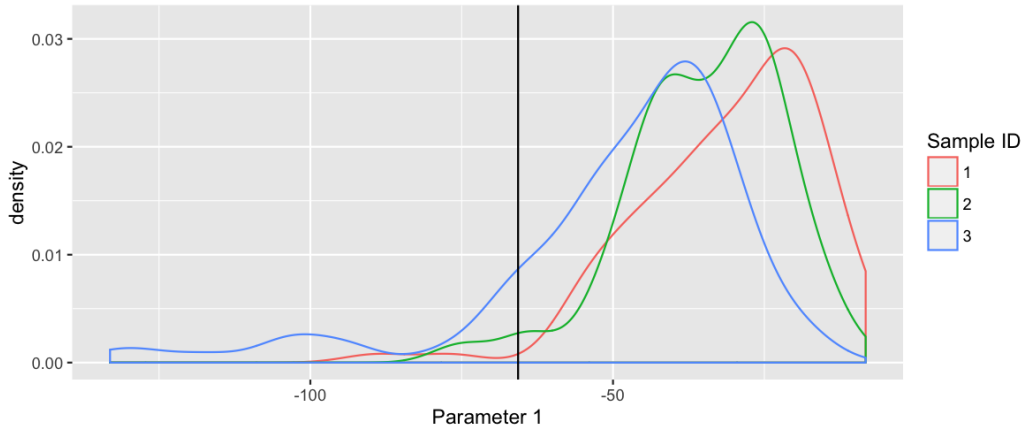


Figure 10: The reduced z_b data (minimum of drawdown curve) for Example 2. Distributions are estimated from the realizations' reduced data per sample.

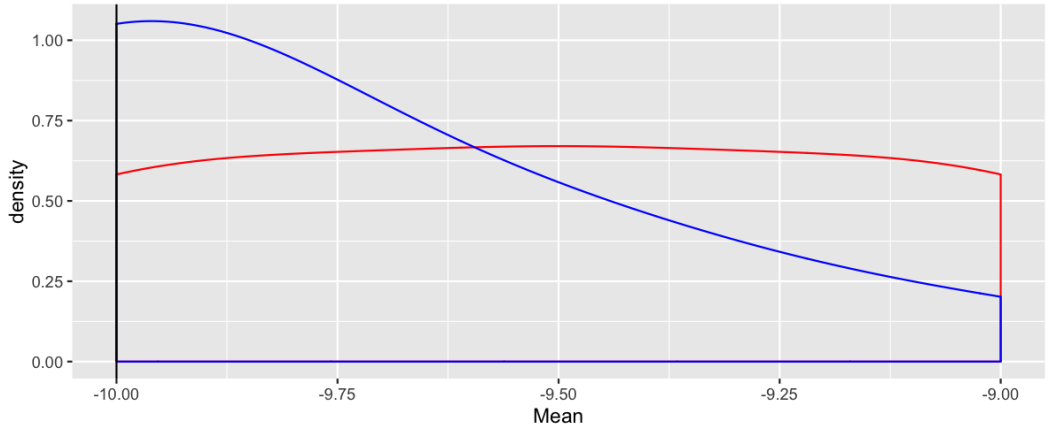


Figure 11: The prior (red) and posterior (blue) distributions with the true value (black) for the mean ln K locations in Example 2.

minimum head value distribution for the three samples, which will be used to calculate the three likelihood samples.

With this new "MADproject" object, calcLikelihoods() and calcPosteriors() can be called. In Figure 11, the posterior distributions are shown for the three samples along with the true value of −10. The posterior distribution assigns greater probability toward the true value.

```
pumping.min <- calcLikelihood(pumping.min)
pumping.min <- calcPosterior(pumping.min)
plotMAD(pumping.min, "posteriors")
```

Summary

The examples given above show how the **anchoredDistr** package allows flexible post-processing of results by virtue of the MAD software such that users can apply their own post-processing analyses, such as dimension-reduction techniques. The first example shown here is available as external and internal datasets in the **anchoredDistr** package. The second example is also included in **anchoredDistr** and is further detailed in the package vignette. The release version of the **anchoredDistr** package is hosted on CRAN and the development version is hosted on GitHub, which can be accessed by calling devtools::install_github("hsavoy/anchoredDistr") or by downloading from http://hsavoy.github.io/anchoredDistr.

Acknowledgements

This work was supported by the National Science Foundation under grant EAR-1011336, "The Method of Anchored Distributions (MAD): Principles and Implementation as a Community Resource." Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

Bibliography

C. a. Osorio-Murillo, M. W. Over, H. Savoy, D. P. Ames, and Y. Rubin. Software framework for inverse modeling and uncertainty characterization. *Environmental Modelling & Software*, 66:98–109, 2015. ISSN 13648152. URL https://doi.org/10.1016/j.envsoft.2015.01.002. [p7]

K. S. Bakar and S. K. Sahu. spTimer: Spatio-temporal Bayesian modeling using R. *Journal of Statistical Software*, 63(15):32, 2015. ISSN 1548-7660. [p6]

J. Carrera, A. Alcolea, A. Medina, J. Hidalgo, and L. J. Slooten. Inverse problem in hydrogeology. *Hydrogeology Journal*, 13(1):206–222, 2005. ISSN 14312174. URL https://doi.org/10.1007/s10040-004-0404-7. [p7]

X. Chen, H. Murakami, M. S. Hahn, G. E. Hammond, M. L. Rockhold, J. M. Zachara, and Y. Rubin. Three-dimensional Bayesian geostatistical aquifer characterization at the Hanford 300 Area using tracer test data. *Water Resources Research*, 48(6):W06501, 2012. ISSN 0043-1397. URL https://doi.org/10.1029/2011wr010675. [p6]

A. O. Finley, S. Banerjee, and A. E. Gelfand. spBayes for large univariate and multivariate point-referenced spatio-temporal data models. *Journal of Statistical Software*, 63(13):1–28, 2015. ISSN 1548-7660. URL http://www.jstatsoft.org/v63/i13. [p6]

P. Goovaerts. Geostatistics in soil science: State-of-the-art and perspectives. *Geoderma*, 89(1–2):1 – 45, 1999. ISSN 0016-7061. URL https://doi.org/10.1016/s0016-7061(98)00078-0. [p6]

W. Harbaugh and M. G. Mcdonald. User's documentation for MODFLOW-96 , an update to the U.S. Geological Survey modular finite-difference ground-water flow model, Open File Report 96-485. Technical report, U.S. Geological Survey, 1996. [p9]

T. Hayfield and J. S. Racine. Nonparametric econometrics: The np package. *Journal of Statistical Software*, 27(5), 2008. URL http://www.jstatsoft.org/v27/i05/. [p8]

F. Heße, H. Savoy, C. A. Osorio-Murillo, J. Sege, S. Attinger, and Y. Rubin. Characterizing the impact of roughness and connectivity features of aquifer conductivity using Bayesian inversion. *Journal of Hydrology*, 531:73–87, 2015. ISSN 00221694. URL https://doi.org/10.1016/j.jhydrol.2015.09.067. [p6]

M. Hohn. *Geostatistics and Petroleum Geology (2nd Ed.)*. Kluwer, 1962. [p6]

P. Kitanidis. *Introduction to Geostatistics: Applications in Hydrogeology*. Cambridge University Press, 2008. [p6]

O. Kolditz, S. Bauer, L. Bilke, N. Böttcher, J. O. Delfs, T. Fischer, U. J. Görke, T. Kalbacher, G. Kosakowski, C. I. McDermott, C. H. Park, F. Radu, K. Rink, H. Shao, H. B. Shao, F. Sun, Y. Y. Sun, A. K. Singh, J. Taron, M. Walther, W. Wang, N. Watanabe, Y. Wu, M. Xie, W. Xu, and B. Zehner. OpenGeoSys: An open-source initiative for numerical simulation of Thermo-Hydro-Mechanical/chemical (THM/C) processes in porous media. *Environmental Earth Sciences*, 67(2):589–599, 2012. ISSN 1866-6280. URL https://doi.org/10.1007/s12665-012-1546-x. [p12]

D. G. Krige. A statistical approach to some basic mine valuation problems on the Witwatersrand. *Journal of the Chemical, Metallurgical and Mining Society of South Africa*, 52(6):119–139, 1951. URL https://doi.org/10.2307/3006914. [p6]

F. Lindgren and H. Rue. Bayesian spatial modelling with R-INLA. *Journal of Statistical Software*, 63(19): 1–25, 2015. ISSN 1548-7660. URL https://doi.org/10.18637/jss.v063.i19. [p6]

G. Matheron. *Traité De Géostatistique Appliquée*, volume 14. Editions Technip, Paris, 1962. [p6]

H. Murakami, X. Chen, M. S. Hahn, Y. Liu, M. L. Rockhold, V. R. Vermeul, J. M. Zachara, and Y. Rubin. Bayesian approach for three-dimensional aquifer characterization at the Hanford 300 Area. *Hydrology and Earth System Sciences*, 14(10):1989–2001, 2010. ISSN 1607-7938. URL https://doi.org/10.5194/hess-14-1989-2010. [p6]

M. W. Over, U. Wollschlaeger, C. a. Osorio-Murillo, and Rubin. Bayesian inversion of Mualem-Van Genuchten parameters in a multilayer soil profile: A data-driven assumption-free likelihood function. *Water Resources Research*, 51(2):861–884, 2015. URL https://doi.org/10.1002/2013wr014956.received. [p6]

E. J. Pebesma. Multivariable geostatistics in S: The gstat package. *Computers & Geosciences*, 30:683–691, 2004. [p6]

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2016. URL https://www.R-project.org/. [p13]

Y. Rubin, X. Chen, H. Murakami, and M. Hahn. A Bayesian approach for inverse modeling, data assimilation, and conditional simulation of spatial random fields. *Water Resources Research*, 46 (October 2009):1–23, 2010. ISSN 00431397. URL https://doi.org/10.1029/2009wr008799. [p6]

H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag, 2009. ISBN 978-0-387-98140-6. URL http://ggplot2.org. [p8]

H. Wickham. The split-apply-combine strategy for data analysis. *Journal of Statistical Software*, 40(1): 1–29, 2011. URL http://www.jstatsoft.org/v40/i01/. [p8]

H. Wickham and W. Chang. *devtools: Tools to Make Developing R Packages Easier*, 2016. URL https://CRAN.R-project.org/package=devtools. R package version 1.11.1. [p8]

H. Wickham and R. Francois. *dplyr: A Grammar of Data Manipulation*, 2016. URL https://CRAN.R-project.org/package=dplyr. R package version 0.5.0. [p8, 292]

H. Wickham, D. A. James, and S. Falcon. *RSQLite: SQLite Interface for R*, 2014. URL https://CRAN.R-project.org/package=RSQLite. R package version 1.0.0. [p8]

Y. Yang, M. Over, and Y. Rubin. Strategic placement of localization devices (such as pilot points and anchors) in inverse modeling schemes. *Water Resources Research*, 48(8):W08519, 2012. ISSN 00431397. URL https://doi.org/10.1029/2012wr011864. [p9]

Heather Savoy
Civil and Environmental Engineering
University of California, Berkeley
Berkeley, CA, USA
frystacka@berkeley.edu

Falk Heße
Computational Hydrosystems

Helmholtz Centre for Environmental Research (UFZ)
Leipzig, Germany
falk.hesse@ufz.de

Yoram Rubin
Civil and Environmental Engineering
University of California, Berkeley
Berkeley, CA, USA
rubin@ce.berkeley.edu

dGAselID: An R Package for Selecting a Variable Number of Features in High Dimensional Data

by Nicolae Teodor Melita and Stefan Holban

Abstract The **dGAselID** package proposes an original approach to feature selection in high dimensional data. The method is built upon a diploid genetic algorithm. The genotype to phenotype mapping is modeled after the Incomplete Dominance Inheritance, overpassing the necessity to define a dominance scheme. The fitness evaluation is done by user selectable supervised classifiers, from a broad range of options. Cross validation options are also accessible. A new approach to crossover, inspired from the random assortment of chromosomes during meiosis is included. Several mutation operators, inspired from genetics, are also proposed. The package is fully compatible with the data formats used in **Bioconductor** and **MLInterfaces** package, readily applicable to microarray studies, but is flexible to other feature selection applications from high dimensional data. Several options for the visualization of evolution and outcomes are implemented to facilitate the interpretation of results. The package's functionality is illustrated by examples.

Introduction

Recent advances in information technology provide tools for gathering an immense amount of data on various scopes. Investigators have increasingly improved tools to collect data describing different areas of research. The need to efficiently manage, analyze and extract the important features from high dimensional data is also growing with the different exploration areas benefiting from these technologies.

The DNA microarray technology is widely used for exploring the differential gene expression. The method is very established and offers a very good opportunity to develop new methods for selecting features in real high dimensional data. The vast amount of microarray data that is freely available along with the results obtained by employing other exploratory techniques, belonging to statistics or artificial intelligence, provide a unique opportunity to evaluate the performance of newly developed techniques.

The Genetic Algorithms (GAs) were extensively used to select features in various high dimensional data, for different research goals. The GA designs evolved and were adapted with particular exploration interests since they were introduced (Holland, 1975). Different GA designs were specifically adapted to address optimizations or diverse feature selection (Xue et al., 2016) assignments.

The literature on GAs is comprehensive and covers various aspects of interest. The fundamentals of GAs, including the schema theorem, are covered in very instructive introductory books (Mitchell, 1998) and (Goldberg, 1989). Other authors propose exhaustive investigations in the GAs' behavior (Berard and Bienvenue, 2003) and properties (Rudolph, 1994). The genetic operators and their impact on evolution, with emphasis on mathematical details (Doerr and Doerr, 2015) were extensively examined. The genetic algorithms model the naturally occurring evolution and are designed to solve particular problems. In consequence, the theoretical foundations are yet to catch up with the practically applied algorithms. Nevertheless, the theory of genetic algorithms is emerging (Droste et al., 2002).

The project R offers a great environment for developing methods for high dimensional data analysis. The variety of techniques already implemented by numerous contributors and the availability of the methods, source code, countless data, and results as well as the very forthcoming community make it the environment of choice for implementing our method. Moreover, the Bioconductor project (Huber et al., 2015) available in R, offers a wide range of methods and tools for analyzing microarray data, as well as real data sets to experiment with and compare the results. Our package **dGAselID** was developed to be cohesive with Bioconductor. The `"ExpressionSet class"` used in Bioconductor was adopted as standard for our package; any data formatted accordingly can be analyzed with our method.

Different GA implementations are available as contributors' packages in R. Implementations of GAs for both floating-point and binary chromosomes are included in the **genalg** (Willighagen and Ballings, 2015) package. The **GA** (Scrucca, 2016), **nsga2R** (Tsou, 2015), and **gaoptim** (Tenorio, 2013) packages are dedicated to optimizations using GAs. A GA designed for determining training populations (Akdemir et al., 2015) is offered in the **STPGA** package. The package **kofnGA** (Wolters, 2015) aims to select a fixed-size set of integers. Variable selection applications of GAs are proposed in the

mogavs (Pajala, 2016) and **gaselect** (Kepplinger, 2015) packages for regression and high-dimensional data respectively.

Algorithm

Haploid GAs were previously employed to address feature selection in microarray studies (Melita et al., 2008). In this type of data, the number of samples is significantly lower than the number of features and the utilization of cross validation techniques is necessary for reliable results. The diploid GAs offer better performance than the haploid implementations for selecting features in a cross validation scenario in general, and for microarray data (Melita and Holban, 2016b) in particular.

The GA implementation in the **dGAselID** package uses a diploid representation. All the features in the data form a genome, and each feature retains a specific locus in the genome, the position in the original data. Every individual in the population will consist of two such genomes.

The fitness evaluation function is a supervised classifier. Any implementation of supervised classifier available in **MLInterfaces** package (Carey et al., 2016) is a possible choice for fitness evaluation function in **dGAselID**. The fitness value is the accuracy of the given classifier in discerning between samples belonging to different classes.

Every feature in the data, inputted according to the format in the `"ExpressionSet class"`, is represented by a gene in the genome. Every gene has two alleles, represented as `0` and `1`. The allele `1` codes for the corresponding feature to be present in the classifier. The allele `0` cyphers for discarding the gene from the classifier. A genome with a limited number of alleles = `1`, codes for the supervised classifier working on a subset of features from the data. The number of desired features is user selectable at the initialization of the algorithm.

Our implementation offers the possibility to divide the genomes into a variable number of chromosomes. The number of chromosomes to split the genomes in is user selectable. The value `1` for the number of chromosomes will result in the genome being treated as a single chromosome, like in the classical GA implementation. The default value in the **dGAselID** is `22`. In this case, the genome is parted into 22 chromosomes, the total of human autosomes. The chromosomes will have variable length, with different number of genes, following the dispersal found in the human autosomes, as illustrated in the Table 1. The number of genes found on each chromosome will follow the spread found in the human autosomes with different values for the number of chromosomes. We chose the default value `22` to emphasize the foundation of our evolutionary approach. Different values will serve diverse practical applications. This parameter is particularly important when variables belong to several previously known categories, as with the custom microarray chips. When no such information is known, an appropriate value can be empirically determined.

The initial population is randomly generated from a discrete uniform distribution. The user can specify the number of genomes in the population, the number of activated genes in each genome and the number of chromosomes to split the genomes in. The population will encompass individuals, with each individual consisting of two sets of haploid chromosomes, randomly assigned.

In a diploid GA it is mandatory to determine how different alleles on heterozygous chromosomes influence the phenotype. The dominance schemes typically used in genetic algorithms are built upon the Complete Dominance model, described in biology by Gregor Mendel in 1865. In this model, one of the alleles, called dominant, produces effects into phenotype and masks the existence of the other allele in genotype. The alternative that does not affect the phenotype is called recessive allele. The Complete Dominance model describes only a few of the interactions between alleles in nature. Various models were later developed to describe different interactions between alleles. In Incomplete Dominance model, the phenotype of an individual is considered to be in between the phenotypes resulting from each of the inherited alleles. Both alleles influence the phenotype and the existence of none is masked in genotype. The main difference between the two models is illustrated in Figure 1. We can suppose that a gene that codes for the color of an organism has two alleles; one of them produces a red individual and the alternative shapes a blue entity. With the Complete Dominance model, one of the alleles is dominant and masks the presence of the other. In our example, the allele that codes for the red color is dominant and is noted with capital letter (R). Every diploid organism that inherits at least one allele R will be a red entity. Only if a diploid organism inherits two copies of the recessive allele (b), the phenotype will be influenced by it, resulting in a blue individual. The interaction described by the Incomplete Dominance model results in three different phenotypes. In this case, an individual can be red, blue or purple. Both alleles affect the phenotype and are noted with capital letters (R and B).

The Incomplete Dominance inheritance is an alternative to genotype to phenotype dominance schemes (Melita and Holban, 2016b) in genetic algorithms and is the approach adopted in our algorithm. Moreover, this method does not require an explicit scheme for genotype-phenotype mapping. The fitness of each individual is consequently evaluated as the mean accuracy of the two classifiers,

Chromosome No.	No. of features
1	9.17%
2	7.64%
3	5.81%
4	4.89%
5	5.19%
6	5.81%
7	5.50%
8	4.28%
9	4.28%
10	4.28%
11	6.11%
12	4.89%
13	2.44%
14	3.66%
15	3.66%
16	3.97%
17	4.89%
18	1.83%
19	5.19%
20	2.75%
21	1.22%
22	2.44%

Table 1: Default distribution of features on chromosomes.

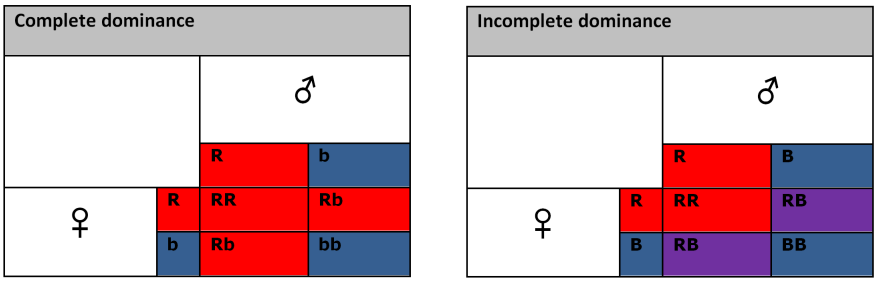


Figure 1: Complete vs. Incomplete Dominance.

each set of haploid chromosomes in the individual. With this approach it is possible to maintain a better variability in the late generations, as features from the poorly performing genotypes will be present in later generations when compared to the classical technique. Exploitation of the Incomplete Dominance model is optional. The value `"ID2"` for the parameter `ID` enables the Incomplete Dominance model, while the value `"ID1"` turns it off. In evaluating the fitness of an individual, any supervised classifier available in **MLInterfaces** package can be selected. The package offers a unified way to call supervised classifiers on data formatted according to `"ExpressionSet class"` specifications with several options (`svmI`, `ldaI`, `rdaI`, `knnI`, `knn.cvI`, `randomForrestI`, `dldaI`, `nnetI`, `qdaI`, `naiveBayesI`, etc). The cross-validation techniques implemented in **MLInterfaces** are also available for fitness evaluation in our package, and are accessible through the `trainTest` parameter. The default value `"LOG"` for the `trainTest` parameter enables the leave-out-group cross-validation, while specifying `"LOO"`, the user can opt for the leave-one-out cross-validation. Moreover, is if possible to shape the data into training and testing sets. A value of the form `x:y` for the `trainTest` parameter identifies the samples with indexes from `x` to `y` as training examples while all the others are used as testing set.

In the next step, the individuals are ranked according to their fitness. Our implementation offers the option of applying an elitist selection over the ranked individuals. The default value for elitism is NA, but a user can decide on the desired value for elitism, keeping the chosen number of best performing genotypes in the population.

Crossovers are applied next, between the two haploid sets of chromosomes in each individual. Two-point crossovers were preferred to the single-point alternative which preferentially affects the string ends. The two-point crossover follows the classical implementation. One two-point crossover is applied between homologous chromosomes, in each individual. A parameter to explicitly specify the chance for a crossover to occur is not implemented in our algorithm. The number of chromosomes implicitly affects the number of crossovers.

Another approach to crossover, modeled after the Random Assortment of Chromosomes in meiosis is also available. The Random Assortment of Chromosomes Crossover (Melita and Holban, 2016a) takes advantage of splitting the genomes in a number of chromosomes with variable size. When selected, two-point crossovers are performed between homologous chromosomes. After the crossovers are applied, the chromosomes are randomly assorted and distributed to one of the chromosomes set in each individual. This process, models the events that occur during meiosis I in eukaryotes. The user can choose at the initialization of the algorithm if the chromosomes will be randomly assorted through the `randomAssortment` parameter. The default value is `TRUE`. This operator is especially important when selecting a small number of features from a very large poll. In this case, the two-point crossover frequently recombines strings containing only zeros, with no effect on the very long genotypes. The Random Assortment operator offers a significant advantage in these situations. The distinctions between these approaches to recombination are highlighted in Figure 2, a part of the R output using the code:

```
> library(dGAselID)
> set.seed(1357)
> c1<-rep(0, 10)
> c2<-rep(1, 10)
> individual<-rbind(c1, c2)
> individual

> chrConf01<-rep(1, 10)
> chrConf01

> #Two-point crossover on genotypes with 1 chromosome
> Crossover(individual [1, ], individual [2, ], chrConf01)

> chrConf03<-c(rep(1, 4), rep(2, 3), rep(3, 3))
> chrConf03

> #Two-point crossovers on genotypes with 3 chromosomes
> cr3<-Crossover(individual [1, ], individual [2, ], chrConf03)
> cr3

> #Random Assortment on the recombined genotypes with 3 chromosomes
> RandomAssortment(cr3, chrConf03)
```

Subsequently, the haploid sets of chromosomes with a higher fitness are kept and the others are discarded from each individual and mutations are applied with a chance that is specified by the user when initializing the algorithm. These haploid sets and the genotypes obtained thru crossovers are

```
Individual
   [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
c1    0    0    0    0    0    0    0    0    0     0
c2    1    1    1    1    1    1    1    1    1     1

Genotypes with 1 chromosome
      1    1    1    1    1    1    1    1    1     1

Classical two-point crossover with 1 chromosome
   [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
c3    0    0    0    0    0    1    1    0    0     0
c4    1    1    1    1    1    0    0    1    1     1
                          a)

Genotypes with 3 chromosomes
      1    1    1    1    2    2    2    3    3     3

Two-point crossovers with 3 chromosomes
   [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
c3    0    0    0    1    1    1    0    0    1     0
c4    1    1    1    0    0    0    1    1    0     1
                          b)

Crossovers and Random Assortment with 3 chromosomes
   [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
c3    0    0    0    1    0    0    1    1    0     1
c4    1    1    1    0    1    1    0    0    1     0
                          c)
```

Figure 2: Recombination operators a) classical two-point crossover with 1 chromosome, b) two-point crossover with 3 chromosomes, c) two-point crossover with 3 chromosomes and Random Assortment

then assembled and a new generation of randomly generated individuals is created. Six different operators for mutation are available in the package. They can be used solitarily or in any combination, to support exploration with the genetic algorithm. The classical point mutation is implemented. However, when selecting a small number of features from a large poll with a genetic algorithm, the point mutation has the tendency to progressively increase the number of activated genes in genotypes, with every generation. To address this drawback, we implemented several mutation mechanisms inspired from genetics. These operators take advantage of the genotypes being partitioned on different chromosomes. The alternatives to point mutation available in the **dGAselID** package are:

1. Nonsense Mutation,
2. Frameshift Mutation,
3. Large Segment Deletion,
4. Whole Chromosome Deletion,
5. Transposons.

The **Nonsense Mutation operator** annuls all the genes on a chromosome following a randomly selected locus (treated as a stop codon). The other chromosomes in the genotype are not influenced by the nonsense mutation. The **Frameshift Mutation operator** randomly selects a locus on an arbitrarily selected chromosome. The gene at the selected locus is deleted and all the following chain is shifted with one position to the left. The last locus on the implicated chromosome is subsequently annulled to conserve its length. Other chromosomes in the genotype are not altered by the mutation. The **Large Segment Deletion operator** annuls all the genes in a randomly generated interval on an arbitrarily selected chromosome. The **Whole Chromosome Deletion operator** acts in a similar fashion, but on the whole chromosome rather than an interval. The **Transposons operator** randomly selects a chromosome, a gene on that chromosome and a distance for relocation. The elected gene is then transferred at a locus indicated by the generated distance. All the mutation operators occur with chances specified by designated parameters. The mutated genotypes are verified to have at least 4 active genes and the invalid mutations are not inherited, to prevent errors during the subsequent fitness evaluations. The following code demonstrates the mutation operators. The R output is partially presented in Figure 3.

```
> library(ALL)
> data(ALL)
>
> demoALL<-ALL[1:12, 1:8]
>
> set.seed(1234)
```

```
> population<-InitialPopulation(demoALL, 4, 9)
> individuals<-Individuals(population)
> individuals
>
> set.seed(123)
> pointMutation(individuals, 4)
>
> chrConf<-splitChromosomes(demoALL, 2)
> chrConf
> individuals
>
> set.seed(123)
> nonSenseMutation(individuals, chrConf, 20)
>
> set.seed(123)
> frameShiftMutation(individuals, chrConf, 20)
>
> set.seed(123)
> largeSegmentDeletion(individuals, chrConf, 20)
>
> set.seed(123)
> wholeChromosomeDeletion(individuals, chrConf, 20)
>
> set.seed(123)
> transposon(individuals, chrConf, 20)
```

```
> individuals
  Id 1000_at 1001_at 1002_f_at 1003_s_at 1004_at 1005_at 1006_at 1007_s_at 1008_f_at 1009_at 100_g_at 1010_at
1  1       0       1          1          1       1       0       1          1          0       1         1        1
2  1       1       1          1          0       1       1       1          1          0       0         1        1
3  2       1       1          1          1       0       1       1          1          1       0         1        0
4  2       1       0          1          1       1       1       1          0          1       1         0        1
>
> set.seed(123)
> pointMutation(individuals, 4)
       Applying 1 Point Mutations...
  Id 1000_at 1001_at 1002_f_at 1003_s_at 1004_at 1005_at 1006_at 1007_s_at 1008_f_at 1009_at 100_g_at 1010_at
1  1       0       1          1          1       1       0       1          1          0       1         1        1
2  1       1       1          1          1       1       1       1          1          0       0         1        1
3  2       1       1          1          1       0       1       1          1          1       0         1        0
4  2       1       0          1          1       1       1       1          0          1       1         0        1
>
> chrConf<-splitChromosomes(demoALL, 2)
> chrConf
 [1] 1 1 1 1 1 1 2 2 2 2 2 2
>
> set.seed(123)
> nonSenseMutation(individuals, chrConf, 20)
       Applying 1 NonSenseMutation mutations...
  Id 1000_at 1001_at 1002_f_at 1003_s_at 1004_at 1005_at 1006_at 1007_s_at 1008_f_at 1009_at 100_g_at 1010_at
1  1       0       1          1          1       1       0       1          1          0       1         1        1
2  1       1       1          0          0       0       0       1          1          0       0         1        1
3  2       1       1          1          1       0       1       1          1          1       0         1        0
4  2       1       0          1          1       1       1       1          0          1       1         0        1
>
> set.seed(123)
> frameShiftMutation(individuals, chrConf, 20)
       Applying 1 FrameShiftMutation mutations...
  Id 1000_at 1001_at 1002_f_at 1003_s_at 1004_at 1005_at 1006_at 1007_s_at 1008_f_at 1009_at 100_g_at 1010_at
1  1       0       1          1          1       1       0       1          1          0       1         1        1
2  1       1       1          0          1       1       0       1          1          0       0         1        1
3  2       1       1          1          1       0       1       1          1          1       0         1        0
4  2       1       0          1          1       1       1       1          0          1       1         0        1
>
> set.seed(123)
> largeSegmentDeletion(individuals, chrConf, 20)
       Applying 1 LargeSegmentDeletion mutations...
  Id 1000_at 1001_at 1002_f_at 1003_s_at 1004_at 1005_at 1006_at 1007_s_at 1008_f_at 1009_at 100_g_at 1010_at
1  1       0       1          1          1       1       0       1          1          0       1         1        1
2  1       1       1          0          0       0       1       1          1          0       0         1        1
3  2       1       1          1          1       0       1       1          1          1       0         1        0
4  2       1       0          1          1       1       1       1          0          1       1         0        1
>
> set.seed(123)
> wholeChromosomeDeletion(individuals, chrConf, 20)
       Applying 1 WholeChromosomeDeletion mutations...
  Id 1000_at 1001_at 1002_f_at 1003_s_at 1004_at 1005_at 1006_at 1007_s_at 1008_f_at 1009_at 100_g_at 1010_at
1  1       0       1          1          1       1       0       1          1          0       1         1        1
2  1       0       0          0          0       0       0       1          1          0       0         1        1
3  2       1       1          1          1       0       1       1          1          1       0         1        0
4  2       1       0          1          1       1       1       1          0          1       1         0        1
>
> set.seed(123)
> transposon(individuals, chrConf, 20)
       Applying 1 Transposons mutations...
  Id 1000_at 1001_at 1002_f_at 1003_s_at 1004_at 1005_at 1006_at 1007_s_at 1008_f_at 1009_at 100_g_at 1010_at
1  1       0       1          1          1       1       0       1          1          0       1         1        1
2  1       1       1          1          1       0       1       1          1          0       0         1        1
3  2       1       1          1          1       0       1       1          1          1       0         1        0
4  2       1       0          1          1       1       1       1          0          1       1         0        1
```

Figure 3: Partial R output illustrating the mutation operators

Iteration with the new generation follows. The number of generations is established at the initialization of the algorithm.

Argument	Description
x	The dataset in "ExpressionSet class" format
response	The response variable
method	Supervised classifier for fitness evaluation
trainTest	Specifies the training set or the cross-validation method
startGenes	Number of alleles=1 in the starting genomes
populationSize	Initial populations size
iterations	Number of generations
noChr	The number of desired chromosomes
elitism	Elitism in percentages
ID	Dominance
pMutationChance	Chance for a point mutation to occur
nSMutationChance	Chance for a Nonsense Mutation to occur
fSMutationChance	Chance for a Frameshift Mutation to occur
lSDeletionChance	Chance for a Large Segment Deletion to occur
wChrDeletionChance	Chance for a Whole Chromosome Deletion to occur
transposonChance	Chance for a Transposon mutation to occur
randomAssortment	Random Assortment of chromosomes for recombinations
embryonicSelection	Remove chromosomes with fitness < specified value
EveryGeneInInitialPopulation	Request for every gene to be present in the initial population
nnetSize	For nnetI
nnetDecay	For nnetI
rdaAlpha	For rdaI
rdaDelta	For rdaI

Table 2: Parameters accepted by the dGAselID() function.

Working with the dGAselID package

The **dGAselID** package is structured around the dGAselID() function. This function manages the initial parameters for the algorithm and, depending on the user selected options, sets the stage for the experiment. The dGAselID() function calls other functions for the different steps and options in the algorithm. The arguments accepted by dGAselID() along with short descriptions are presented in Table 2. The other functions, for different operators used during the genetic algorithm search, are summarized in Table 3.

Graphical representations of the evolution are available with the built-in functions in real-time. The maximum and average accuracy, accompanied by the most frequently selected genes can be displayed after each generation, offering a very intuitive image of the evolution. Evidence about the number of individuals in the current population, crossovers or the number of mutations are displayed for each generation.

The algorithm retains various data about the evolution for further analysis. For each gene, the frequency of selection across generations is recorded, along with other characteristics of the evolution. The output data format with a hypothetical result is shown below, together with a description of the recorded variables in the Table 4.

```
> ## Not run:
> names(result)        #hypothetical result
[1] "DGenes"          "dGenes"          "MaximumAccuracy" "MeanAccuracy"
[5] "MinAccuracy"     "BestIndividuals"
> ## End(Not run)
```

Example

We illustrate the functionality of the **dGAselID** package with the **ALL** dataset (Li, 2009), a very known set of real DNA microarray data, available in Bioconductor. We are searching for the differentially expressed genes that could characterize the patients suffering from acute lymphoblastic leukemia but have different BCR/ABL classification, negative or positive. For this example, we use a subset of the original ALL data, non-specifically and specifically filtered to 628 features and 79 samples, from the

Function	Description
`dGAselID()`	Main function
`AnalyzeResults()`	Ranks individuals according to their fitness and records the results
`Crossover()`	Operator for the two-point crossover
`Elitism()`	Performs elitism for the desired threshold
`EmbryonicSelection()`	Deletes individuals with a fitness below a specified threshold
`EvaluationFunction()`	Evaluates the individuals' finesses after each iteration
`frameShiftMutation()`	Operator for the Frameshift Mutation
`Individuals()`	Generates individuals from haploid chromosome sets
`InitialPopulation()`	Generates the initial random haploid chromosome sets
`largeSegmentDeletion()`	Operator for the Large Segment Deletion mutation
`nonSenseMutation()`	Operator for the Nonsense Mutation
`PlotGenAlg()`	Plots the evolution after each generation
`pointMutation()`	Operator for the point mutation
`RandomAssortment()`	Performs the Random Assortment of chromosomes
`RandomizePop()`	Creates the random population for the next generation
`splitChromosomes()`	Divides the genotypes in a set with the desired number of chromosomes
`transposon()`	Operator for the Transposon mutation
`wholeChromosomeDeletion()`	Operator for the Whole Chromosome Deletion mutation

Table 3: Functions in the dGAselID package.

Variable	Description
`DGenes`	The occurrences in selected genotypes for every gene
`dGenes`	The occurrences in discarded genotypes for every gene
`MaximumAccuracy`	Maximum accuracy in every generation
`MeanAccuracy`	Average accuracy in every generation
`MinAccuracy`	Minimum accuracy in every generation
`BestIndividuals`	Best individual in every generation

Table 4: `dGAselID()` output.

12625 features and 128 patients in the complete data set. The data was filtered using the capabilities offered in the **genefilter** package (Gentleman et al., 2016). The algorithm works as well on the original data set, but the filtered data is searched faster. However, our experiments with real data show that more reliable results are obtained with full featured data.

The code for constructing the dataset used in the following examples is presented below:

```
> library(genefilter)
> library(ALL)
> data(ALL)
> bALL = ALL[, substr(ALL$BT,1,1) == "B"]
> smallALL = bALL[, bALL$mol.biol %in% c("BCR/ABL", "NEG")]
> smallALL$mol.biol = factor(smallALL$mol.biol)
> smallALL$BT = factor(smallALL$BT)
> f1 <- pOverA(0.25, log2(100))
> f2 <- function(x) (IQR(x) > 0.5)
> f3 <- ttest(smallALL$mol.biol, p = 0.1)
> ff <- filterfun(f1, f2, f3)
> selectedsmallALL <- genefilter(exprs(smallALL), ff)
> smallALL = smallALL[selectedsmallALL, ]
```

An example of function call is:

```
> set.seed(149)
> resNoID<-dGAselID(smallALL, "mol.biol", trainTest = 1:79, startGenes = 12,
+ populationSize = 200,  iterations = 300, noChr = 5, pMutationChance = 0.0075,
+ elitism = 4)
```

The choice for evaluation function was `knn.cvI` from the **MLInterfaces** package, with the parameters `k=3` and `l=2`. This is the default `method` in the package. The evaluation function used in this example, `knn.cvI`, is a kNN classifier with the leave-one-out cross validation embedded. For this reason, the requirement for the `trainTest` parameter is special, addresses all the instances in the data, `1:79`. For any other supervised classifier, the `trainTest` parameter shapes the training and testing subsets in the same fashion as the `trainInd` parameter in **MLInterfaces**. When cross-validation is required, the `trainTest` parameter specifies the desired method as `"LOO"` or `"LOG"`, and is equivalent to `xvalSpec("LOO")` or `xvalSpec("LOG")` respectively, in **MLInterfaces** package. An illustration of the evolution is presented in Figure 4. The figure pictures a juncture during the search, including the information provided by the verbose mode and graphical representations of the evolution, as they appear in real-time on the computer screen. The evolutions of the Maximum Accuracy, Average Accuracy and the most frequently selected genes are presented after each generation.

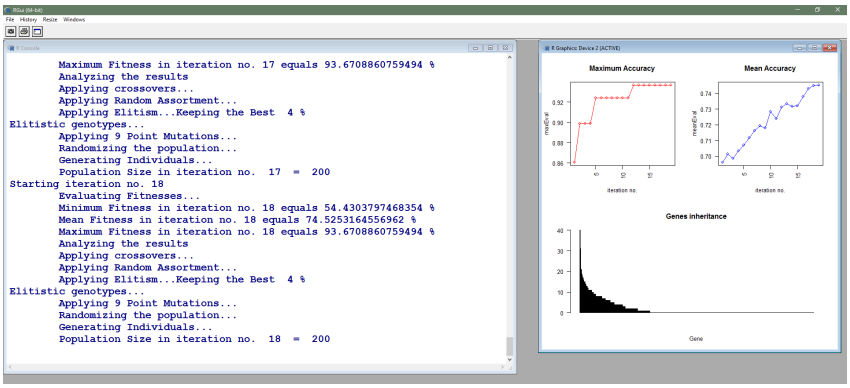


Figure 4: Screenshot of the algorithm execution after 18 generations.

After the desired number of generations, the evolution of the Maximum Accuracy and Average Accuracy in every generation and the most frequently selected genes can be displayed with the included functions. The Figure 5 represents the most frequently selected genes after the specified number of generations, 300 in our example. Their characteristics can be acquired with methods already implemented in Bioconductor. The 10 most selected genes are presented and could be obtained using **hgu95av2.db** (Carlson, 2016) with the code below. The selected genes can be studied and evaluated with all the methods available in Bioconductor. For reliable and interpretable results, an adequate number of replications are mandatory in such an experiment, due to the stochastic makeup of the genetic algorithms. The evolution of the maximum accuracy and the average accuracy can be plotted as illustrated in the Figure 6 and Figure 7, respectively.

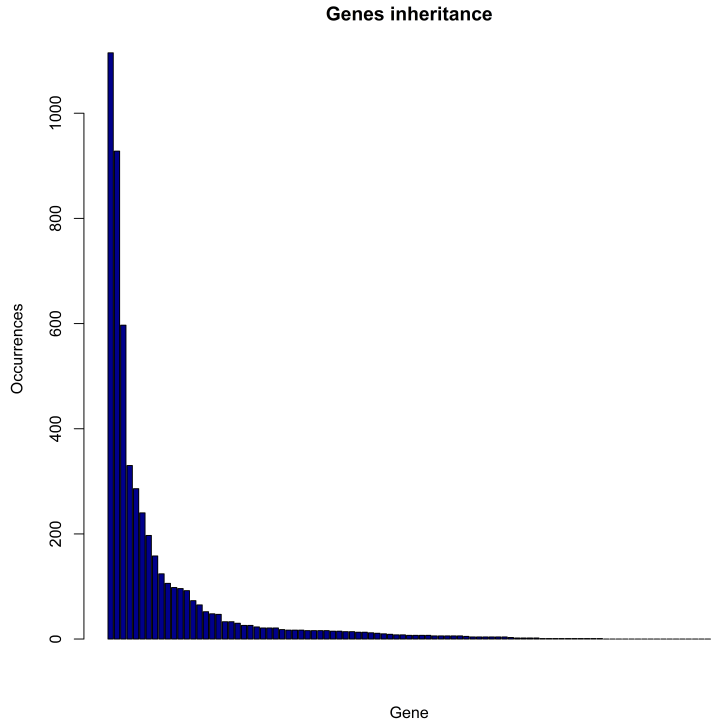


Figure 5: The most frequently selected genes after 300 generations.

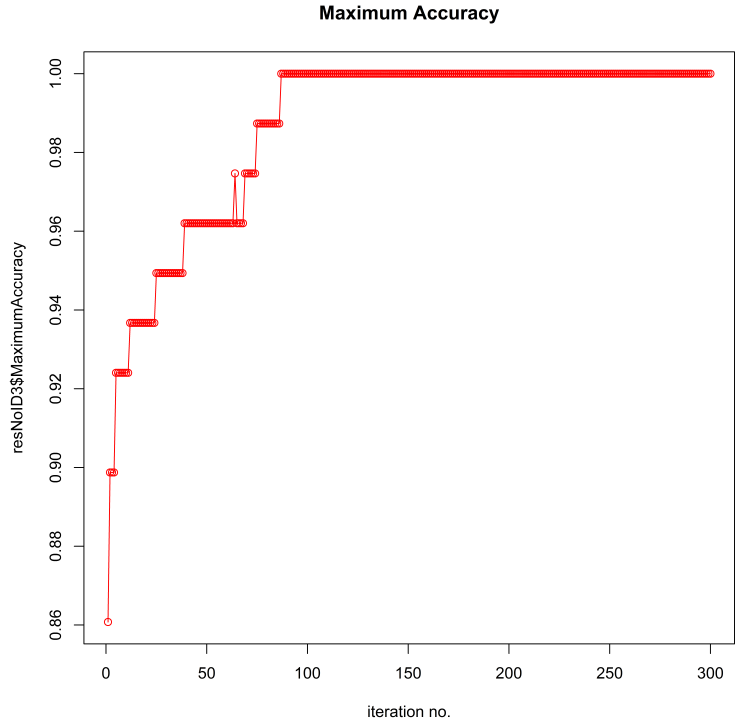


Figure 6: Evolution of the maximum accuracy after 300 generations.

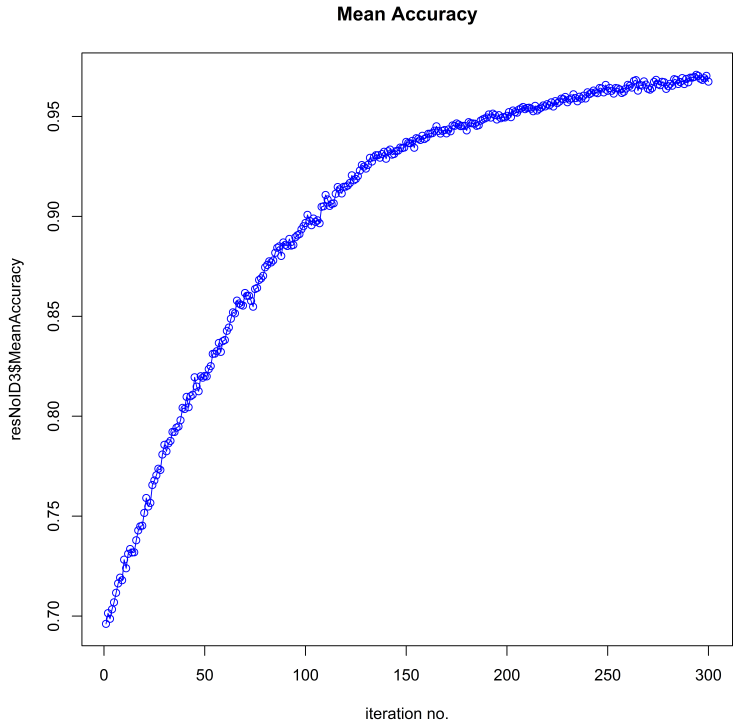


Figure 7: Evolution of the average accuracy after 300 generations.

```
> library(hgu95av2.db)
> DGenes<-resNoID$DGenes
> selectedD<-colnames(DGenes)[order(DGenes, decreasing = TRUE)]
> mget(selectedD, hgu95av2GENENAME, ifnotfound=NA)[1:10]
$`1635_at`
[1] "ABL proto-oncogene 1, non-receptor tyrosine kinase"

$`39730_at`
[1] "ABL proto-oncogene 1, non-receptor tyrosine kinase"

$`39070_at`
[1] "fascin actin-bundling protein 1"

$`34362_at`
[1] "solute carrier family 2 member 5"

$`39338_at`
[1] "S100 calcium binding protein A10"

$`40091_at`
[1] "B-cell CLL/lymphoma 6"

$`1135_at`
[1] "G protein-coupled receptor kinase 5"

$`38385_at`
[1] "destrin, actin depolymerizing factor"

$`40396_at`
[1] "purinergic receptor P2X 5"

$`38069_at`
[1] "chloride voltage-gated channel 7"
```

Other graphical representation tools offered in R are readily available for further investigation. An image of the evolution for the best individual in every generation can be depicted as in Figure 8. The tendency to increment the number of active genes in the genotypes with the successive generations, induced by the point mutation, becomes apparent. In contrast, the transposon mutation does not convey this inconvenience, as illustrated in Figure 9. The graphical representations are accessible with the code:

```
> bestsNoID<-resNoID$BestIndividuals
> dev.off()
> image(1:ncol(bestsNoID), 1:nrow(bestsNoID), t(bestsNoID),
+ xlim = c(0, ncol(bestsNoID)), ylim = c(0, nrow(bestsNoID)), col = c("white", "red"),
+ cex.axis = 0.7, cex.lab = 0.8, cex.main = 1.2, lty = 1, lwd = 2, las= 2, xaxs = "r",
+ yaxs = "r", pty = "m", ylab = "Generation no.", xlab = "Gene no.",
+ main = "Best Individuals with the Point Mutation")

> set.seed(149)
> restransp<-dGAselID(smallALL, "mol.biol", trainTest = 1:79, startGenes = 12,
+ populationSize = 200, iterations = 300, noChr = 5, pMutationChance = 0,
+ transposonChance = 2, elitism= 4)

> beststransp<-restransp$BestIndividuals
> dev.off()
> image(1:ncol(beststransp), 1:nrow(beststransp), t(beststransp),
+ xlim = c(0, ncol(beststransp)), ylim = c(0, nrow(beststransp)),
+ col = c("white", "red"), cex.axis = 0.7, cex.lab = 0.8, cex.main = 1.2, lty = 1,
+ lwd = 2, las = 2, xaxs = "r", yaxs = "r", pty = "m", ylab = "Generation no.",
+ xlab = "Gene no.", main = "Best Individuals with the Transposon Mutation")
```

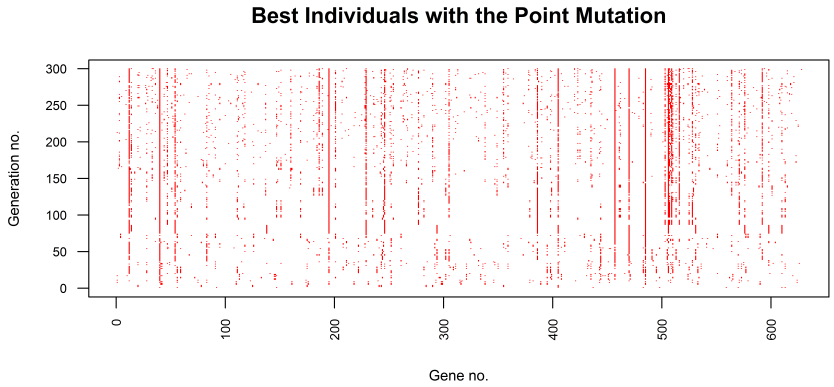


Figure 8: Evolution of the best individual after 300 generations with Point Mutation.

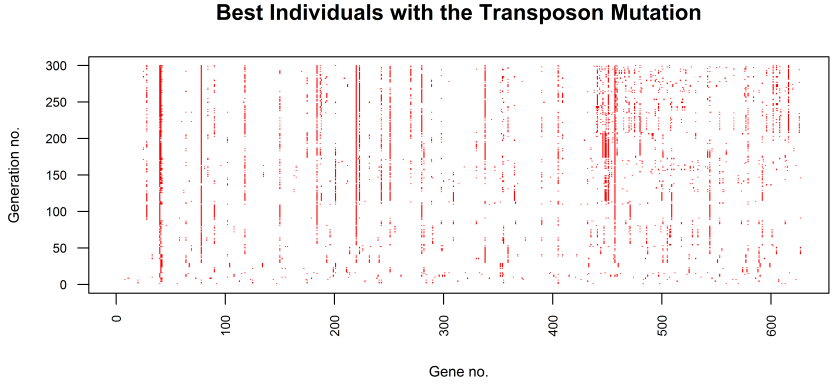


Figure 9: Evolution of the best individual after 300 generations with Transposon Mutation.

An illustrative comparison between the classical and the Incomplete Dominance implementations is accessible with the subsequent code. The two approaches can be assessed in terms of evolutions of the maximum and average fitness, as depicted in Figure 10 and Figure 11, respectively. The **scales** package (Wickham, 2016) is very useful for the direct visual comparison. The most frequently selected features are compared in Figure 12. It is noticeable that the Incomplete Dominance approach favors exploration, while still evolving very solidly. This behavior is desirable when selecting features with a genetic algorithm. The same tendency is perceptible when examining Figure 12, where the number of significant features is higher with the Incomplete Dominance method. Multiple replications of an experiment are mandatory to draw reliable conclusions. This example is presented for illustration purpose only.

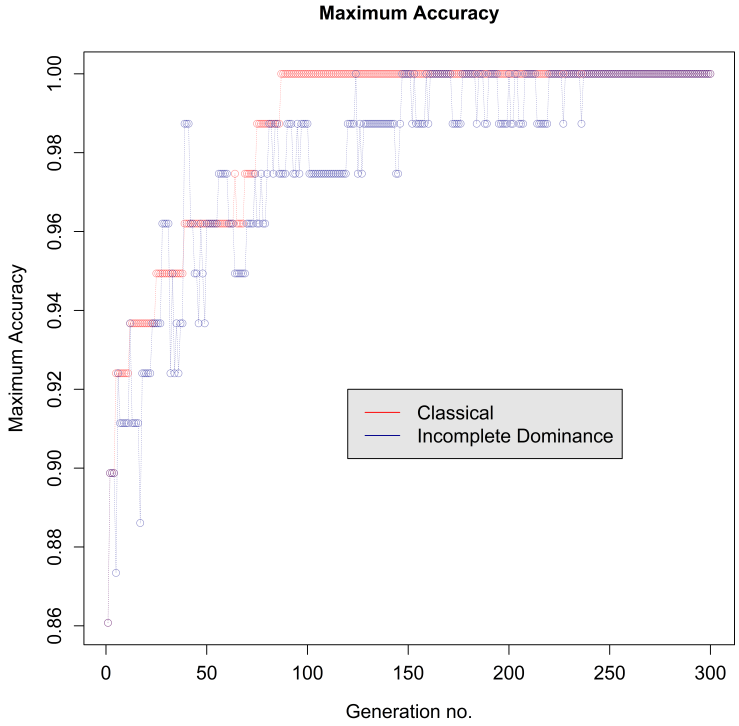


Figure 10: Comparative evolution of the maximum fitness over 300 generations.

```
> library(scales)
> set.seed(149)
> resID<-dGAselID(smallALL, "mol.biol", trainTest = 1:79, startGenes = 12,
+ populationSize = 200, iterations = 300, noChr = 5, pMutationChance = 0.0075,
+ elitism = 4, ID = "ID2")

> dev.off()
> par("xlog"=FALSE)
> plot(resNoID$MaximumAccuracy, type = "o", col = alpha("red", 0.5), pch = 1,
+ cex.axis = 1.2, cex.lab = 1.2, cex.main = 1.2, lty = 3, lwd = 0.5,
+ xlab = "Generation no.", ylab = "Maximum Accuracy", main = "Maximum Accuracy")
> points(resID$MaximumAccuracy, type = "o", col = alpha("darkblue", 0.5), pch = 1,
+ lty = 3, lwd = 0.5)
> legend(120, 0.92, c("Classical", "Incomplete Dominance"), cex = 1.2,
+ col = c("red", "darkblue"), merge = FALSE, bg = "gray90", lty = c(1, 1, 1))

> plot(resNoID$MeanAccuracy, type = "o", col = alpha("red", 0.5), pch = 1, lwd = 0.5,
+ cex.axis = 1.2, cex.lab = 1.2, cex.main = 1.2, xlab = "Generation no.",
+ ylab = "Mean Accuracy", main = "Mean Accuracy")
> points(resID$MeanAccuracy, type = "o", col = alpha("darkblue", 0.5), pch = 1,
+ lty = 3, lwd = 0.5)
> legend(120, 0.8, c("Classical", "Incomplete Dominance"), cex = 1.2,
+ col = c("red", "darkblue"), merge = FALSE, bg = "gray90", lty = c(1, 1, 1))
```

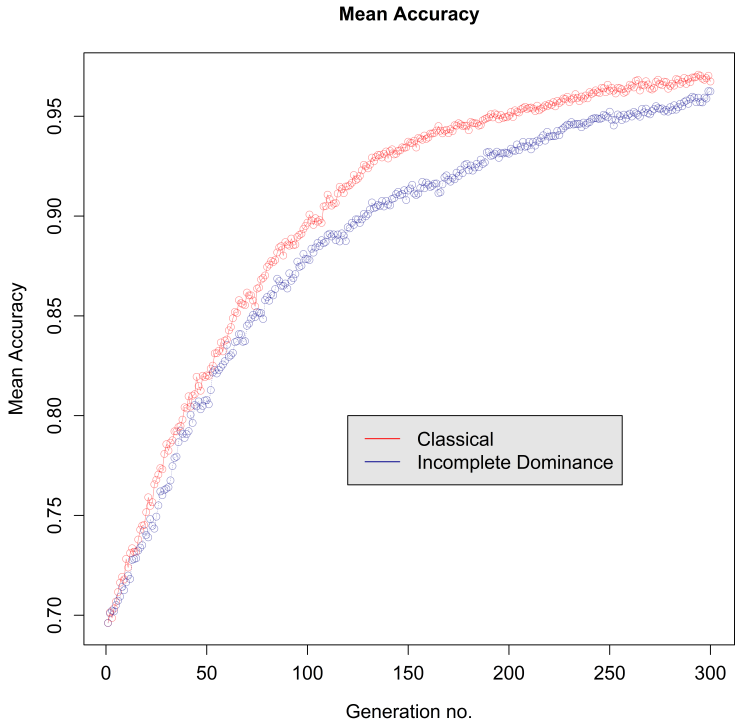


Figure 11: Comparative evolution of the average fitness over 300 generations.

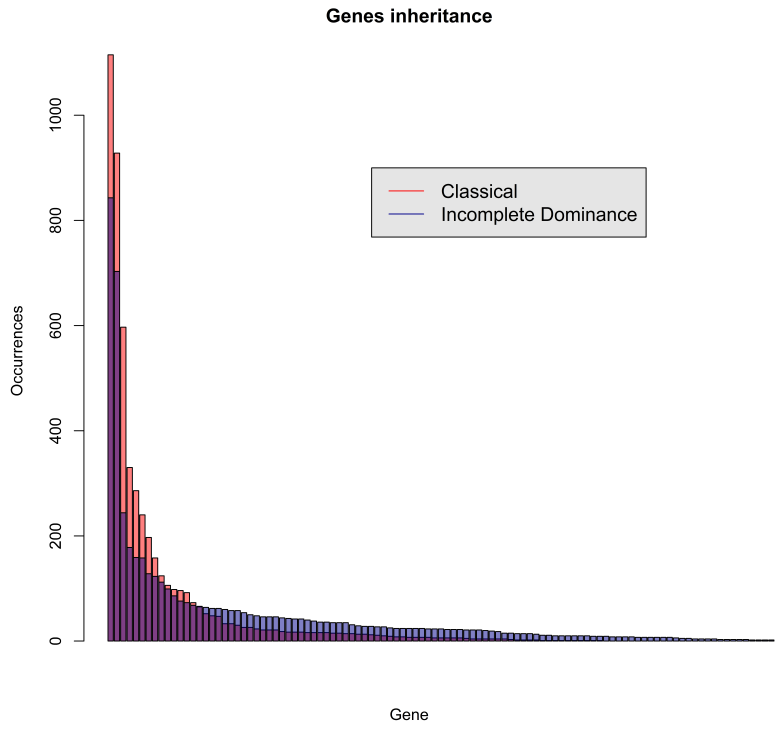


Figure 12: Comparatison of the most selected genes after 300 generations.

```
> DGenes1<-resNoID$DGenes[sort(resNoID$DGenes, decreasing = TRUE,
+ index.return = TRUE)$ix]
> DGenes2<-resID$DGenes[sort(resID$DGenes, decreasing = TRUE, index.return = TRUE
+ )$ix]
> significant1<-DGenes1[sort(resNoID$DGenes, decreasing = TRUE, index.return = TRUE
+ )$ix] > floor(5*max(DGenes1)/100)
> significant2<-DGenes2[sort(resID$DGenes, decreasing = TRUE, index.return = TRUE
+ )$ix] > floor(5*max(DGenes2)/100)
> barplot(DGenes1[significant1], main = "Genes inheritance", xlab = "Gene",
+ ylab = "Occurrences", col = alpha("red", 0.5), beside = FALSE, add = FALSE)
> barplot(DGenes2[significant2], main = "Genes inheritance", xlab = "Gene",
+ ylab = "Occurrences", col = alpha("darkblue", 0.5), beside = FALSE, add = TRUE)
> legend(50, 900, c("Classical", "Incomplete Dominance"), cex = 1.2, col = c("red",
+ "darkblue"), merge = FALSE, bg = "gray90", lty = c(1, 1, 1))
```

Conclusions

The **dGAselID** package provides a creative approach to feature selection in high dimensional data. The package utilizes the data format used in Bioconductor and is readily operational for microarray data analysis. The algorithm is flexible for high dimensional data other than microarray, when data is provided according to the `"ExpressionSet class"` specifications. In our experience, the diploid implementation offers advantages over the haploid GA, especially when cross-validation techniques are engaged. The Incomplete Dominance approach allows for a diploid framework, bypassing the requirement to specify a dominance scheme. Also, the Incomplete Dominance inheritance models an evolution process present in nature, tested by billions of years of evolution. Moreover, the diploid structure is a foundation for crossover and mutation operators that model the natural processes and provide improvements in performance over the classical versions. In our tests, the Random Assortment of Chromosomes provides an important performance advantage, in many situations.

Future development

The main disadvantages of the algorithm presented in **dGAselID** package are the necessity to replicate an experiment several times for reliable results, given the fortuity in generating the initial population, and the tendency of the GA to converge in local optima. Following the conduit in evolutionary computation, we will reconsider the principles of evolution and accurately model operators for crossover and mutation to offer a better tension between exploration and exploitation.

Bibliography

D. Akdemir, J. Sanchez, and J.-L. Jannink. Optimization of genomic selection training populations with a genetic algorithm. *Genetics Selection Evolution*, 47(1):38, 2015. URL https://doi.org/10.1186/s12711-015-0116-6. [p18]

J. Berard and A. Bienvenue. Sharp asymptotic results for simplified mutation-selection algorithms. *The Annals of Applied Probability*, 13(4):1534–1568, 2003. URL https://doi.org/10.1214/aoap/1069786510. [p18]

V. Carey, R. Gentleman, J. Mar, J. Vertrees, and L. Gatto. *MLInterfaces: Uniform interfaces to R machine learning procedures for data in Bioconductor containers*, 2016. URL http://bioconductor.org/packages/MLInterfaces/. R package version 1.52.0. [p19]

M. Carlson. *hgu95av2.db: Affymetrix Human Genome U95 Set annotation data (chip hgu95av2)*, 2016. URL http://bioconductor.org/packages/hgu95av2.db/. R package version 3.2.3. [p26]

B. Doerr and C. Doerr. A tight runtime analysis of the (1 + (λ,λ)) genetic algorithm on onemax. *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 1423–1430, 2015. URL http://doi.acm.org/10.1145/2739480.2754683. [p18]

S. Droste, T. Jansen, and I. Wegener. On the analysis of the (1+ 1) evolutionary algorithm. *Theoretical Computer Science*, 276(1-2):51–81, Apr. 2002. ISSN 0304-3975. URL https://doi.org/10.1016/S0304-3975(01)00182-7. [p18]

R. Gentleman, V. Carey, W. Huber, and F. Hahne. *genefilter: genefilter: methods for filtering genes from high-throughput experiments*, 2016. URL https://bioconductor.org/packages/genefilter/. R package version 1.56.0. [p26]

D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1989. ISBN 0201157675. [p18]

J. Holland. *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence*. University of Michigan Press, 1975. ISBN 0472084607. Extended new Edition, MIT Press, Cambridge (1992). [p18]

W. Huber, V. Carey, R. Gentleman, ..., and M. Morgan. Orchestrating high-throughput genomic analysis with bioconductor. *Nature Methods*, 12(2):115–121, 2015. ISSN 1548-7091. URL https://doi.org/10.1038/nmeth.3252. [p18]

D. Kepplinger. *gaselect: Genetic Algorithm (GA) for Variable Selection from High-Dimensional Data*, 2015. URL https://cloud.r-project.org/web/packages/gaselect/index.html. R package version 1.0.5. [p19]

X. Li. *ALL: A data package*, 2009. URL http://bioconductor.org/packages/release/data/experiment/html/ALL.html. R package version 1.14.0. [p24]

N. Melita and S. Holban. The random assortment for selecting features with genetic algorithms in microarray data analysis. *in publishing*, 2016a. [p21]

N. Melita and S. Holban. An incomplete dominance genetic algorithm approach to microarray data analysis. *2016 IEEE 12th International Conference on Intelligent Computer Communication and Processing (ICCP)*, 2016b. URL https://doi.org/10.1109/ICCP.2016.7737137. [p19]

N. Melita, I. Popescu, and S. Holban. A genetic algorithm approach to dna microarrays analysis of pancreatic cancer. *Advances in Electrical and Computer Engineering*, 8(2):43–48, 2008. ISSN 1582-7445. URL https://doi.org/10.4316/AECE.2008.02008. [p19]

M. Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA, USA, 1998. ISBN 0262631857. [p18]

T. Pajala. *mogavs: Multiobjective Genetic Algorithm for Variable Selection in Regression*, 2016. URL https://cloud.r-project.org/web/packages/mogavs/index.html. R package version 1.0.1. [p19]

G. Rudolph. Convergence analysis of canonical genetic algorithms. *IEEE Trans. Neural Net.*, 5(1): 96–101, 1994. URL https://doi.org/10.1109/72.265964. [p18]

L. Scrucca. On some extensions to ga package: hybrid optimisation, parallelisation and islands evolution. *Submitted to R Journal*, 2016. URL http://arxiv.org/abs/1605.01931. [p18]

F. Tenorio. *gaoptim: Genetic Algorithm optimization for real-based and permutation-based problems*, 2013. URL https://cloud.r-project.org/web/packages/gaoptim/index.htmll. R package version 1.1. [p18]

C.-S. Tsou. *nsga2R: Elitist Non-dominated Sorting Genetic Algorithm based on R*, 2015. URL https://cloud.r-project.org/web/packages/nsga2R/index.html. R package version 1.0. [p18]

H. Wickham. *scales: Scale Functions for Visualization*, 2016. URL https://CRAN.R-project.org/package=scales. R package version 0.4.1. [p30]

E. Willighagen and M. Ballings. *genalg: R Based Genetic Algorithm*, 2015. URL https://cloud.r-project.org/web/packages/genalg/index.html. R package version 0.2.0. [p18]

M. Wolters. A genetic algorithm for selection of fixed-size subsets with application to design problems. *Journal of Statistical Software*, 68(1):1–18, 2015. URL https://doi.org/10.18637/jss.v068.c01. [p18]

B. Xue, M. Zhang, W. Browne, and X. Yao. Survey on evolutionary computation approaches to feature selection. *IEEE Transactions on Evolutionary Computation*, 20(4), 2016. URL https://doi.org/10.1109/TEVC.2015.2504420. [p18]

Nicolae Teodor MELITA
Department of Computer and Software Engineering
Politehnica University of Timisoara
Timisoara, RO-300223
Romania
nt_melita@yahoo.com

Stefan HOLBAN
Department of Computer and Software Engineering
Politehnica University of Timisoara
Timisoara, RO-300223
Romania
stefan.holban@cs.upt.ro

Allele Imputation and Haplotype Determination from Databases Composed of Nuclear Families

by Nathan Medina-Rodríguez and Ángelo Santana

Abstract The **alleHap** package is designed for imputing genetic missing data and reconstruct non-recombinant haplotypes from pedigree databases in a deterministic way. When genotypes of related individuals are available in a number of linked genetic markers, the program starts by identifying haplotypes compatible with the observed genotypes in those markers without missing values. If haplotypes are identified in parents or offspring, missing alleles can be imputed in subjects containing missing values. Several scenarios are analyzed: family completely genotyped, children partially genotyped and parents completely genotyped, children fully genotyped and parents containing entirely or partially missing genotypes, and founders and their offspring both only partially genotyped. The **alleHap** package also has a function to simulate pedigrees including all these scenarios. This article describes in detail how our package works for the desired applications, including illustrated explanations and easily reproducible examples.

Introduction

The knowledge about human genetic variation has been growing exponentially over the last decade. Collaborative efforts of international projects such as HapMap (Consortium and others, 2005) and 1000 Genomes (Consortium and others, 2012) have contributed to improving the discovery about human genetic diversity.

Genotype imputation and haplotype reconstruction have achieved an important role in *Genome-Wide Association Studies* (GWAS) during recent years. Estimation methods are frequently used to infer missing genotypes as well as haplotypes from databases containing related and/or unrelated subjects. The majority of these analyses have been developed using several statistical methods (Browning and Browning, 2011a) which are able to impute genotypes as well as perform haplotype phasing (also known as haplotype estimation) of the corresponding genomic regions.

Most of the currently available computer programs such as IMPUTE2 (Howie and Marchini, 2010), MINIMAC3 (Das, 2015), BEAGLE (Browning and Browning, 2011b), and others, or R packages such as: **haplo.ccs** (French and Lumley, 2012), **haplo.stats** (JP and DJ, 2016), **hsphase** (Ferdosi et al., 2014), **linkim** (Xu and Wu, 2014), **rrBLUP** (Endelman, 2011), and **synbreed** (Wimmer et al., 2012) carry out genotype imputation or haplotype reconstruction using probabilistic methods to achieve their objectives when deterministic methods are insufficient to get them without errors. These methods are usually focused on population data and in the case of pedigree data, families normally are comprised by duos (parent-child) or trios (parents-child) (Browning and Browning, 2009). Studies focused on more than two offspring for each line of descent are uncommon. In these cases, the above programs do not take full advantage of the information contained in the global family structure to improve the process of imputation and construction of haplotypes. The program HAPLORE (Zhang et al., 2005), developed in C++, takes a similar approach as **alleHap** for haplotype reconstruction in pedigrees, but can not be easily integrated into an environment where R packages are extensively used.

On the other hand, certain genomic regions are very stable against recombination but at the same time they may have a considerable amount of mutations. For this reason, in some well-studied regions, such as the *Human Leukocyte Antigen* (HLA) loci (Mack et al., 2013), located in the extended *Major Histocompatibility Complex* (MHC) (de Bakker et al., 2006), an alphanumeric nomenclature is needed to facilitate later analysis. At this juncture, the available typing techniques usually are not able to determine the allele phase and therefore the constitution of the appropriate haplotypes is not possible.

This paper presents new improvements and a detailed description for the R package **alleHap** (Medina-Rodríguez et al., 2014). Our program is capable of imputing missing alleles and identifying haplotypes from non-recombinant regions considering the mechanism of heredity and the genetic information present in parents and offspring. The algorithm is deterministic in the sense that haplotypes are identified from the existing genotypes guaranteeing compatibility between parents and children. When a haplotype can not be identified (due to genotyping errors, or recombination events in the genetic region), the procedure does not infer more haplotypes in the corresponding family members. The following sections will describe the implemented methods as well as some functional examples.

Basics

The algorithms in **alleHap** are based on a preliminary analysis of all possible combinations that may exist in the genotype of a marker, considering that each member of the family should unequivocally have inherited two alleles, one from each parent. The analysis is based on the differentiation of seven cases, as described in (Berger-Wolf et al., 2007). Each case is characterized by the number of different alleles present in the family and the way these alleles are distributed among parents that determine the set of possible genotypes in children.

Table 1 shows these cases when there are no missing genotypes. For example, in case 1 both parents are $a|a$ and so the only possible child is $a|a$; in case 2 if a parent is $a|a$ and the other is $b|b$ the only possible child is $a|b$. Note that in both cases it is possible for a child to determine from which parent comes each allele. The rest of the cases can be easily understood in the same way. Note that in case 5 if a child is $a|b$ it is not possible to know from whom comes each allele. The notation '|' indicates that the source of each allele can be assigned without error, and the notation '/' implies that origin is unknown.

To determine the haplotypes, **alleHap** creates an IDentified/Sorted (IDS) matrix from the genotypes of each family. For example, in a child, the genotype a/b of a marker is *phased* if it can be unequivocally determined that the first allele comes from the father and the second from the mother. In this way, the sequence of first (second) alleles of phased markers is the haplotype inherited from the father (or mother). So, when a marker in a child can be phased this way its IDS value is 1; in other case its value is 0. In parents, genotypes can be phased if there exists at least one child with all its genotypes phased *reference child*[1]. Then, for every marker, the alleles of a parent genotype are sorted in such a way that first allele coincides with the corresponding allele inherited from that parent in the reference child. When this sorting is achieved, the IDS value in the parent is 1; in other case its value is 0.

An example of the IDS matrix values (right) and the corresponding phased genotypes (left) is shown in Table 1. Note that when the genotype a/b is phased we denote it by $a|b$; the first child in each group is considered to be the reference child for phasing the parents. In this case the IDS values of parents have been deduced considering as *reference child*[1] the first one of the family.

	Phased Data							**IDS Matrix**						
Marker	1	2	3	4	5	6	7	1	2	3	4	5	6	7
Parents	a|a	a|a	a|a	a|a	a|b	a|b	a|b	1	1	1	1	1	1	1
	a|a	b|b	a|b	b|c	a|b	a|c	c|d	1	1	1	1	1	1	1
Offspring	a|a	a|b	a|a	a|b	a|a	a|a	a|c	1	1	1	1	1	1	1
			a|b	a|c	b|b	a|b	a|d			1	1	1	1	1
					a/b	a|c	b|c					0	1	1
						b|c	b|d						1	1

Table 1: Phased genotypes and IDS matrix.

Sometimes, missing values may occur. These can be located either in parents or children. An example of this is depicted in Table 2, where missing values have been denoted as NA (Not Available).

	Phased Data							**IDS Matrix**						
Marker	1	2	3	4	5	6	7	1	2	3	4	5	6	7
Parents	a|a	a|a	a|a	a|a	NA	NA	NA	1	1	1	1	0	0	0
	NA	NA	NA	b|c	a|b	a|b	a|b	0	0	0	1	1	1	1
Offspring	a|a	a|b	a|a	NA	a|a	a|a	a|c	1	1	1	0	1	1	1
			a|b	a|c	b|b	a/b	a|d			1	1	1	0	1
					a/b	a|c	b|c					0	1	1
						b|c	b|d						1	1

Table 2: Phased genotypes and IDS matrix containing missing data.

An identification of the homozygous genotypes for each family is also necessary for the proper operation of **alleHap**. This identification is done in the Homozygosity matrix (HMZ). This matrix has as many rows as members in the family and as many columns as markers. The term $HMZ_{i,j}$ is 0 if the subject i is heterozygous in the marker j, and 1 if he/she is heterozygous. An example of some unphased genotypes (left) and their corresponding HMZ values is shown in Table 3.

[1] *Reference child* has the highest number of phased marker's alleles (maximum number of IDS values equal to 1).

	Unphased Data							Homozygosity Info.						
Marker	1	2	3	4	5	6	7	1	2	3	4	5	6	7
Parents	a/a	a/a	a/a	a/a	a/b	a/b	a/b	1	1	1	1	0	0	0
	a/a	b/b	a/b	b/c	a/b	a/c	c/d	1	1	0	0	0	0	0
Offsring	a/a	a/b	a/a	a/b	a/a	a/a	a/c	1	0	1	1	1	0	0
			a/b	a/c	b/b	a/b	a/d			0	1	1	0	0
					a/b	a/c	b/c					0	0	0
						b/c	b/d						0	0

Table 3: Biallelic unphased genotypes and HMZ matrix.

Formats

The **alleHap** package only works with PED files, although it can be easily adapted to similar formats (with similar structure) to later be loaded into the program.

PED files

A PED file (Purcell et al., 2007) is a white-space (space or tab) delimited file where the first six columns are mandatory and the rest of columns are the genotype: '`Family ID`' (identifier of each family), '`Individual ID`' (identifier of each member of the family), '`Paternal ID`' (identifier of the paternal ancestor), '`Maternal ID`' (identifier of the maternal ancestor), '`Sex`' (genre of each individual: 1=*male*, 2=*female*, other=*unknown*), '`Phenotype`' (quantitative trait or affection status of each individual: -9=*missing*, 1=*unaffected*, 2=*affected*), and the '`genotype`' of each individual (in biallelic or coded format).

The identifiers are alphanumeric: the combination of family and individual ID should uniquely identify a person. **PED files must have one and only one phenotype in the sixth column**. The phenotype can be either a quantitative trait or an affection status column. Genotypes (seventh column onwards) should also be white-space delimited; they can be any character (e.g. 1, 2, 3, 4 or A, C, G, T or anything else) except 0, -9, -99. All markers should be biallelic and must have two alleles specified (Purcell et al., 2007). Note that **alleHap does not use the phenotypic information** that is located in these columns.

NA values

The missing or `NA` values may be placed either in the first six columns or also in genotype columns. In the last case, when some values are missing, both alleles should be 0, -9, -99 or `NA`. For example, a family composed of five individuals typed along three markers can be represented in the following way:

```
famID  indID patID matID sex phenot Mk1_1 Mk1_2 Mk2_1 Mk2_2 Mk3_1 Mk3_2
FAM001     1     0     0   1      1     1     2    NA    NA     1     2
FAM001     2     0     0   2      2     3     4     1     2     3     4
FAM001     3     1     2   1      2     1     3     1     2     1     3
FAM001     4     1     2   2      1    NA    NA     1     1     2     4
FAM001     5     1     2   1      2     1     4     1     1     2     4
```

Workflow

The workflow of **alleHap** comprises three stages: data loading, data imputation, and data haplotyping. Optionally if simulated data are to be used, a "pre-stage" data simulation must be done. The next subsections will describe each of them.

Data simulation

This "pre-stage" is implemented by an R function called `alleSimulator` that simulates genotypic data for parent-offspring pedigrees taking into account many different factors such as: number of families to generate, number of markers (allele pairs), maximum number of different alleles per marker in the population, type of alleles (numeric or character), number of unique haplotypes in the population,

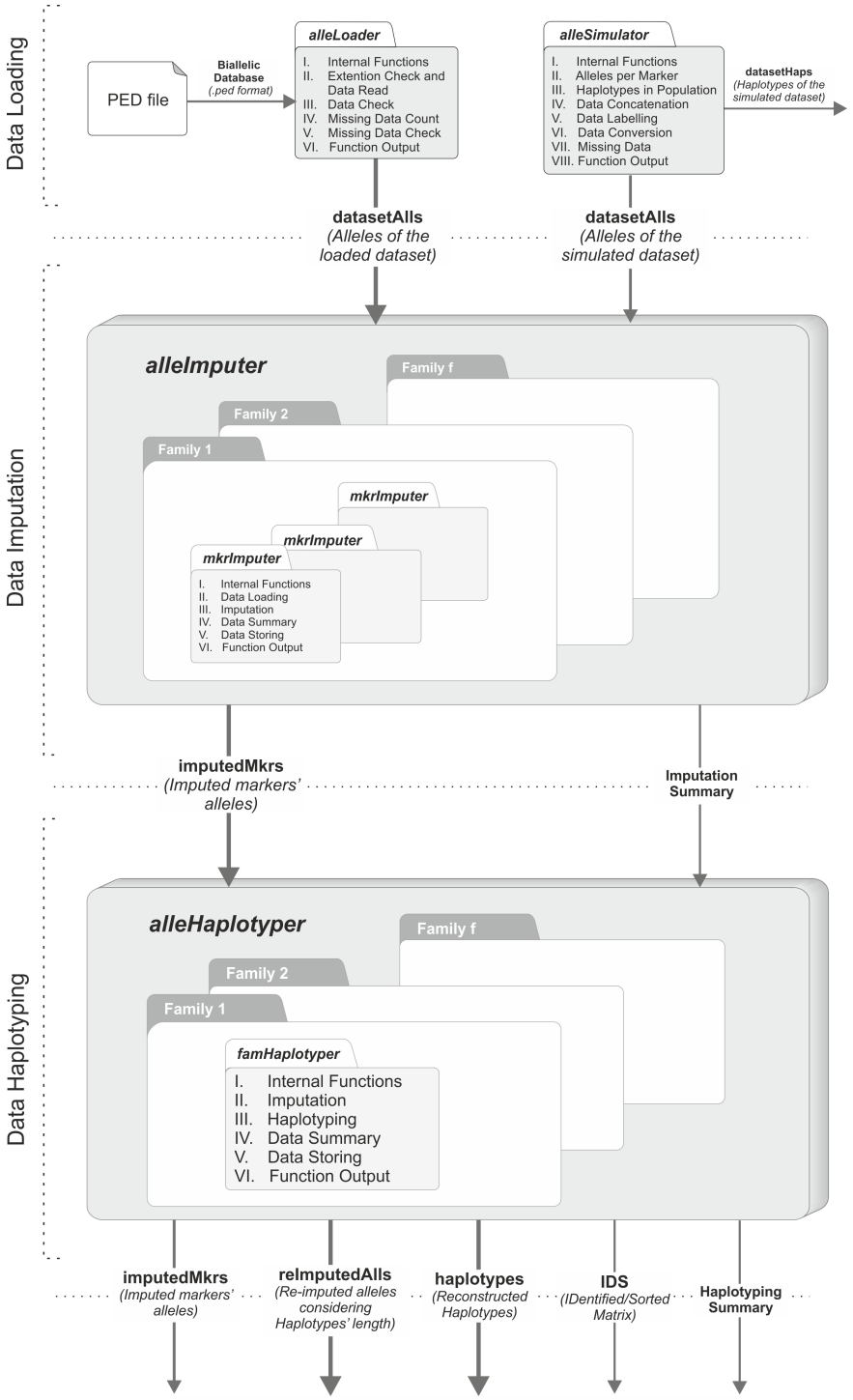


Figure 1: Graphical description of the package's workflow.

probability of parent/offspring missing genotypes, proportion of missing genotypes (genotyping errors) per individual, probability of being affected by disease, and recombination rate.

To perform the data simulation, this function goes through the following steps:

I. **Internal functions**: In this step all the necessary functions to simulate the data are loaded. These functions are: `labelMrk` (which creates the 'A', 'C', 'G', or 'T' character labels), `simHapSelection` (which selects h different haplotypes between the total number of possible haplotypes), `simOffspring` (which generates n offspring by selecting randomly one haplotype from each parent), `simOneFamily` (which simulates one family from a population containing the haplotypes `'popHaplos'`) and `simRecombHap` (which simulates the recombination of haplotypes).

II. **Alleles per marker**: The second step is the simulation of the number of different alleles per marker for the entire population (if the user does not supply them). The user can specify whether the alleles are letters (coded as `A`, `C`, `G`, or `T`) or if they are coded numerically. When the alleles are letters, only two possible different values are assigned to each marker; otherwise, between two and nine different values are randomly allotted.

III. **Haplotypes in population**: If there are many markers or alleles per marker, the number of possible haplotypes can be very large. By default, the number of possible different haplotypes generated by the function is limited to 1200, although the user can modify this value with the argument `nHaplos`.

IV. **Data concatenation**: In this step, the non-genetic information of all families and previously simulated data are concatenated.

V. **Data Labeling**: The fifth step is the labeling of the previous concatenated data ('`famID`', '`indID`', '`patID`', '`matID`', '`sex`', '`phen`', '`markers`', '`recombNr`', '`ParentalHap`', '`MaternalHap`').

VI. **Data conversion**: This step performs the conversion of previously generated data into a more suitable data type which will lead to a more efficient processing.

VII. **Missing data generation**: The seventh step is the insertion of missing values in the previous generated dataset (only when users require it). The missing values may be generated taking into account four different factors: *missParProb* (probability of parents' missing genotype), *missOffProb* (probability of offspring' missing genotype), *ungenotPars* (proportion of ungenotyped parents) and *ungenotOffs* (proportion of ungenotyped offspring).

VIII. **Function output**: The last step is the creation of a list containing two different data frames, for genotypes and haplotypes respectively. This may be useful to compare simulated haplotypes with later reconstructed haplotypes.

The following examples show how `alleSimulator` works:

alleSimulator Example 1: *Simulation of a family containing parental missing data.*

```
> simulatedFam1 <- alleSimulator(1, 2, 3, missParProb=0.3)
> simulatedFam1[[1]] # Alleles (genotypes) of the 1st simulated family

  famID indID patID matID sex phen Mk1_1 Mk1_2 Mk2_1 Mk2_2 Mk3_1 Mk3_2
1 FAM01     1     0     0   1    1     T     T     C     C  <NA>  <NA>
2 FAM01     2     0     0   2    1  <NA>  <NA>     T     C     C     G
3 FAM01     3     1     2   2    1     T     T     C     T     G     C
4 FAM01     4     1     2   1    1     T     T     C     T     C     C

> simulatedFam1[[2]] # 1st simulated family haplotypes (without missing values)

  famID indID patID matID sex phen Paternal_Hap Maternal_Hap
1 FAM01     1     0     0   1    1        T-C-G        T-C-C
2 FAM01     2     0     0   2    1        T-T-C        T-C-G
3 FAM01     3     1     2   2    1        T-C-G        T-T-C
4 FAM01     4     1     2   1    1        T-C-C        T-T-C
```

alleSimulator Example 2: *Simulation of a family containing offspring missing data.*

```
> simulatedFam2 <- alleSimulator(1, 2, 3, missOffProb=0.3)
> simulatedFam2[[1]] # Alleles (genotypes) of the 2nd simulated family
  famID indID patID matID sex phen Mk1_1 Mk1_2 Mk2_1 Mk2_2 Mk3_1 Mk3_2
1 FAM01     1     0     0   1    1     T     T     C     T     C     C
2 FAM01     2     0     0   2    2     T     C     C     C     C     T
3 FAM01     3     1     2   2    1     T     T     C     T  <NA>  <NA>
4 FAM01     4     1     2   2    1     T     T     C     C  <NA>  <NA>

> simulatedFam2[[2]] # 2nd simulated family haplotypes (without missing values)

  famID indID patID matID sex phen Paternal_Hap Maternal_Hap
1 FAM01     1     0     0   1    1        T-C-C        T-T-C
2 FAM01     2     0     0   2    2        T-C-C        C-C-T
3 FAM01     3     1     2   2    1        T-C-C        T-T-C
4 FAM01     4     1     2   2    1        T-C-C        T-C-C
```

Data loading

Before the data loading process, since **alleHap** can handle large amounts of missing data, users should check what kind of missing values will be loaded. If those values are different from "−9" or "−99", the parameter '"missingValues"' of alleLoader has to be updated with the corresponding value. Per example, if the file to be loaded has been codified with zeros as missing values, 'missingValues = 0' must be specified.

Data loading may be used with either simulated or actual genetic data. This stage has been implemented in the alleLoader function for '.ped' files, the default input format. This function tries to read family data from an R data frame or from an external file, to later pass it into the alleImputer and/or alleHaplotyper functions. For this purpose this function goes through these five steps:

I. **Loading of the internal function** recodeNA: This auxiliary function recodes pre-specified missing data as NA values.

II. **Extention check and data read**: In this step, the extension file is checked and if it has a '.ped' extension the dataset is loaded into R as a data frame. Should this not occur, the message *"The file must have a .ped extension"* is returned and the data will not be loaded. Then, if the file extension is appropriate, data is loaded and missing values (by default '-9' or '-99') are recoded as NAs (users may supply other codings values).

III. **Data check**: The third step counts the number of families, individuals, parents, children, males, females and markers of the dataset, as well as, it checks the ranges of 'Paternal IDs', 'Maternal IDs', 'genotypes' and 'phenotype' values.

IV. **Missing data count**: This step counts the missing/unknown data which may exist in either genetic data or subjects' identifiers.

V. **Function output**: In the final step, the dataset is returned as an R data frame, with the same structure as a PED file, with the variables renamed and the missing values correctly identified and coded. If 'dataSummary = TRUE' a summary of previous data counting, ranges, and missing values is printed to the screen.

The intended datasets must conform to the specifications of a PED file: in each row the first six variables correspond to 'family ID', 'subject ID', 'paternal ID', 'maternal ID', 'sex', and affection status ('phenotype'). The rest of the variables are the observed genotypes in each marker, where each marker comprises two other variables.

The following examples depict how alleLoader should be used:

alleLoader Example 1: *Loading of a dataset in PED format with alphabetical alleles (A, C, G, or T).*

```
> example1 <- file.path(find.package("alleHap"), "examples", "example1.ped")
> example1Alls <- alleLoader(example1)  # Loaded alleles of example 1

==========================================
===== alleHap package: version x.y.z ======
==========================================

Data have been successfully loaded from:
/home/nmr/R/x86_64-pc-linux-gnu-library/3.2/alleHap/examples/example1.ped

===== DATA COUNTING ======
Number of families: 50
Number of individuals: 227
Number of founders: 100
Number of children: 127
Number of males: 118
Number of females: 109
Number of markers: 12
==========================

======== DATA RANGES =========
Family IDs: [1,...,50]
Individual IDs: [1,...,8]
Paternal IDs: [0,1]
Maternal IDs: [0,2]
Sex values: [1,2]
```

```
Phenotype values: [1,2]
============================

========= MISSING DATA =========
Missing founders: 0
Missing ID numbers: 0
Missing paternal IDs: 0
Missing maternal IDs: 0
Missing sex: 0
Missing phenotypes: 0
Missing alleles: 0
Markers with missing values: 0
==============================

> example1Alls[1:9, 1:12]  # Alleles of the first 9 subjects

  famID indID patID matID sex phen Mk1_1 Mk1_2 Mk2_1 Mk2_2 Mk3_1 Mk3_2
1     1     1     0     0   1    1     T     T     C     T     A     A
2     1     2     0     0   2    1     A     T     C     G     C     C
3     1     3     1     2   1    2     A     T     G     T     A     C
4     1     4     1     2   2    1     A     T     C     G     A     C
5     1     5     1     2   2    1     A     T     C     G     A     C
6     2     1     0     0   1    1     A     T     A     G     A     G
7     2     2     0     0   2    1     G     T     A     C     A     G
8     2     3     1     2   2    1     G     T     A     C     A     G
9     2     4     1     2   1    1     A     G     C     G     G     G
```

alleLoader Example 2: *Loading of a dataset in PED format with numerical alleles*

```
> example2 <- file.path(find.package("alleHap"), "examples", "example2.ped")
> example2Alls <- alleLoader(example2, dataSummary=FALSE)  # Example 2
> example2Alls[1:6,]  # Alleles of the first 6 subjects

  famID indID patID matID sex phen Mk1_1 Mk1_2 Mk2_1 Mk2_2 Mk3_1 Mk3_2
1  1036     1     0     0   1    1   101  1601   101   102   501   502
2  1036     2     0     0   2    1   301   401   301   501   201   301
3  1036     3     1     2   1    2   301  1601   102   501   201   502
4  1036     4     1     2   1    2   301  1601   102   501   201   502
5  1239     1     0     0   1    1    NA    NA    NA    NA    NA    NA
6  1239     2     0     0   2    1    NA    NA    NA    NA    NA    NA
```

Allele imputation marker by marker

At this stage, the imputation of missing alleles in previously loaded/simulated datasets is performed. For this purpose, first a simple quality control of data is conducted and second a "marker by marker" allele imputation is carried out in those cases where possible. Both procedures are implemented in the `alleImputer` function where the corresponding operation can be reduced to the following steps:

1. **Internal functions**: In this step all the necessary functions to impute the data are loaded. The most important ones are:
   - `mkrImputer`, which performs the imputation in one marker. This function first receives as input data the alleles of that marker in one family, and then applies the quality control and makes imputation when possible, attending to the family structure as shown in Table 1. In the most simple cases, missing alleles in children are imputed only if a parent is homozygous. Missing alleles in a parent are imputed when a child is homozygous, or when the other parent has no missing values and alleles not present in that parent are found in some children.
   - `famImputer`, which applies `mkrImputer` sequentially to impute all the markers in a family.
   - `famsImputer`, which applies `famImputer` to all the families of the given data frame, returning a dataset with the same format and dimensions as the input data (with imputed values in those alleles where imputation has been possible).

2. **Data loading**: The second step tries to read genotypic data and the families information into a fully compatible format employing the `alleLoader` function. If this process is successful, data are stored in an R data frame with the same structure as a PED file.
3. **Imputation**: This is the most important step of the `alleImputer` function. First, marker by marker and then family by family, the imputation of the corresponding missing alleles is performed by the `mkrImputer` function in two stages: children imputation first and then parent imputation, as has been described.
4. **Data summary**: Once the imputation is done, a summary of the imputed data is collected. This summary contains information about the imputation process, i.e., number of imputed alleles, detected incidences (number of canceled markers due to problems detected in the quality control process), imputation rate (quotient of the imputed alleles to the number of originally missing alleles) and time consumed in the process.
5. **Data storing**: In this step, the imputed data are stored in the same path where the PED file was located. The generated new file will have the same name and extension as the original, ending as 'imputed.ped'.
6. **Function output**: In this final step, if '`dataSummary = TRUE`' the imputation summary is printed out. Imputed data is directly returned as an R data frame (with the same structure and dimensions as the input dataset). Incidence messages are shown if they are detected at the '`quality control`' phase.

The following examples show how `alleImputer` works:

alleImputer Example 1: *Deterministic imputation for familial data containing parental missing values.*

```
## Simulation of a family containing parental missing data
> simulatedFam1 <- alleSimulator(1, 2, 3, missParProb=0.6)
> simulatedFam1[[1]]  # Simulated alleles

  famID indID patID matID sex phen Mk1_1 Mk1_2 Mk2_1 Mk2_2 Mk3_1 Mk3_2
1 FAM01     1     0     0   1    1     A     G  <NA>  <NA>     A     G
2 FAM01     2     0     0   2    1  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
3 FAM01     3     1     2   1    1     G     G     G     G     G     G
4 FAM01     4     1     2   2    2     A     A     T     T     A     A

## Genotype imputation of previous simulated data
> imputedFam1 <- alleImputer(simulatedFam1[[1]], dataSummary=FALSE)
> imputedFam1['imputedMkrs'] # Imputed alleles (markers)

  famID indID patID matID sex phen Mk1_1 Mk1_2 Mk2_1 Mk2_2 Mk3_1 Mk3_2
1 FAM01     1     0     0   1    1     A     G     G     T     A     G
2 FAM01     2     0     0   2    1     G     A     G     T     G     A
3 FAM01     3     1     2   1    1     G     G     G     G     G     G
4 FAM01     4     1     2   2    2     A     A     T     T     A     A
```

alleImputer Example 2: *Deterministic imputation for familial data containing offspring missing values.*

```
## Simulation of two families containing offspring missing data
> simulatedFam2 <- alleSimulator(2, 2, 3, missOffProb=0.6)
> simulatedFam2[[1]]  # Simulated alleles

  famID indID patID matID sex phen Mk1_1 Mk1_2 Mk2_1 Mk2_2 Mk3_1 Mk3_2
1 FAM01     1     0     0   1    1     A     A     C     T     C     C
2 FAM01     2     0     0   2    1     A     G     T     T     C     C
3 FAM01     3     1     2   2    1     A     G  <NA>  <NA>  <NA>  <NA>
4 FAM01     4     1     2   1    1     A     G     T     T     C     C
5 FAM02     1     0     0   1    1     G     G     C     T     T     T
6 FAM02     2     0     0   2    2     A     G     C     C     T     T
7 FAM02     3     1     2   2    1     A     G     C     C  <NA>  <NA>
8 FAM02     4     1     2   2    1  <NA>  <NA>     C     T  <NA>  <NA>

## Genotype imputation of previous simulated data
> imputedFam2 <- alleImputer(simulatedFam2[[1]], dataSummary=FALSE)
> imputedFam2['imputedMkrs'] # Imputed alleles (markers)
```

```
  famID indID patID matID sex phen Mk1_1 Mk1_2 Mk2_1 Mk2_2 Mk3_1 Mk3_2
1 FAM01     1     0     0   1    1     A     A     C     T     C     C
2 FAM01     2     0     0   2    1     A     G     T     T     C     C
3 FAM01     3     1     2   2    1     A     G  <NA>     T     C     C
4 FAM01     4     1     2   1    1     A     G     T     T     C     C
5 FAM02     1     0     0   1    1     G     G     C     T     T     T
6 FAM02     2     0     0   2    2     A     G     C     C     T     T
7 FAM02     3     1     2   2    1     A     G     C     C     T     T
8 FAM02     4     1     2   2    1     G  <NA>     C     T     T     T
```

It must be taken into account that the `alleImputer` function makes the imputation for each marker without "looking" at the rest of the markers in the subject/family. Imputation results obtained with `alleImputer` improve when the rest of the markers come into consideration assuming that there is no recombination. This task is addressed by the function `alleHaplotyper`.

Data haplotyping

At this stage, the corresponding haplotypes of the pedigree database are generated. To accomplish this, based on the user's knowledge of the genomic region to be analysed, it is necessary to slice the data into non-recombinant chunks in order to perform the haplotype reconstruction in each one of them.

Depending on the existence of missing alleles in parents or/and children, we have considered four haplotyping scenarios. In the first one, there are no completely missing markers in parents, and children may be complete without missing alleles or may have full or partially missing data. In the second one, all of the parental markers may be entirely missing, and there are at least three children in the family without missing alleles. The third scenario is a mixture of the previous two: some markers have completely missing alleles in parents but are complete (without missing alleles) in at least three children; some markers have non-missing alleles in parents, with some missing values in children; and some markers may have no missing values in parents nor children. In the fourth scenario, parents have completely missing markers, and non-missing markers are available in only two children; in this scenario, deterministic reconstruction of the haplotypes is possible only in a small number of cases under some specific conditions.

Several algorithms have been developed in **alleHap** for the reconstruction of haplotypes and the imputation of missing alleles in each one of these scenarios.

The function `alleHaplotyper` identifies the adequate scenario in each case and applies the corresponding algorithm for imputing and haplotyping. The user does not have to worry about deciding what scenario corresponds to each family in the database, since the function takes care of it.

Users may choose among several icons in order to specify the non-identified and missing values in the haplotypes. It is also possible to define the character that will be used as a separator between the alleles for the corresponding haplotypes. By default, the non-identified/missing allele symbol is '?', and haplotypes will be joined without any separator symbol between their correspondig alleles.

The `alleHaplotyper` function constructs the haplotypes "family by family" taking into account the initially known genotypes as well as the genotypes already imputed by the function `alleImputer`, along with the matrix `IDS`. In order to generate the haplotypes, this function performs six successive steps:

I. **Loading of internal functions**: In this step, several functions are loaded, the most important ones being:
   - `famHaplotyper`, which carries out the haplotype reconstruction for each family data as follows:
     1) Receives as input data the matrix of imputed data returned by `alleImputer` for **one** family.
     2) Applies the adequate algorithm depending on the specific scenario of each family (according to the amount of genotypic information available).
     3) Returns: *a)* a matrix equal to the input matrix, but with the new imputed alleles, *b)* a matrix with the same dimensions as the previous one filled with 0's an 1's. The value 0 indicates a non-phased allele, and the value 1 represents a phased allele, and *c)* another matrix with two columns corresponding to the haplotypes found in each member of the family.
   - `famsHaplotyper`, which applies `famHaplotyper` sequentially to all the families in the dataset.
   - `summarizeData`, which generates a summary of the haplotyping process.

II. **Allele imputation marker by marker in each family**: This step calls the `alleImputer` function which performs the imputation marker by marker and then, family by family.

III. **Haplotyping**: This part is the most important of the `alleHaplotyper` function since it tries to solve the haplotypes when possible. The process is the following: once each family genotype has been imputed marker by marker, those markers containing two unique heterozygous alleles (both in parents and offspring) are excluded from the process. Then, a set of IDentified/Sorted (IDS) matrices is generated per family (one per subject), organized in an R array. Later, the internal function `famHaplotyper` tries to solve the haplotypes of each family, comparing the information between parents and children in an iterative and reciprocal way. When there are not genetic data in both parents, and there are two or more "unique" offspring (not twins or triplets), the internal functions `makeHapsFromThreeChildren` and `makeHapsFromTwoChildren` try to solve the remaining data. Finally, the HoMoZygosity (HMZ) matrix is updated, and the excluded markers are again included. *Even if both parental alleles are missing in each marker, it is possible in some cases to reconstruct all the family haplotypes, identifying the corresponding children's haplotypes, although* ***in certain cases their parental provenance will be unknown***.

IV. **Data summary**: Once the data haplotyping is done, a data summary is collected, containing a *re-imputation*[2] rate (after the haplotyping process), the proportions of phased and non-phased alleles, the proportion of full, partial and empty reconstructed haplotypes, and the time employed in the process.

V. **Data storing**: In this step, the *re-imputed* data are stored in the same path where the PED file was located, when data have been read from an external file. Two new files will be generated with the same name and extension as the original, but ending as '`re-imputed.ped`' and '`haplotypes.txt`', for the re-imputed genotypes and the reconstructed haplotypes, respectively.

VI. **Function output**: In this final step, a summary of the generated data may be printed out, if '`dataSummary=TRUE`'. All the results can be directly returned, whether '`invisibleOutput=FALSE`'. The list of results contains: `imputedMkrs` (with the preliminary imputed marker alleles), `IDS` (including the resulting IDentified/Sorted matrix), `reImputedAlls` (including the re-imputed alleles) and `haplotypes` (storing the reconstructed haplotypes), and `haplotypingSummary` (showing a summary of the haplotyping process). Incidence messages can also be shown if they are detected. These may be caused by haplotype recombination (detected on children), genotyping errors, or inheritance from non-declared parents.

The following example depicts how `alleHaplotyper` works:

alleHaplotyper Example 1: *Haplotype reconstruction of a simulated family with parental missing data.*

```
## Simulation of a family containing parental missing data
> simulatedFam1 <- alleSimulator(1, 9, 8, missParProb=0.9, missOffProb=0.3)

## Haplotype reconstruction of previous simulated data
> fam1List <- alleHaplotyper(simulatedFam1[[1]], dataSummary=FALSE)
> simulatedFam1[[1]]     # Original data

   famID indID patID matID sex phen Mk1_1 Mk1_2 Mk2_1 Mk2_2 Mk3_1 Mk3_2
1  FAM01     1     0     0   1    2  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
2  FAM01     2     0     0   2    1  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
3  FAM01     3     1     2   1    1     A     G     C     C  <NA>  <NA>
4  FAM01     4     1     2   2    1  <NA>  <NA>  <NA>  <NA>     A     G
5  FAM01     5     1     2   2    1     G     G     C     T     A     G
6  FAM01     6     1     2   1    2     G     G  <NA>  <NA>     A     G
7  FAM01     7     1     2   2    1  <NA>  <NA>  <NA>  <NA>     G     G
8  FAM01     8     1     2   1    1  <NA>  <NA>     C     C     G     G
9  FAM01     9     1     2   1    1  <NA>  <NA>     C     C     G     G
10 FAM01    10     1     2   2    1  <NA>  <NA>     C     C     G     G
11 FAM01    11     1     2   1    1     A     G     C     C  <NA>  <NA>

   Mk4_1 Mk4_2 Mk5_1 Mk5_2 Mk6_1 Mk6_2 Mk7_1 Mk7_2 Mk8_1 Mk8_2
1   <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
2   <NA>  <NA>  <NA>  <NA>  <NA>  <NA>     T     T  <NA>  <NA>
3      C     T  <NA>  <NA>     T     T  <NA>  <NA>     C     T
```

[2]We have called "*re-imputation*" to the second imputation process (performed by the `alleHaplotyper` function) in which new alleles (that have not been previously imputed by the `alleImputer` function) can be imputed.

```
4   <NA>  <NA>     C     G     C     C  <NA>  <NA>     C     T
5      C     T     C     G     C     C     C     T     C     T
6      C     T  <NA>  <NA>     C     C     C     T     C     T
7      C     C     G     G     C     T     T     T     T     T
8   <NA>  <NA>     C     G     T     T     T     T  <NA>  <NA>
9      C     T     C     G     T     T     T     T     C     T
10     C     T  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>     C     T
11  <NA>  <NA>     G     G  <NA>  <NA>     T     T  <NA>  <NA>

> fam1List['reImputedAlls']    # Re-imputed alleles

   famID indID patID matID sex phen Mk1_1 Mk1_2 Mk2_1 Mk2_2 Mk3_1 Mk3_2
1  FAM01     1     0     0   1    2     A     G     C     T     G     A
2  FAM01     2     0     0   2    1     G     G     C     C     G     G
3  FAM01     3     1     2   1    1     A     G     C     C     G     G
4  FAM01     4     1     2   2    1     G     G     T     C     A     G
5  FAM01     5     1     2   2    1     G     G     T     C     A     G
6  FAM01     6     1     2   1    2     G     G     T     C     A     G
7  FAM01     7     1     2   2    1     A     G     C     C     G     G
8  FAM01     8     1     2   1    1     A     G     C     C     G     G
9  FAM01     9     1     2   1    1     A     G     C     C     G     G
10 FAM01    10     1     2   2    1     A     G     C     C     G     G
11 FAM01    11     1     2   1    1     A     G     C     C     G     G

   Mk4_1 Mk4_2 Mk5_1 Mk5_2 Mk6_1 Mk6_2 Mk7_1 Mk7_2 Mk8_1 Mk8_2
1      C     T     G     C     T     C     T     C     T     C
2      T     C     C     G     T     C     T     T     C     T
3      C     T     G     C     T     T     T     T     T     C
4      T     C     C     G     C     C     C     T     C     T
5      T     C     C     G     C     C     C     T     C     T
6      T     C     C     G     C     C     C     T     C     T
7      C     C     G     G     T     C     T     T     T     T
8      C     T     G     C     T     T     T     T     T     C
9      C     T     G     C     T     T     T     T     T     C
10     C     T     G     C     T     T     T     T     T     C
11     C     C     G     G     T     C     T     T     T     T

> fam1List['haplotypes']  # Reconstructed haplotypes

famID indID patID matID sex phen     hap1     hap2
1  FAM01     1     0     0   1    2 ACGCGTTT GTATCCCC
2  FAM01     2     0     0   2    1 GCGTCTTC GCGCGCTT
3  FAM01     3     1     2   1    1 ACGCGTTT GCGTCTTC
4  FAM01     4     1     2   2    1 GTATCCCC GCGCGCTT
5  FAM01     5     1     2   2    1 GTATCCCC GCGCGCTT
6  FAM01     6     1     2   1    2 GTATCCCC GCGCGCTT
7  FAM01     7     1     2   2    1 ACGCGTTT GCGCGCTT
8  FAM01     8     1     2   1    1 ACGCGTTT GCGTCTTC
9  FAM01     9     1     2   1    1 ACGCGTTT GCGTCTTC
10 FAM01    10     1     2   2    1 ACGCGTTT GCGTCTTC
11 FAM01    11     1     2   1    1 ACGCGTTT GCGCGCTT

> fam1List['haplotypingSummary']  # Haplotyping Summary

  nAlls pPhasAlls pNonPhasAlls nHaps pFullHaps pEmptyHaps
1   176         1            0    22         1          0
  pPartialHaps newImputedAlleles reImputationRate haplotypingTime
1            0                55                1           0.357
```

alleHaplotyper Example 2: *Haplotype reconstruction of a family containing parental and offspring missing data from a PED file.*

```
## PED file path
> family2path <- file.path(find.package("alleHap"), "examples", "example3.ped")
```

```
## Loading of the ped file placed in previous path
> family2Alls <- alleLoader(family2path, dataSummary=FALSE)

## Haplotype reconstruction of previous loaded data
> family2List <- alleHaplotyper(family2Alls, dataSummary=FALSE)
> family2Alls  # Original data

  famID indID patID matID sex phen Mk1_1 Mk1_2 Mk2_1 Mk2_2 Mk3_1 Mk3_2
1     1     1     0     0   2    1     C     T  <NA>  <NA>  <NA>  <NA>
2     1     2     0     0   2    1  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
3     1     3     1     2   1    2     C     C     A     G     A     T
4     1     4     1     2   1    2     C     T     A     C  <NA>  <NA>
5     1     5     1     2   1    2     C     T     A     G     C     T
6     1     6     1     2   1    2     C     T     A     G     C     T
7     1     7     1     2   2    1  <NA>  <NA>  <NA>  <NA>     C     G

  Mk4_1 Mk4_2 Mk5_1 Mk5_2 Mk6_1 Mk6_2 Mk7_1 Mk7_2 Mk8_1 Mk8_2
1  <NA>  <NA>  <NA>  <NA>     A     C  <NA>  <NA>  <NA>  <NA>
2  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
3     A     A     G     T     A     C     A     C     A     G
4     A     T     C     G     C     C     C     T     C     G
5  <NA>  <NA>     G     T     A     C     A     C     A     A
6     A     A     G     T     A     C     A     C     A     A
7     A     T     C     G  <NA>  <NA>     C     T  <NA>  <NA>

> family2List['reImputedAlls']  # Re-imputed alleles

  famID indID patID matID sex phen Mk1_1 Mk1_2 Mk2_1 Mk2_2 Mk3_1 Mk3_2
1     1     1     0     0   2    1     C     T     G     C     T     G
2     1     2     0     0   2    1     C     T     A     A     A     C
3     1     3     1     2   1    2     C     C     G     A     T     A
4     1     4     1     2   1    2     T     C     C     A     G     A
5     1     5     1     2   1    2     C     T     G     A     T     C
6     1     6     1     2   1    2     C     T     G     A     T     C
7     1     7     1     2   2    1     T     T     C     A     G     C

  Mk4_1 Mk4_2 Mk5_1 Mk5_2 Mk6_1 Mk6_2 Mk7_1 Mk7_2 Mk8_1 Mk8_2
1     A     T     T     C     A     C     A     T     A     C
2     A     A     G     G     C     C     C     C     G     A
3     A     A     T     G     A     C     A     C     A     G
4     T     A     C     G     C     C     T     C     C     G
5     A     A     T     G     A     C     A     C     A     A
6     A     A     T     G     A     C     A     C     A     A
7     T     A     C     G     C     C     T     C     C     A

> family2List['haplotypes']  # Reconstructed haplotypes

  famID indID patID matID sex phen     hap1     hap2
1     1     1     0     0   2    1 CGTATAAA TCGTCCTC
2     1     2     0     0   2    1 CAAAGCCG TACAGCCA
3     1     3     1     2   1    2 CGTATAAA CAAAGCCG
4     1     4     1     2   1    2 TCGTCCTC CAAAGCCG
5     1     5     1     2   1    2 CGTATAAA TACAGCCA
6     1     6     1     2   1    2 CGTATAAA TACAGCCA
7     1     7     1     2   2    1 TCGTCCTC TACAGCCA

> fam1List['haplotypingSummary']  # Haplotyping Summary

  nAlls pPhasAlls pNonPhasAlls nHaps pFullHaps pEmptyHaps
1   112         1            0    14         1          0
  pPartialHaps newImputedAlleles reImputationRate haplotypingTime
1            0                34                1           0.098
```

Accuracy

Initial considerations

The **alleHap** package was originally thought to determine haplotypes from nuclear families, i.e. both parents and their offspring (several children). The program can reconstruct haplotypes even when the genotypes of both parents (or only one) are completely missing. However, it does not fit well with the typical situation of cattle breeding in which a single progenitor (male) has had offspring with many females; or vice versa when there is a female that has had offspring of several males.

Although in future versions of **alleHap** this functionality will be included, the following simulations have been developed using the standard context of nuclear families containing the same parents and a variable number of descendants.

We established the foregoing to compare the accuracy and performance of **alleHap** versus other software we have selected AlphaImpute (Hickey et al., 2012) and FImpute (Sargolzaei et al., 2014) programs, since both programs have similar characteristics to **alleHap**, namely both consider pedigree information for inferring haplotypes and imputing missing data. As such, the following comparisons have been performed regarding the proportion of genotyping errors, missing values, and phasing (haplotyping) errors for the three programs.

Simulation parameters

Data for nuclear families ranging from one to 15 children and 50 alleles per haplotype have been simulated. In each case, haplotypes have been generated for missing values in the children's alleles, 0.10 (10% of the children's alleles are missing) and 0.25 (25% of genotypes in children are missing). Likewise, families with missing rates for parental alleles from 0% (fully genotyped) to 100% (completely ungenotyped) were simulated. In order not to slow down the simulations, only 100 families were simulated for each case.

Incorrectly identified alleles per haplotype

Figure 2 shows the proportion of incorrectly identified alleles in each haplotype. As it can be appreciated, when using **alleHap** the ratio is always zero. In those cases when an allele can not be unequivocally identified it is left as a missing value. However, both AlphaImpute and FImpute have algorithms that use HMM and the information of the rest of the families to impute probabilistically.

When using AlphaImpute, it can be seen that when there is a low proportion of children and parents with missing genotypes, the rate of imputation errors is low (practically null when the parents are completely genotyped). However, as the rate of alleles of lost parents increases, the proportion of alleles wrongly accused also increases. This effect is greater the higher the number of children per family, which indicates that with more data AlphaImpute tends to impute more, but returning higher number of imputation errors.

With FImpute, it is observed that even when there are not missing alleles in parents or children, the program tends to erroneously impute a proportion of alleles (it does so in cases of heterozygous SNPs in which it is not possible to deterministically decide which haplotype belongs to each allele). The imputation error rate decreases as there are available larger families. In any case, the imputation error rates reach between 20% and 25% when parents have all alleles missing.

Missing alleles per haplotype

In Figure 3 it can be seen that the proportion of non-imputed alleles that remain after the application of the algorithm tends to be lower in **alleHap** than in AlphaImpute for all conditions. In addition, the greater the number of children available and the lower the rate of missing alleles in both parents and children, the lower the proportion of alleles remaining unallocated.

For FImpute, if the rate of missing alleles in parents is less than 75%, in the cases we have explored (missing allele rate in children up to 25%), there are practically no alleles left unallocated, that is almost all the alleles are imputed. However, as mentioned previously, FImpute has a high imputation error rate, which increases precisely with the rate of missing alleles in parents/children, and decreases with the number of families.

So, regarding the rate of missing alleles per haplotype, it could be said that **alleHap** is advantageous with respect to AlphaImpute, since it produces a smaller proportion of alleles that are finally left not imputed, and does not generate imputation errors. On the other hand, FImpute has an allele imputation rate higher than **alleHap**, but at the cost of making many more imputation errors.

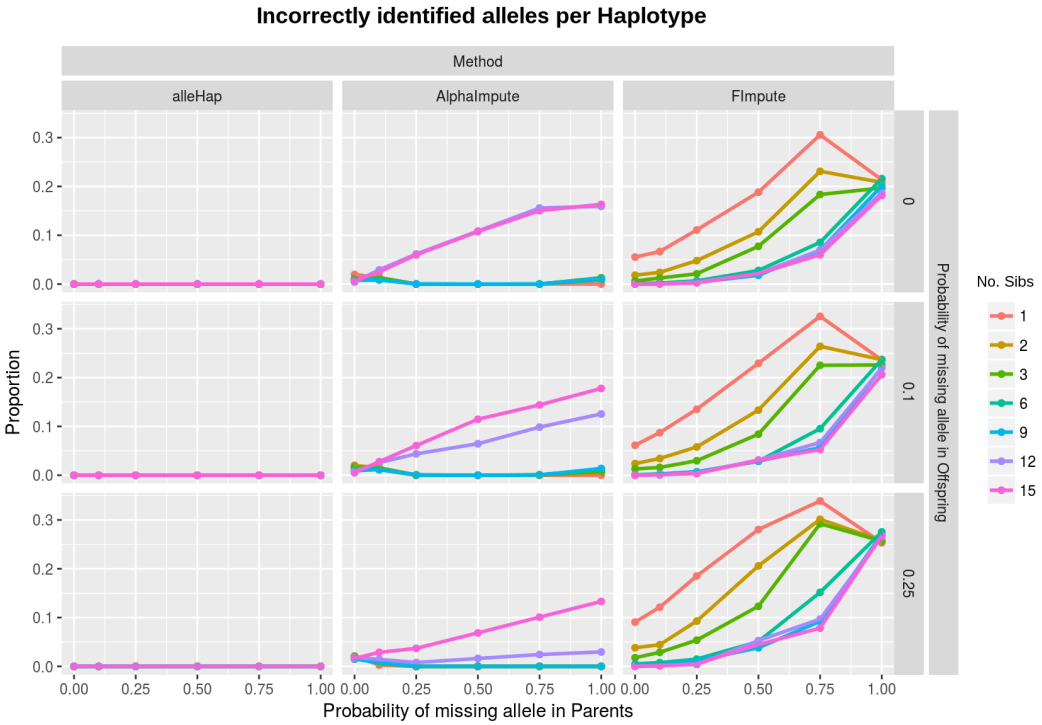


Figure 2: Proportiton of incorrectly identified alleles per haplotype.

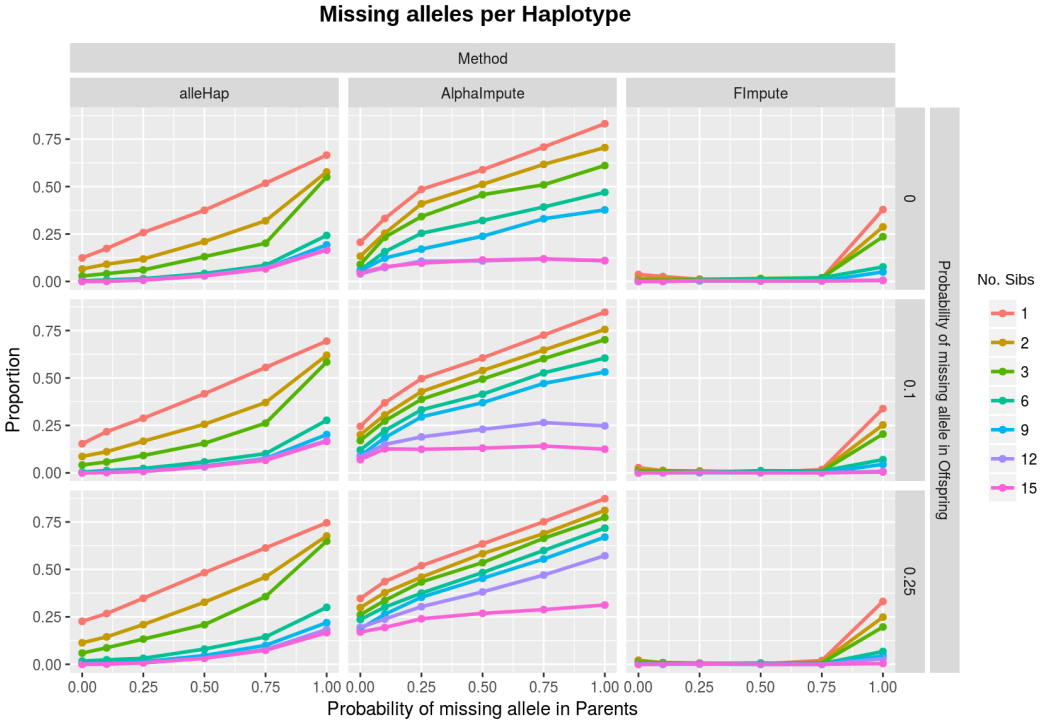


Figure 3: Proportion of missing alleles per haplotype.

Completely reconstructed haplotypes after imputation and haplotyping

In Figure 4 it is shown the proportion of haplotypes that were completely and correctly created with each method. It can be seen that for a large number of families with a low rate of missing alleles (under 10%), **alleHap** is able to completely and correctly reconstruct more than 90% of haplotypes.

Obviously, as the rate of missing alleles in parents and offspring increases, **alleHap**'s haplotyping rate decreases (if 50% of the parental alleles are missing, **alleHap** is able to completely and correctly reconstruct 25% of haplotypes, provided that large families are available).

When the allele missing rate in parents or children exceeds 10%, complete haplotype reconstruction rates are (in comparison to **alleHap**) generally lower with AlphaImpute and somewhat higher with FImpute. However, it should be noted that the completely reconstructed haplotypes by **alleHap** do not have any incorrectly allocated alleles.

In the case of FImpute, for an entirely reconstructed haplotype, we can not know whether reconstruction has been good (no errors), or if there have been misidentified alleles (could have a high proportion of misidentified alleles).

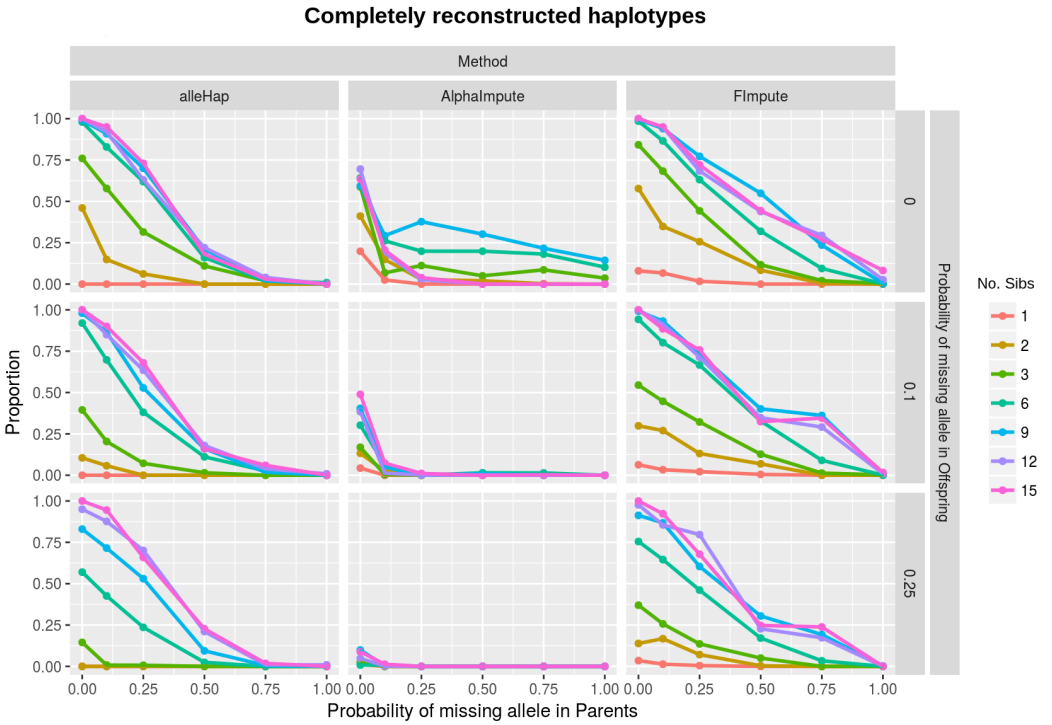


Figure 4: Proportion of fully reconstructed haplotypes after imputation and haplotyping.

Performance

Initial considerations

Since our package was mainly created for primase accuracy, i.e., the univocal treatment of family-based allelic databases, and not for processing large genomic data, it does not make much sense to perform a comparison on equal terms with programs that were implemented and compiled in other, faster languages.

In any case, we have carried out a benchmarking where the **alleHap** computing times were measured, evaluating the performance of the simulation, imputation, and haplotyping processes depending on two main factors: number of individuals and number of markers.

Computing times regarding the number of individuals

The simulations depending on the ***number of individuals*** had the following parameters: from 1 to 8000 individuals (two children per family), 3 markers (three allele pairs), two different alleles per marker, non-numerical alleles (A, C, G, or T), 25% of parental missing genotypes, 25% of children's missing genotypes, and 1200 different haplotypes in the population.

In Figure 5, it can be seen how haplotyping and imputation times grow linearly as the number of individuals increases while simulation times hardly grow. Therefore, it can be said that **alleHap** consumes very little time when using a small number of markers, even for a considerabe number of individuals.

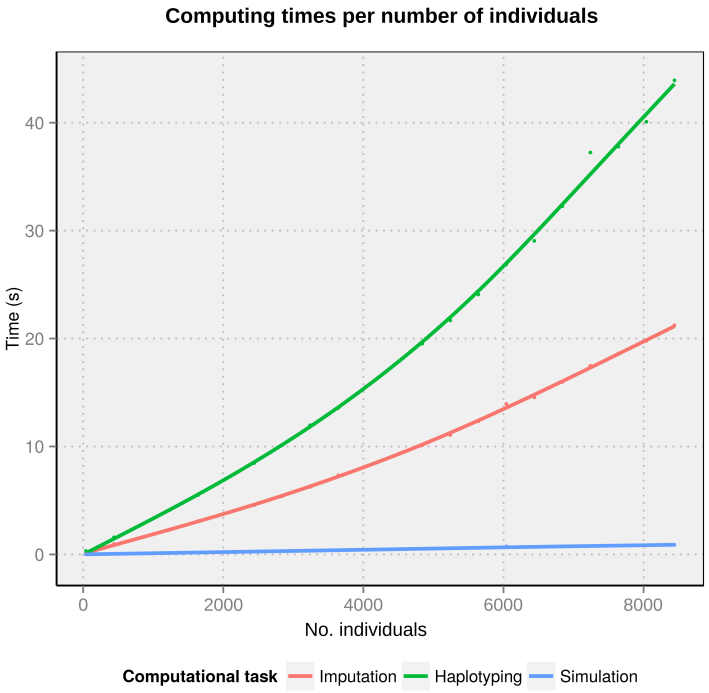


Figure 5: Computing times for simulation, imputation, and haplotyping, depending on the number of individuals.

Computing times regarding the number of markers

The simulations depending on the ***number of markers*** were developed using the following factors: from 1 to 5000 markers (from one allele pair to five thousand), one family (four individuals), two different alleles per marker, non-numerical alleles (A,C,G or T), 25% of parental missing genotypes, 25% of children's missing genotypes, and 1200 different haplotypes in the population.

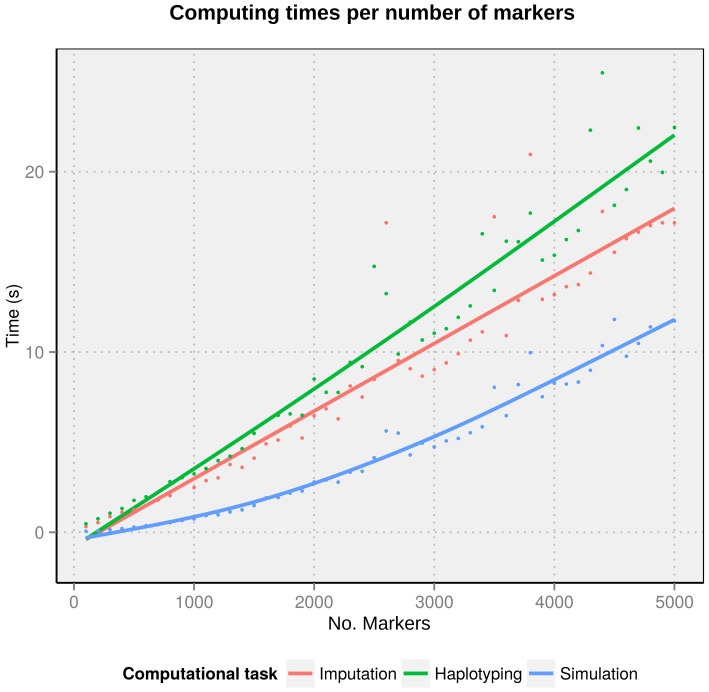


Figure 6: Computing times for simulation, imputation, and haplotyping, depending on the number of markers.

Figure 5 shows how simulation times grow in a non-linear way as the number of markers increase, while imputation and reconstruction times remain linear (even considering a large number of markers). Note that for this analysis only, it has been taken into account a family (containing four individuals), although it is presumable that for a larger number of indivuals this growth will also continue in linear manner.

Summary

We have developed an improved version of the R package **alleHap** for the imputation of alleles and the reconstruction of haplotypes in non-recombinant genomic regions using pedigree databases. The procedure is entirely deterministic and uses the information contained in the family structure to guide the process of imputation and reconstruction of haplotypes, without resorting to a reference panel. The package has two main functions `alleImputer` and `alleHaplotyper`.

The first one, `alleImputer`, makes allele imputation marker by marker, taking into account the alleles present in parents and siblings and considering that each individual (due to meiosis) should unequivocally have two alleles per marker, one inherited from each parent.

The function `alleHaplotyper` first calls `alleImputer` for an initial imputation of missing alleles and then, considering that there is no recombination (by comparing parental genotypes with children) determines the compatible haplotypes with the family structure. When an inconsistency is detected, `alleHaplotyper` reports an error message specifying if such inconsistency can be due to a possible recombination or to a genotyping error. Besides, the procedure of construction of haplotypes allows the imputation of those alleles that were not previously imputed by `alleImputer`. The function `alleHaplotyper` has been entirely rewritten with respect to previous versions of the package. Also the function `alleImputer` has been modified to include a new quality control procedure and to improve the presentation of a summary of results.

Genotypic information can be read from an R data frame or from a PED file by the function `alleLoader` designed specifically with this aim.

The package also includes the function `alleSimulator` for the simulation of pedigrees. This function handles many arguments, such as number of families, markers, alleles per marker, probability and proportion of missing genotypes, recombination rate, etc., and it generates an R data frame with two lists, one with the structure of a `.ped` file, and other with the haplotypes generated for each member of the simulated families. We have used `alleSimulator` for testing the performance of the other functions, with satisfactory results regarding imputation rate, haplotyping rate, and time of execution, even when handling large amounts of genetic data.

Future improvements of the package include the possibility of considering extended pedigrees (including grandparents, grandchildren, and other relatives) and to make inferences on haplotypes and missing alleles when these can not be deterministically derived.

Bibliography

T. Y. Berger-Wolf, S. I. Sheikh, B. DasGupta, M. V. Ashley, I. C. Caballero, W. Chaovalitwongse, and S. L. Putrevu. Reconstructing sibling relationships in wild populations. *Bioinformatics*, 23(13):49–56, 2007. [p36]

B. L. Browning and S. R. Browning. A unified approach to genotype imputation and haplotype-phase inference for large data sets of trios and unrelated individuals. *The American Journal of Human Genetics*, 84(2):210–223, 2009. [p35]

B. L. Browning and S. R. Browning. Haplotype phasing: Existing methods and new developments. *Nature Reviews Genetics*, 12(10):703–714, 2011a. [p35]

B. L. Browning and S. R. Browning. A fast, powerful method for detecting identity by descent. *The American Journal of Human Genetics*, 88(2):173–182, 2011b. [p35]

G. P. Consortium and others. An integrated map of genetic variation from 1,092 human genomes. *Nature*, 491(7422):56–65, 2012. [p35]

I. H. Consortium and others. A haplotype map of the human genome. *Nature*, 437(7063):1299–1320, 2005. [p35]

S. Das. *Minimac3*. Center for Statistical Genetics, University of Michigan, 2015. [p35]

P. I. de Bakker, G. McVean, P. C. Sabeti, M. M. Miretti, T. Green, J. Marchini, X. Ke, A. J. Monsuur, P. Whittaker, M. Delgado, and others. A high-resolution hla and snp haplotype map for disease association studies in the extended human mhc. *Nature genetics*, 38(10):1166–1172, 2006. [p35]

J. B. Endelman. Ridge regression and other kernels for genomic selection with R package rrblup. *Plant Genome*, 4:250–255, 2011. [p35]

M. H. Ferdosi, B. P. Kinghorn, J. H. Van der Werf, S. H. Lee, and C. Gondro. Hsphase: An R package for pedigree reconstruction, detection of recombination events, phasing and imputation of half-sib family groups. *BMC bioinformatics*, 15(1):1, 2014. [p35]

B. French and T. Lumley. *Haplo.ccs: Estimate Haplotype Relative Risks in Case-Control Data*, 2012. URL http://CRAN.R-project.org/package=haplo.ccs. R package version 1.3.1. [p35]

J. M. Hickey, B. P. Kinghorn, B. Tier, J. H. van der Werf, and M. A. Cleveland. A phasing and imputation method for pedigreed populations that results in a single-stage genomic evaluation. *Genetics Selection Evolution*, 44(1):9, 2012. [p47]

B. Howie and J. Marchini. *Using IMPUTE2 for Phasing of GWAS and Subsequent Imputation*. University of Chicago and University of Oxford, 2010. [p35]

S. JP and S. DJ. *Haplo.stats: Statistical Analysis of Haplotypes with Traits and Covariates When Linkage Phase Is Ambiguous*, 2016. URL http://CRAN.R-project.org/package=haplo.stats. R package version 1.7.7. [p35]

S. J. Mack, P. Cano, J. A. Hollenbach, J. He, C. K. Hurley, D. Middleton, M. E. Moraes, S. E. Pereira, J. H. Kempenich, E. F. Reed, M. Setterholm, A. G. Smith, M. G. Tilanus, M. Torres, M. D. Varney, C. E. M. Voorter, G. F. Fischer, K. Fleischhauer, D. Goodridge, W. Klitz, A.-M. Little, M. Maiers, S. G. E. Marsh, C. R. Müller, H. Noreen, E. H. Rozemuller, A. Sanchez-Mazas, D. Senitzer, E. Trachtenberg, and M. Fernandez-Vina. Common and well-documented hla alleles: 2012 update to the cwd catalogue. *Tissue Antigens*, 81(4):194–203, 2013. [p35]

N. Medina-Rodríguez, A. Santana, A. M. Wägner, and J. M. Quinteiro. allehap: An efficient algorithm to reconstruct zero-recombinant haplotypes from parent-offspring pedigrees. *BMC Bioinformatics*, 15(Suppl 3):A6, 2014. [p35]

S. Purcell, B. Neale, K. Todd-Brown, L. Thomas, M. A. Ferreira, D. Bender, J. Maller, P. Sklar, P. I. De Bakker, M. J. Daly, and others. Plink: a tool set for whole-genome association and population-based linkage analyses. *The American Journal of Human Genetics*, 81(3):559–575, 2007. [p37]

M. Sargolzaei, J. P. Chesnais, and F. S. Schenkel. A new approach for efficient genotype imputation using information from relatives. *BMC genomics*, 15(1):478, 2014. [p47]

V. Wimmer, T. Albrecht, H.-J. Auinger, and C.-C. Schoen. Synbreed: a framework for the analysis of genomic prediction data using R. *Bioinformatics*, 28(15):2086–2087, 2012. [p35]

Y. Xu and J. Wu. *Linkim: Linkage Information Based Genotype Imputation Method*, 2014. URL http://CRAN.R-project.org/package=linkim. R package version 0.1. [p35]

K. Zhang, F. Sun, and H. Zhao. Haplore: a program for haplotype reconstruction in general pedigrees without recombination. *Bioinformatics*, 21(1):90–103, 2005. [p35]

Nathan Medina-Rodríguez
Departament of Mathematics
IUMA - Information and Communication Systems
University of Las Palmas de Gran Canaria
Spain
nathan.medina@ulpgc.es

Ángelo Santana
Departament of Mathematics
University of Las Palmas de Gran Canaria
Spain
angelo.santana@ulpgc.es

Appendix

Genotype combinations

An example of the genotype combinations that may exist in a nuclear family (considering that each member should unequivocally have inherited two alleles) is depicted in Figure 7.

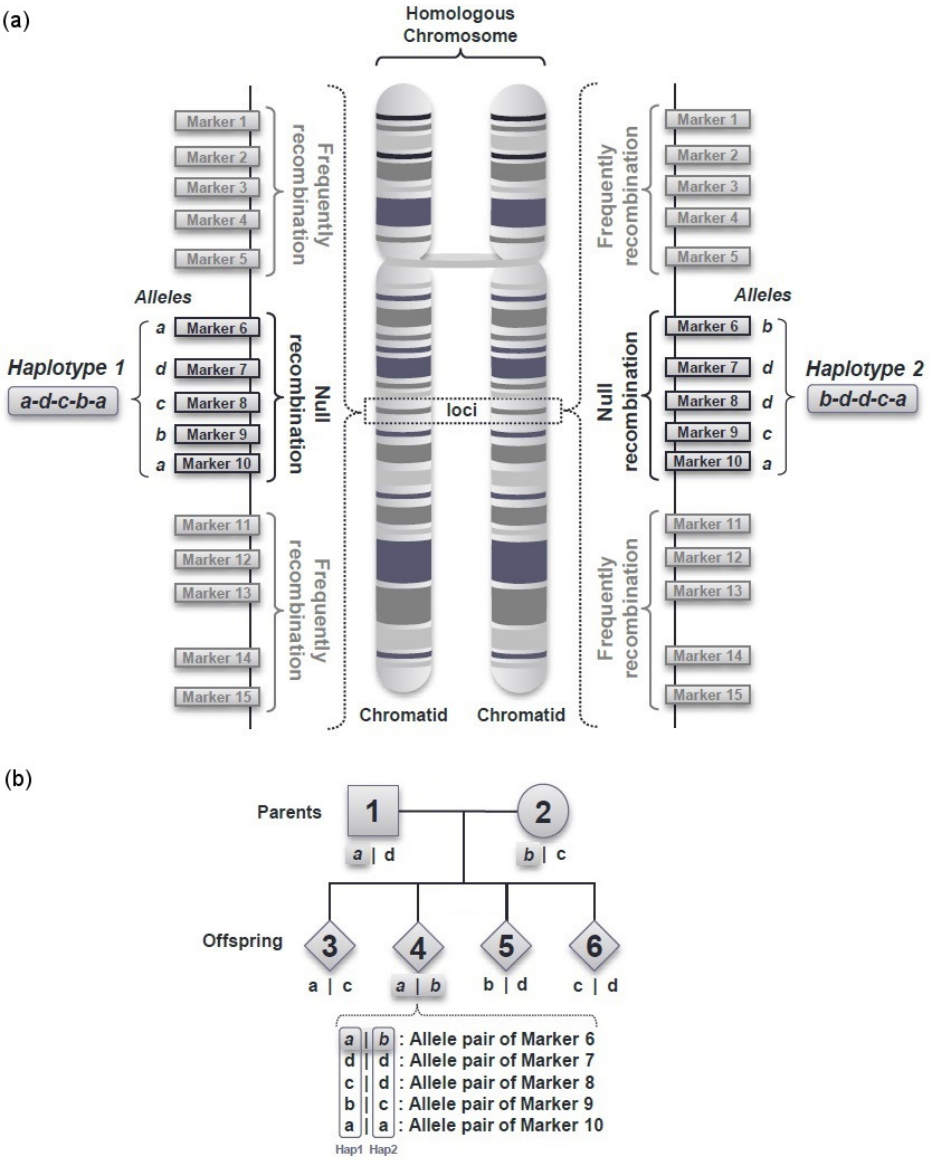


Figure 7: Graphical description of inherited genetic information. (a) Description of null and frequently recombination regions and their corresponding alleles/haplotypes located in a homologous chromosome. (b) Illustration of a parent-offspring pedigree containing 6 members: 2 parents and 4 children.

Genetic information can be arranged so that an allele may correspond to a single nucleotide (A, C, G, T) or a genomic nucleotide sequence. It has to be taken into account that the allele nomenclature in both cases would comprise different alphanumeric values chosen for a given purpose.

Used notation

As it has been described in the ***Basics*** section, our theoretical analysis is based on the differentiation of seven configurations, each one grouped considering the number of unique (or different) alleles per family. Thus, in order to clarify the genotype identification and to simplify posterior computations, when K markers are observed, the following notation has been used for describing the alleles in the i^{th}

subject of a family (being $i = 1$ for the father, $i = 2$ for the mother and $i > 2$ for the offspring):

$$\mathbf{A_i} = \begin{bmatrix} A_{11i} & A_{12i} & \dots & A_{1Ki} \\ A_{21i} & A_{22i} & \dots & A_{2Ki} \end{bmatrix} \tag{1}$$

Each column k of this matrix represents a marker, being (A_{1ki}, A_{2ki}) the pair of alleles identified in that marker. Either allele (or both) may be missing, and would then be denoted as *Not Available* (NA).

Associated to the previous matrix, we can define inheritance identifiers, which we have renamed as *IDentified/Sorted* (IDS) for all subjects, being IDS_{hki} the corresponding identifier of marker k in the individual i. The resulting matrix containing the IDS_{hki} values can be defined as:

$$\mathbf{IDS_i} = \begin{bmatrix} IDS_{11i} & IDS_{12i} & \dots & IDS_{1Ki} \\ IDS_{21i} & IDS_{22i} & \dots & IDS_{2Ki} \end{bmatrix} \tag{2}$$

If all terms in the matrix IDS_i are 0, the phase of each allele is unknown. In turn, when all terms are equal to 1, the alleles are phased and the rows of the matrix A_i can be read as the haplotypes of the i^{th} subject of the family. In this way, being $h = 1, 2$, we have:

$$IDS_{hki} = \begin{cases} 0 & \text{if allele } A_{hki} \text{ does not belong to} \\ & \text{haplotype } h, \text{ or is missing} \\ 1 & \text{if allele } A_{hki} \text{ belongs to haplotype } h \end{cases} \tag{3}$$

When familial genotypes are read, the matrices IDS_i are initially equal to 0 for all members, as the genotype phase is unknown.

At the data haplotyping stage, the main objective is to order the A_{hki} alleles in each marker of every subject in such a way that the IDS_i matrices may contain as values equal to 1 as possible. When the h row in IDS_i is completely (or partially) filled with ones, the corresponding h^{th} row in the A_i matrix can be deterministically phased.

In the same way as described above, a vector of *Homozygosity* (HMZ) identifiers per individual was defined. Therefore, $HMZ_{ik} = 1$ if individual i is homozygous in marker k (i.e. identical alleles), and $HMZ_{ik} = 0$ if the subject is heterozygous. Consequently, considering a given individual of family, the HMZ vector is defined as:

$$\mathbf{HMZ_i} = \begin{bmatrix} HMZ_{i1} & HMZ_{i2} & \dots & HMZ_{ik} \end{bmatrix} \tag{4}$$

Data imputation

As long as one child was genotyped, in certain cases it is possible for an unequivocal imputation of missing genotypes both in parents and children. For this purpose, first is conducted a "marker by marker" *Quality Control* (QC) of the genetic data and second, imputation is performed, in those cases where possible. The QC rules which has been considered are the following:

1. There cannot be more than two different homozygous children in a family.
2. If there are two different homozygous children in the family, there cannot be a different allele in any other family member.
3. Considering all the individuals in the family, there can be at most four different alleles in a marker.
4. If a family has four different alleles in a marker, no child can be homozygous.
5. If there are three of more unique heterozygous children, they cannot share a common allele.
6. There cannot be more than four genotypically different children per marker and family.
7. If a child has the same alleles as one of his/her parent, there can only be at most three different alleles in the family.
8. When parental alleles are not missing:
   (a) Each child must have at least a common allele with each parent.
   (b) A child can not have an allele not present in any of the parents.

In the above, when referring to families with more than one child, it has been considered that children have different genotypes each other. This is because of two or more children share the same genotype in all the markers. For the purposes of genotype identification, he/she will count as an only child (since will not provide different or new information).

Throughout this procedure, a *"marker by marker"* allele imputation is performed as follows:

1. Imputation of children's genotypes: It is identified which allele has been inherited from the father and which from the mother. If a parent has a homozygous genotype, the corresponding allele is imputed in all children with missing alleles (which do not already have this allele). Moreover, if both parents are homozygous, all children with missing alleles are readily imputed.
2. Imputation of parental genotypes: Given a reference child, it is determined which allele has been transmitted to such child:
   (a) If a child has a homozygous genotype, the allele is imputed in that parent that do not already have this allele.
   (b) If a parent has missing alleles and the other not, and there are heterozygous children, the alleles present in those children (which are not located in the non-missing parent) are imputed to the parent with missing alleles.

Assuming that genotypic markers are part of the same haplotype, i.e., there is no recombination between markers, we have considered that when missing data occurs in a subject's marker, missingness affects both alleles (i.e. each marker is fully missing for the given subject); but if the subject is a child and a parent is homozygous at the same marker (say G/G), only one allele will be imputed in such a child by `alleImputer` function. Thus, the child's genotype would be G/NA (where NA stands for missing value). The same would occur if a fully missing marker is located in a parent and there is a homozygous child in that marker.

Data haplotying

The haplotyping procedure begins by considering only the offspring, trying to identify/sort the alleles in each marker in such a way that the allele in the first row of the matrix, A_i, be the one inherited from the father (see ***Used notation*** section), and the allele in the second row be the inherited from the mother. So, if all the markers are sorted this way, the first row of the matrix A_i would inherit the first haplotype from the father and the second one from the mother. Once these haplotypes have been found in children, they can be readily identified in parents. What complicates this idea and makes difficult its direct application is the fact that in some cases both parents and a child can share the same genotype (say G/T for the three subjects), and therefore it is not possible to know which allele has been inherited from which parent. Thus, we considered these four scenarios for the haplotyping procedure:

- Scenario 1: There are no fully missing markers in parents.
- Scenario 2: There are missing markers in parents.
- Scenario 3: Mixture of the previous scenarios.
- Scenario 4: There are missing markers in parents and only two fully genotyped children.

In the first one there are no fully missing markers in parents, and children may be completely without missing alleles or may have full or partially missing data. In the second one we have taken into account that all of the parental markers are fully missing and there are at least three children in the family without missing alleles. The third scenario is a mixture of the previous two: some markers have parents with fully missing alleles and at least three complete children (without missing alleles). Some markers have non-missing alleles in parents, with some missing values in children, and some markers may have no missing values in parents nor in children. Finally, in the fourth scenario we show the conditions in which alleles can be deterministically phased with only two children, when parents have completely missing markers.

Furthermore there may be missing alleles in parents or children that prevent the determination the provenance of some alleles in some markers. In particular, if both parents have all the alleles missing in a marker it is impossible to determine the provenance of the alleles of that marker in the children, at least if there are less than three children in the family. When Scenario 2 occurs the family has three or more children, if there are no missing alleles in at least three children, deterministic phasing can be carried out even when parents are fully missing. Also, in Scenario 4, for the particular case of having only two children, if parental alleles are available in some markers and in the other markers are fully missing, it is possible (under certain conditions) to determine the alleles' phase in those missing markers.

Visualization of Regression Models Using visreg

by Patrick Breheny and Woodrow Burchett

Abstract Regression models allow one to isolate the relationship between the outcome and an explanatory variable while the other variables are held constant. Here, we introduce an R package, **visreg**, for the convenient visualization of this relationship via short, simple function calls. In addition to estimates of this relationship, the package also provides pointwise confidence bands and partial residuals to allow assessment of variability as well as outliers and other deviations from modeling assumptions. The package provides several options for visualizing models with interactions, including lattice plots, contour plots, and both static and interactive perspective plots. The implementation of the package is designed to be fully object-oriented and interface seamlessly with R's rich collection of model classes, allowing a consistent interface for visualizing not only linear models, but generalized linear models, proportional hazards models, generalized additive models, robust regression models, and many more.

Introduction

In simple linear regression, it is both straightforward and extremely useful to plot the regression line. The plot tells you everything you need to know about the model and what it predicts. It is common to superimpose this line over a scatter plot of the two variables. A further refinement is the addition of a confidence band. Thus, in one plot, the analyst can immediately assess the empirical relationship between x and y in addition to the relationship estimated by the model and the uncertainty in that estimate, and also assess how well the two agree and whether assumptions may be violated.

Multiple regression models address a more complicated question: what is the relationship between an explanatory variable and the outcome as the other explanatory variables are held constant? This relationship is just as important to visualize as the relationship in simple linear regression, but doing so is not nearly as common in statistical practice.

As models get more complicated, it becomes more difficult to construct these sorts of plots. With multiple variables, we cannot simply plot the observed data, as this does not hold the other variables constant. Interactions among variables, transformations, and non-linear relationships all add extra barriers, making it time-consuming for the analyst to construct these plots. This is unfortunate, however – as models grow more complex, there is an even greater need to represent them with clear illustrations.

In this paper, we aim to eliminate the hurdle of implementation through the development of a simple interface for visualizing regression models arising from a wide class of models: linear models, generalized linear models, robust regression models, additive models, proportional hazards models, and more. We implement this interface in R and provide it as the package **visreg**, publicly available from the Comprehensive R Archive Network. The purpose of the package is to automate the work involved in plotting regression functions, so that after fitting one of the above types of models, the analyst can construct attractive and illustrative plots with simple, one-line function calls. In particular, **visreg** offers several tools for the visualization of models containing interactions, which are among the easiest to misinterpret and the hardest to explain.

It is worth noting that there are two distinct goals involved in plotting regression models: illustrating the fitted model visually and diagnosing violations of model assumptions through examination of residuals. The approach taken by **visreg** is to construct a single plot that simultaneously addresses both goals. This is not a new idea. Indeed, this project was inspired by the work of Trevor Hastie, Robert Tibshirani, and Simon Wood, who have convincingly demonstrated the utility of these types of plots in the context of generalized additive models (Hastie and Tibshirani, 1990; Wood, 2006).

In particular, **visreg** offers partial residuals, which can be defined for any regression model and are easily superimposed on visualization plots. Partial residuals are widely useful in detecting many types of problems, although several authors have pointed out that they are not without limitations (Mallows, 1986; Cook, 1993). Various extensions and modifications of partial residuals have been proposed, and there is an extensive literature on regression diagnostics (Belsley et al., 1980; Cook and Weisberg, 1982); indeed, many diagnostics are specific to the type of model (e.g., Pregibon, 1981; Grambsch and Therneau, 1994; Loy and Hofmann, 2013). Partial residuals are a useful, easily generalized idea that can applied to virtually any type of model although it is certainly worth being aware of other types of diagnostics that are specific to the modeling framework in question.

There are a number of R packages that offer functions for visualizing regression models, including

rms (Harrell, 2015), **rockchalk** (Johnson, 2016), **car** (Fox and Weisberg, 2011), **effects** (Fox, 2003), and, in base R, the `termplot` function. The primary advantage of **visreg** over these alternatives is that each of them is specific to visualizing a certain class of model, usually `lm` or `glm`. **visreg**, by virtue of its object-oriented approach, works with any model that provides a `predict` method – meaning that it can be used with hundreds of different R packages as well as user-defined model classes. We also feel that **visreg** offers a simpler interface and produces nicer-looking plots, but admit that beauty is in the eye of the beholder. Nevertheless, there are situations in which each of these packages are very useful and offer some features that others do not, such as greater flexibility for other types of residuals (**car**) and better support for visualizing three-way interactions (**effects**).

Each type of model has different mathematical details. All models, however, describe how the response is expected to vary as a function of the explanatory variables. In R, this is implemented for an extensive catalog of models that provide an associated `predict` method. Although there are no explicit rules forcing programmers to write `predict` methods for a given class in a consistent manner, there is a widely agreed-upon convention to follow the general syntax of `predict.lm`. It is this abstraction upon which **visreg** is based: the use of object-oriented programming to provide a single tool with a consistent interface for the convenient visualization of a wide array of models.

There are thousands of R packages, many of which provide an implementation of some type of model. It is impossible for any programmer or team of programmers to write an R package that is familiar with the details of all of them. However, the encapsulation and abstraction offered by an object-oriented programming language allow for an elegant solution to this problem. By passing a fitted model object to **visreg**, we can call the `predict` method provided by that model class to obtain appropriate predictions and standard errors without needing to know any of the details concerning how those calculations work for that type of model; the same applies to construction of residuals through the `residual` method.

The only other R package that we are aware of that provides this kind of object-oriented flexibility is **plotmo** by Stephen Milborrow. The **visreg** and **plotmo** projects were each started independently around the year 2011 and have developed into mature, widely used packages for model visualization. The organization and syntax of the packages is quite different, but both are based on the idea of using the generic `predict` and `residuals` methods provided by a model class to offer a single interface capable of visualizing virtually any type of model. The primary difference between the two packages is that **plotmo** separates the visualization of models and the plotting of residuals, constructed using the `plotmo()` and `plotres()` functions, respectively, while as mentioned earlier, **visreg** combines the two into a single plot (**plotmo** offers an option to superimpose the unadjusted response onto a plot, but this is very different from plotting partial residuals). Furthermore, as one would expect, each package offers a few options that the other does not. For example, **plotmo** offers the ability to construct partial dependence plots (Hastie et al., 2009), while **visreg** offers options for contrast plots and what we call "cross-sectional" plots (Figs. 6, 7, and 8). Broadly speaking, **plotmo** is somewhat more oriented towards machine learning-type models, while **visreg** is more oriented towards regression models, though both packages can be used for either purpose. In particular, **plotmo** supports the `X,y` syntax used by packages like `glmnet`, which is more popular among machine learning packages, while **visreg** focuses exclusively on models that use a formula-based interface.

The outline of the paper is as follows. In "Conditional and contrast plots", we explicitly define the relevant mathematical details for what appears in **visreg**'s plots. The remainder of the article is devoted to illustrating the interface and results produced by the software in three extensions of simple linear regression: multiple (additive) linear regression models, models that possess interactions, and finally, other sorts of models, such as generalized linear models, proportional hazards models, random effect models, random forests, etc.

Conditional and contrast plots

We begin by considering regression models, where all types of **visreg** plots are well-developed and clearly defined. At the end of this section, we describe how these ideas can be extended generically to any model capable of making predictions.

In a regression model, the relationship between the outcome and the explanatory variables is expressed in terms of a linear predictor η:

$$\boldsymbol{\eta} = \mathbf{X}\boldsymbol{\beta} = \sum_j \mathbf{x}_j \beta_j, \tag{1}$$

where $\mathbf{x}_j$ is the jth column of the design matrix $\mathbf{X}$. For the sake of clarity, we focus in this section on linear regression, in which the expected value of the outcome $\mathbb{E}(Y_i)$ equals η_i; extensions to other, nonlinear models are discussed in "Other models". In the absence of interactions (see "Linear models

with interactions"), the relationship between X_j and Y is neatly summarized by β_j, which expresses the amount by which the expected value of Y changes given a one-unit change in X_j.

Partial residuals are a natural multiple regression analog to plotting the observed x and y in simple linear regression. Partial residuals were developed by Ezekiel (1924), rediscovered by Larsen and McCleary (1972), and have been discussed in numerous papers and textbooks ever since (Wood, 1973; Atkinson, 1982; Kutner et al., 2004). Letting $\mathbf{r}$ denote the vector of residuals for a given model fit, the partial residuals belonging to variable j are defined as

$$\mathbf{r}_j = \mathbf{y} - \mathbf{X}_{-j}\widehat{\boldsymbol{\beta}}_{-j} \tag{2}$$

$$= \mathbf{r} + \mathbf{x}_j\widehat{\boldsymbol{\beta}}_j, \tag{3}$$

where the $-j$ subscript refers to the portion of $\mathbf{X}$ or $\boldsymbol{\beta}$ that remains after the jth column/element is removed.

The reason partial residuals are a natural extension to the multiple regression setting is that the slope of the simple linear regression of $\mathbf{r}_j$ on $\mathbf{x}_j$ is equal to the value $\widehat{\boldsymbol{\beta}}_j$ that we obtain from the multiple regression model (Larsen and McCleary, 1972).

Thus, it would seem straightforward to visualize the relationship between X_j and Y by plotting a line with slope β_j through the partial residuals. Clearly, however, we may add any constant to the line and to $\mathbf{r}_j$ and the above result would still hold. Nor is it obvious how the confidence bands should be calculated.

We consider asking two subtly different questions about the relationship between X_j and Y:

(1) What is the relationship between $\mathbb{E}(Y)$ and X_j given $\mathbf{x}_{-j} = \mathbf{x}^*_{-j}$?

(2) How do changes in X_j relative to a reference value x^*_j affect $\mathbb{E}(Y)$?

The biggest difference between the two questions is that the first requires specification of some $\mathbf{x}^*_{-j}$, whereas the second does not. The reward for specifying $\mathbf{x}^*_{-j}$ is that specific values for the predicted $\mathbb{E}(Y)$ may be plotted on the scale of the original variable Y; the latter type of plot can address only relative changes. Here, we refer to the first type of plot as a *conditional plot*, and the second type as a *contrast plot*. As we will see, the two questions produce regression lines with identical slopes, but with different intercepts and confidence bands. It is worth noting that these are not the only possible questions; other possibilities, such as "What is the marginal relationship between X_j and Y, integrating over $\mathbf{X}_{-j}$?" exist, although we do not explore them here.

For a contrast plot, we consider the effect of changing X_j away from an arbitrary point x^*_j; the choice of x^*_j thereby determines the intercept, as the line by definition passes through $(x^*_j, 0)$. The equation of this line is $y = (x - x^*_j)\widehat{\beta}_j$. For a continuous X_j, we set x^*_j equal to $\bar{x}_j$. The confidence interval at the point $x_j = x$ is based on

$$V(x) = \mathbb{V}\left\{\hat{\eta}(x) - \hat{\eta}(x^*_j)\right\} = (x - x^*_j)^2\mathbb{V}(\widehat{\beta}_j).$$

When X_j is categorical, we plot differences between each level of the factor and the reference category (see Figure 3 for an example); in this case, we are literally plotting contrasts in the classical ANOVA sense of the term (hence the name). Our usage of the term "contrast" for continuous variables is somewhat looser, but still logical in the sense that it estimates the contrast between a value of X_j and the reference value.

For a conditional plot, on the other hand, all explanatory variables are fully specified by x and $\mathbf{x}^*_{-j}$. Let $\lambda(x)^T$ denote the row of the design matrix that would be constructed from $x_j = x$ and $\mathbf{x}^*_{-j}$. Then the equation of the line is $y = \lambda(x)^T\widehat{\boldsymbol{\beta}}$ and the confidence interval at x is based on

$$V(x) = \mathbb{V}\left\{\lambda(x)^T\widehat{\boldsymbol{\beta}}\right\} = \lambda(x)^T\mathbb{V}(\widehat{\boldsymbol{\beta}})\lambda(x).$$

In both conditional and contrast plots, the confidence interval at x is then formed around the estimate in the usual manner by adding and subtracting $t_{n-p,1-\alpha/2}\sqrt{V(x)}$, where $t_{n-p,1-\alpha/2}$ is $1-\alpha/2$ quantile of the t distribution with $n-p$ degrees of freedom. Examples of contrast plots and conditional plots are given in Figures 2 and 3. Both plots depict the same relationship between wind and ozone level as estimated by the same model (details given in the following section). Note the difference, however, in the vertical scale and confidence bands. In particular, the confidence interval for the contrast plot has zero width at x^*_j; all other things remaining the same, if we do not change X_j, we can say with certainty that $\mathbb{E}(Y)$ will not change either. There is still uncertainty, however, regarding the

actual value of $\mathbb{E}(Y)$, which is illustrated in the fact that the confidence interval of the conditional plot has positive width everywhere.

This description of confidence intervals focuses on Wald-type confidence intervals of the form of estimate $\pm$ multiple of the standard error, constructed on the scale of the linear predictor. This is the most common type of interval provided by modeling packages in R, and the only one for which a widely agreed-upon, object-oriented consensus has emerged in terms of what the `predict` method returns. For this reason, this is usually the only type of interval available for plotting by **visreg**. However, it should be noted that these intervals are common for their convenience, not due to superiority; it is typically the case that more accurate confidence intervals exist (see, for example, Efron, 1987; Withers and Nadarajah, 2012). In principle, one could plot other types of intervals, but **visreg** does not calculate intervals itself so much as plot the intervals that the modeling package returns. Thus, unless the modeling package provides methods for calculating other types of intervals, **visreg** is restricted to plotting Wald intervals.

Contrast plots can only be constructed for regression-based models, as they explicitly require an additive decomposition in terms of a design matrix and coefficients. Conditional plots, however, can be constructed for any model that produces predictions. Denote this prediction $f(\mathbf{x})$, where $\mathbf{x}$ is a vector of predictors for the model. Writing this as a one-dimensional function of predictor j with the remaining predictors fixed at $\mathbf{x}^*_{-j}$, let us express this prediction as $f(x|\mathbf{x}^*_{-j})$. In a conditional plot, the partial residuals for predictor j are

$$\begin{aligned}\mathbf{r}_j &= \mathbf{r} + x_j\widehat{\beta}_j + \mathbf{x}^*_{-j}\widehat{\boldsymbol{\beta}}_{-j} \\ &= \mathbf{r} + f(x|\mathbf{x}^*_{-j}),\end{aligned}$$

which offers a clear procedure for constructing the equivalent of partial residual for general prediction models. Note that this construction requires the model class to implement a `residuals` method. If a model class lacks a `residuals` method, **visreg** will still produce a plot, but must omit the partial residuals; see "Non-regression models" for additional details. Likewise, **visreg** requires the `predict` method for the model class to return standard errors in order to plot confidence intervals; see "Hierarchical and random effect models" for an example in which standard errors are not returned.

It is worth mentioning that **visreg** is only concerned with confidence bands for the conditional mean $\mathbb{E}(Y|\mathbf{X})$, not "prediction intervals" that have a specified probability of containing a future outcome Y observed for a certain value of $\mathbf{X}$. Unlike standard errors for the mean, very few model classes in R offer methods for calculating such intervals – indeed, such intervals are often not well-defined outside of classical linear models.

Additive linear models

We are now ready to describe the basic framework and usage of **visreg**. In this section, we will fit various models to a data set involving the relationship between air quality (in terms of ozone concentration) and various aspects of weather in the standard R data set `airquality`.

Basic framework

The basic interface to the package is the function **visreg**, which requires only one argument: the fitted model object. So, for example, the following code produces Figure 1:

```
fit <- lm(Ozone ~ Solar.R + Wind + Temp, data=airquality)
visreg(fit)
```

By default, **visreg** provides conditional plots for each of the explanatory variables in the model. For the conditioning, the other variables in $\mathbf{x}^*_{-j}$ are set to their median for numeric variables and to the most common category for factors. All of these options can be modified by passing additional arguments to visreg. For example, contrast plots can be obtained with the `type` argument; the following code produces Figure 2.

```
visreg(fit, "Wind", type="contrast")
visreg(fit, "Wind", type="conditional")
```

The second argument specifies the explanatory variable to be visualized; note that the right plot in Figure 2 is the same as the middle plot in Figure 1.

In addition to continuous explanatory variables, **visreg** also allows the easy visualization of differences between the levels of categorical variables (factors). The following block of code creates a

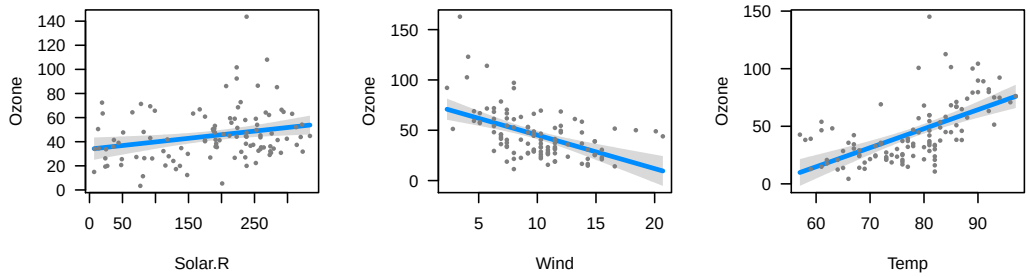


Figure 1: Basic output of **visreg** for an additive linear model: conditional plots for each explanatory variable.

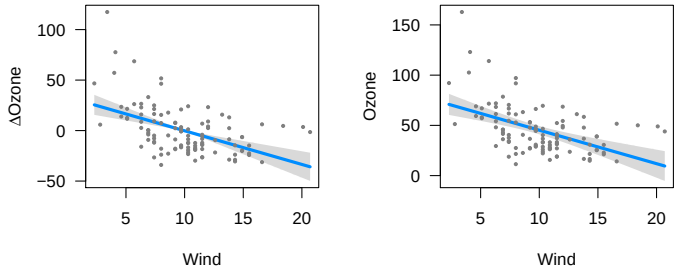


Figure 2: The estimated relationship between wind and ozone concentration in the same model, as illustrated by two different types of plots. Left: Contrast plot. Right: Conditional plot.

factor called `Heat` by discretizing `Temp`, and then visualizes its relationship with `Ozone`, producing the plot in Figure 3.

```
airquality$Heat <- cut(airquality$Temp, 3, labels=c("Cool", "Mild", "Hot"))
fit.heat <- lm(Ozone ~ Solar.R + Wind + Heat, data=airquality)
visreg(fit.heat, "Heat", type="contrast")
visreg(fit.heat, "Heat", type="conditional")
```

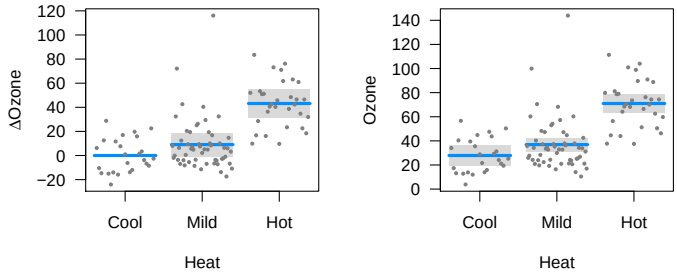


Figure 3: Visualization of a regression function involving a categorical explanatory variable. Left: Contrast plot. Right: Conditional plot.

Again, note that the confidence interval for the contrast plot has zero width for the reference category. There is no uncertainty about how the expected value of ozone will change if we remain at the same level of `Heat`; it is zero by definition. On the other hand, the width of the confidence interval for `Mild` heat is wider for the contrast plot than it is for the conditional plot. There is less uncertainty about the expected value of ozone on a mild day than there is about the difference in expected values between mild and cool days.

Transformations

Often in modeling, we introduce transformations of explanatory variables, transformations of the response variable, or both. The **visreg** package automatically handles these transformations when visualizing the regression model.

Linear models assume a linear relationship between the explanatory variables and the outcome. A common way of extending the linear model is to introduce transformations of the original explanatory variables. For example, to allow the effect of wind on ozone to be nonlinear, we may introduce a quadratic term for wind into the model:

```
fit1 <- lm(Ozone ~ Solar.R + Wind + I(Wind^2) + Temp, data=airquality)
```

Transformations of the response are also common. For example, ozone levels must be positive. However, as Figure 1 illustrates, a standard linear model allows the estimated relationship and its confidence band to fall below 0. One way of remedying this is to model the log of ozone concentrations instead of the ozone concentrations directly:

```
fit2 <- lm(log(Ozone) ~ Solar.R + Wind + Temp, data=airquality)
```

And of course, these elements may be combined:

```
fit3 <- lm(log(Ozone) ~ Solar.R + Wind + I(Wind^2) + Temp, data=airquality)
```

Visualization is particularly important in these models, as it is difficult to determine the exact nature of the relationship between explanatory variable and response simply by looking at the regression coefficients when that relationship is nonlinear. The **visreg** package provides a convenient way to view such relationships. Transformations involving explanatory variables are handled automatically, while transformations involving the response require the user to provide the inverse transformation. The following code produces Figure 4.

```
visreg(fit1, "Wind")
visreg(fit2, "Wind", trans=exp, ylab="Ozone", partial=TRUE)
visreg(fit3, "Wind", trans=exp, ylab="Ozone", partial=TRUE)
```

By default, **visreg** suppresses partial residuals when `trans` is specified, as this can provide a distorted view of outliers (a mild outlier can become an extreme outlier once a transformation has been applied, and vice versa), but we include them here by explicitly specifying `partial=TRUE`.

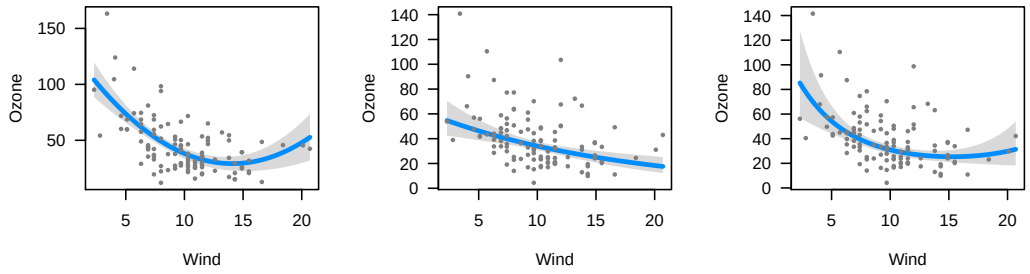


Figure 4: Plots of the modeled relationship between wind and ozone concentration, as estimated by different models. Left: The model contains a transformation of wind. Middle: The model contains a transformation of ozone concentration. Right: The model contains transformations of both wind and ozone.

Conditioning

As noted in "Basic framework", the default behavior of **visreg** when constructing a conditional plot is to fill in $\mathbf{x}^*_{-j}$ with the median for continuous variables and the most common category for categorical variables. This behavior can be modified using the `cond` argument. Note that this has no bearing on contrast plots in additive models, which do not require a full specification of $\mathbf{x}^*_{-j}$.

The `cond` argument must be provided as a named list. Each element of that list specifies the value for an element of $\mathbf{x}^*_{-j}$; any elements left unspecified are filled in with the median/most common category. We revisit our initial model from "Basic framework" with this code, which produces Figure 5.

```
visreg(fit, "Wind", cond=list(Temp=50))
visreg(fit, "Wind")
visreg(fit, "Wind", cond=list(Temp=100))
```

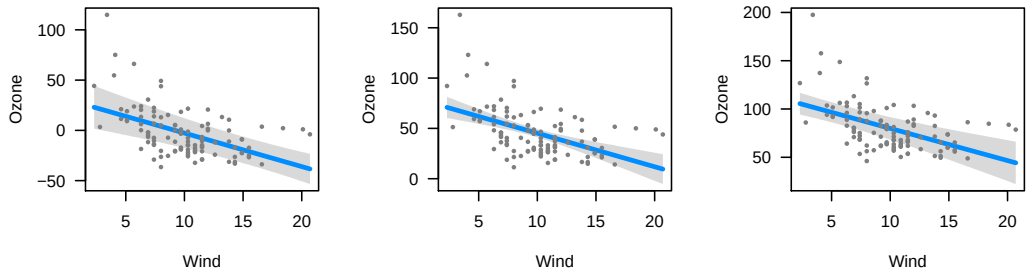


Figure 5: Estimated relationship between wind and ozone concentration, conditioning on different values of temperature. Left: Temperature=50 °F. Middle: The median temperature, 79 °F (default). Right: Temperature=100 °F.

We make several observations concerning Figure 5: i) The values on the vertical axis differ; as we condition on higher temperatures, the expected ozone concentration goes up since the regression coefficient for temperature is positive. ii) The slope of the line, the distance from the line to each residual, and the range of the residuals is the same in all three plots; conditioning on different values of temperature merely adds a constant to the regression line and the partial residuals. iii) The width of the confidence band *does* change, however: the data set has few observations at very high and very low temperatures, so the standard errors are much larger for the plots on the right and left than for the plot in the middle. iv) The shape of the confidence band also changes. In the middle plot, the confidence band is narrowest in the middle and wider at the ends. In the left plot (conditioning on low temperature), however, the confidence band is narrowest for high wind levels. This arises because there is a negative correlation between wind and temperature ($\hat{\rho} = -0.46$), and thus, more cold windy days in the data set than cold calm days. The opposite phenomenon happens in the right plot, where the relative absence of hot windy days causes the confidence band to be wider for high winds than for low winds.

Recall that this model had three explanatory variables; in the above example, **visreg** calculated the conditional response by filling in solar radiation with its median value, as it was not specified otherwise in the cond argument.

Linear models with interactions

Visualization is also very important for models with interactions – as with polynomial terms, in these models the relationship between an explanatory variable and the response depends on multiple regression coefficients, and a model's fit is more readily understood with a visual representation than by looking at a table of regression coefficients.

For models with interactions, we must simultaneously visualize the effect of two explanatory variables. The **visreg** package offers two methods for doing this: cross-sectional plots, which plot one-dimensional relationships between the response and one predictor for several values of another predictor, and surface plots, which attempt to provide a picture of the regression surface over both dimensions simultaneously.

Cross-sectional plots

To begin, let's fit a model that involves an interaction between a continuous term and a categorical term, using our derived variable Heat from "Basic framework":

```
fit <- lm(Ozone ~ Solar.R + Wind * Heat, data=airquality)
```

The **visreg** package creates cross-sectional plots using, by default, the **lattice** package (Sarkar, 2008). To request a cross-sectional plot, the user specifies a by variable, as in the following code which produces Figure 6.

```
visreg(fit, "Wind", by="Heat")
```

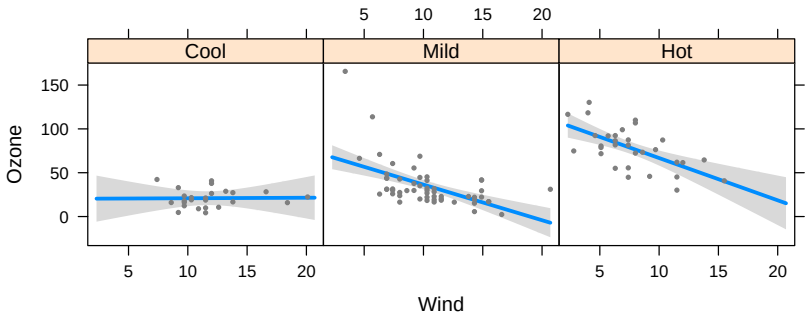


Figure 6: Cross-sectional plots depicting the fit of a model with an interaction between a continuous term (Wind) and a categorical term (Heat), with the continuous term on the horizontal axis.

Alternatively, one can use **ggplot2** (Wickham, 2009) as the plotting engine using the option `gg=TRUE`, as in the following code which produces Figure 7.

```
visreg(fit, "Wind", by="Heat", gg=TRUE)
```

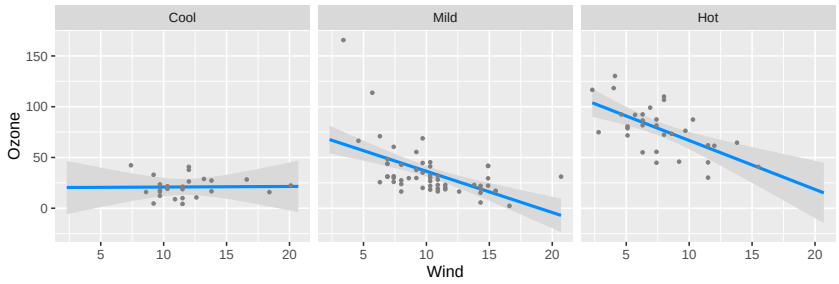


Figure 7: Same as Figure 6, but using **ggplot2** as the plotting engine.

The cross-sectional plots in either Figure 6 or 7 allow us to see that the relationship between wind and ozone concentration appears to become more pronounced depending on how hot the day is. On cool days, wind has no effect on ozone concentration. Wind has a moderate effect on ozone concentrations on mild days, and an even larger effect on hot days.

Note that **visreg** handles the partial residuals properly – the partial residuals for observations collected on cool days appear only in the left panel, and so on. As with the earlier plots, this ensures that the least squares line drawn through the residuals on the plot will yield the same slope as that estimated by the full model fit. Furthermore, this allows us to see potentially influential observations like the one in the middle panel, which has very low wind and very high ozone concentration. Finally, the proper handling of partial residuals also allows us to observe the lack of hot windy days and cool days with no wind that we commented on in "Conditioning". Note that the confidence intervals in these regions are comparatively wide.

Alternatively, we may wish to overlay these cross-sections. This allows for a more direct comparison between the different regression lines, although it often becomes difficult to include partial residuals and confidence bands without crowding the figure. The visreg package allows an `overlay` option for creating these plots:

```
visreg(fit, "Wind", by="Heat", overlay=TRUE, partial=FALSE)
```

The above code produces Figure 8, where the plotting of partial residuals has been turned off for the sake of clarity (similarly, `band=FALSE` can be specified to turn off the confidence bands). If `partial=TRUE`, the partial residuals are colored according to the existing scheme.

The above examples featured a continuous variable along the horizontal axis and a categorical variable as the by variable. However, **visreg** allows each of these variables to be either continuous or categorical. For example, let us try plotting the same model, but reversing the roles of Heat and Wind (Figure 9).

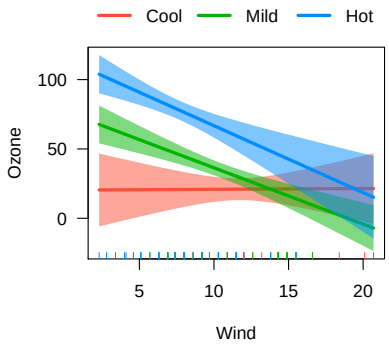


Figure 8: Cross-sectional plot depicting the fit of a model with an interaction between a continuous term (Wind) and a categorical term (Heat), where the regression lines for each category are overlaid.

```
visreg(fit, "Heat", by="Wind")
```

The model is the same, but the emphasis of the plot is now on heat instead of wind. Figure 9 illustrates that heat has a pronounced effect on ozone concentration when the day is not windy, but a relatively insignificant effect on ozone for windy days.

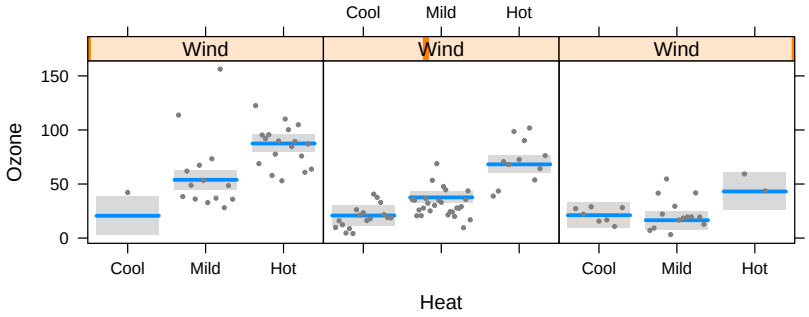


Figure 9: Cross-sectional plots depicting the fit of a model with an interaction between a continuous term (Wind) and a categorical term (Heat), with the categorical term on the horizontal axis.

In contrast to Figure 6, where it was natural to construct a panel for each level of the categorical variable, Figure 8 requires arbitrary decisions concerning how many cross-sections to take, and where to place them. The default behavior of **visreg** is to take cross-sections at the 10th, 50th, and 90th percentiles of the by variable, although both the number of points and their location can be modified using the `breaks` option. Again, each residual appears only once, in the panel it is closest to. However, the least squares estimates are no longer equivalent to drawing a line through the partial residuals due to the continuous manner in which information is pooled across the panels.

We have been focusing here on conditional plots, but contrast plots can be made as well by specifying `type="contrast"`. It is worth noting that for a model containing an interaction, a basic call to **visreg** (*i.e.*, without a `by` argument) amounts to plotting a main effect in the presence of an interaction. Because this has the potential to be misleading, **visreg** by default prints a message warning the user of this and reminding him or her of the levels of the other variables at which the plot is constructed. For example, since `"Mild"` is the most common level of `Heat`, `visreg(fit,"Wind")` will produce the middle panel of Figure 6. The left and right panels, respectively, could be produced by passing `Heat="Cool"` and `Heat="Hot"` to the `cond` argument.

Surface plots

Another approach to visualizing models with interactions is plotting the regression surface using contour or perspective plots. Suppose we fit a complicated model involving a multiplicative interaction between two-degree-of-freedom natural spline terms for wind and temperature (the function `ns` is from the **splines** () package):

```
fit <- lm(Ozone ~ Solar.R +ns(Wind, df=2)*ns(Temp, df=2), data=airquality)
```

Putting aside the question of whether or not this is a good model for analyzing these data, our purpose here is to show that it is difficult to grasp the fit of the model by looking at the regression coefficients directly, but easy to do so using **visreg**. In addition to the tools for creating cross-sectional plots described in the "Cross-sectional plots", the **visreg** package provides the function `visreg2d`, which can be used to produce two-dimensional contour and perspective plots. The following code produces Figure 10:

```
visreg2d(fit, "Wind", "Temp", plot.type="image")
visreg2d(fit, "Wind", "Temp", plot.type="persp")
```

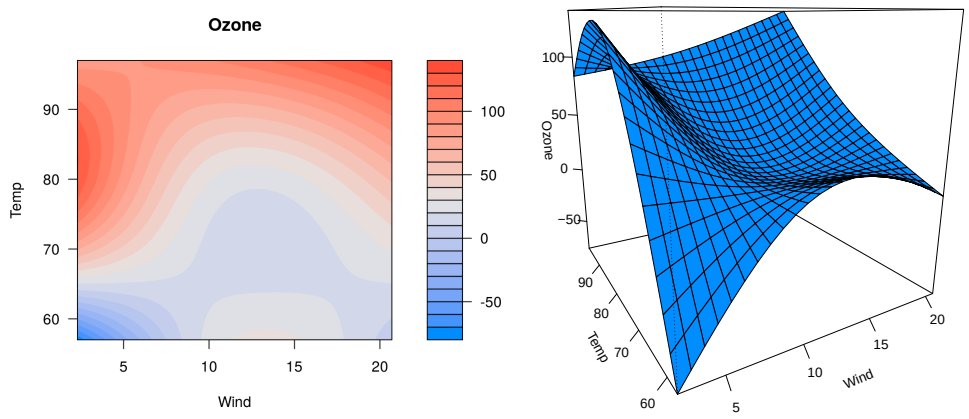


Figure 10: Representations of the regression surface as a function of wind and temperature. Left: Filled contour plot. Right: Perspective plot.

The advantage of these kinds of plots compared with those in "Cross-sectional plots" are that they allow us to visualize the effect of simultaneously varying two factors. The disadvantage is that there is no convenient way of superimposing either residuals or confidence intervals. These plots are most useful when both variables are continuous, as one is not forced to take cross-sections over a continuous variable. The `visreg2d` function still functions correctly when one or both of its arguments is a categorical variable, although in our opinion, the cross-section plots of "Cross-sectional plots" are more useful in these settings.

In addition to the static perspective plot presented above, `visreg2d` can also create interactive perspective plots using the **rgl** package (Adler and Murdoch, 2011), which allow the user to rotate, tilt, and spin the regression surface. This makes it considerably easier to comprehend its three-dimensional shape. Such plots can be constructed with the code:

```
visreg2d(fit, x="Wind", y="Temp", plot.type="rgl")
```

Visualization of higher-order interactions, such as three-way or four-way interactions, becomes increasingly difficult. To some extent, **visreg** facilitates visualization of such models through the use of the `cond` argument. For example, code such as the following could be used to visualize a three-way interaction:

```
fit <- lm(Ozone ~ Solar.R * Wind * Temp, data=airquality)
visreg2d(fit, "Wind", "Temp", cond=list(Solar.R=100))
visreg2d(fit, "Wind", "Temp", cond=list(Solar.R=300))
```

Other models

As mentioned at the outset, the goal in creating the **visreg** package was to implement visualization in an object-oriented manner, so that it works with as many classes of models from different functions and packages as possible. All that it requires is functioning `model.frame` and `predict` methods for the fitted model object (plotting of partial residuals requires a `residuals` method as well). Thus, the **visreg** package and all its options work not only with linear model objects produced by `lm`, but with generalized linear models produced by `glm`, proportional hazards models produced by `coxph` (Therneau, 2012), robust linear models produced by `rlm` (from **MASS**: Venables and Ripley, 2002),

negative binomial models produced by `glm.nb` (from **MASS**), generalized additive models produced by `gam` (from **mgcv**: Wood, 2012), local regression models produced by `loess` and **locfit** (Loader, 2010), and many more. Indeed, the type of object does not even need to be part of an R package; user-defined model classes can also be visualized with **visreg**, provided that they are compatible with `model.frame` and `predict`. In this section, we briefly illustrate the use of **visreg** with some of the above types of models.

Generalized linear models

We begin with a logistic regression model applied to a study investigating risk factors associated with low birth weight (Hosmer and Lemeshow, 2000). The following code produces Figure 11.

```
data("birthwt", package="MASS")
fit <- glm(low ~ age + race + smoke + lwt, data=birthwt, family="binomial")
visreg(fit, "lwt", xlab="Mother's weight", ylab="Log odds (low birthweight)")
visreg(fit, "lwt", scale="response", rug=2, xlab="Mother's weight",
       ylab="P(low birthweight)")
```

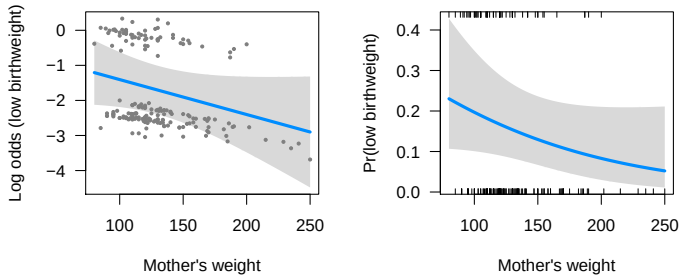


Figure 11: Visualization of a logistic regression model. Left: Log odds scale. Right: Probability scale.

On the left side of Figure 11, the model is plotted on the scale of the linear predictor (the default scale in **visreg**), where the model is indeed linear. The confidence intervals in the figure are Wald confidence intervals based on standard errors returned by `predict.glm`. The partial residuals are calculated based on Equation 2, with $\mathbf{r}$ the deviance residuals (the default residuals returned by `residuals.glm`). The plot on the right is simply a transformed version of the plot on the left, where an inverse logistic transformation has been applied to the regression line and confidence bands (this is handled automatically by the `scale="response"` option).

Note that for the plot on the right, we have opted to plot a rug as opposed to the partial residuals. The **visreg** package provides two types of rug annotations. With `rug=TRUE` or `rug=1`, a standard rug along the bottom of the plot is provided. With `rug=2`, separate rugs are drawn on the top for observations with positive residuals and on the bottom for observations with negative residuals (for logistic regression, this corresponds to $Y = 1$ and $Y = 0$, respectively).

In practice, we have found plots like those on the left useful for visualizing the model fit and observing potential departures from model assumptions such as outliers and influential points, and plots like those on the right very useful for communicating modeling results to non-statisticians.

Other regression models

Here, we provide a brief demonstration applying **visreg** to some other types of models (note that these are models for which the **effects** package is incompatible): a proportional hazards model, a robust regression model, and a local regression model. The left side of Figure 12 presents a visualization of the following proportional hazards model:

```
require("survival")
fit <- coxph(Surv(futime, fustat) ~ age + rx, data=ovarian)
visreg(fit, "age", ylab="log(Hazard ratio)")
```

Note that in proportional hazards models, baseline hazard functions are not explicitly estimated, and therefore the meaning behind a conditional plot is questionable. For this reason, contrast plots are (arguably) more appropriate. A similar phenomenon occurs with logistic regression applied to

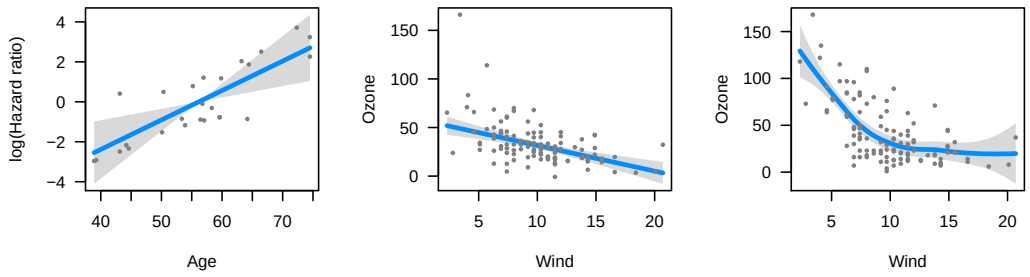


Figure 12: Visualizations of proportional hazards (left), robust regression (middle), and loess (right) models.

case-control studies, in which an intercept is estimated, but is the estimate is biased by the study design.

The middle of Figure 12 presents a visualization of the following robust regression model (using `rlm` from the **MASS** package):

```
fit <- rlm(Ozone ~ Solar.R + Wind * Heat, data=airquality)
visreg(fit, "Wind", ylab="Ozone")
```

Note that the design matrix for the robust regression model is the same as that from "Cross-sectional plots", and that the plot in the middle of Figure 12 is analogous to the middle panel from Figure 6. Note, however, that the robust regression model produces a different fit, due in part to the reduced impact of the potential outlier mentioned in "Cross-sectional plots". Specifically, the fit produced by the robust regression model is flatter and does not predict negative ozone concentrations for high wind levels as the linear regression model does.

Finally, we apply **visreg** to a local regression model fit with `loess`, producing a much more useful visualization of the model than the default `plot` method for `loess`. This plot appears on the right side of Figure 12.

```
fit <- loess(Ozone ~ Wind, airquality)
visreg(fit, "Wind", ylab="Ozone")
```

All of the features and options we mentioned earlier; in particular cross-section and surface plots work in the same way for nonlinear models as they do for linear models.

Computationally, the extension of **visreg** to nonlinear models is straightforward due to its object-oriented implementation, but it is worth making some comments about partial residuals for nonlinear models. In particular, it is no longer the case that the regression line through the partial residuals produces a line with the same slope as that produced by the model. Viewing nonlinear models as reweighted least squares models, the observations have different weights and these weights are not reflected in the partial residuals plotted by **visreg**. This phenomenon has been commented on by many authors, with a variety of proposals for alternative types of reweighted partial residuals that may be better at detecting outliers and influential observations (Pregibon, 1981; Landwehr et al., 1984; O'Hara Hines and Carter, 1993).

Non-regression models

Moving even further from linear models, **visreg** is also compatible with modeling frameworks that are not even regression-based, such as random forests and support vector machines. Such methods are often thought of as "black boxes", but **visreg** offers a convenient way to visualize the resulting fit and possibly gain some insight into the model. The following code fits each of the aforementioned models to the airquality data using the **randomForest** (Breiman et al., 2015) and **e1071** (Meyer et al., 2017) packages, and plots the resulting estimated association between ozone and temperature (Figure 13). Some of these packages do not automatically handle missing data, so we first create a complete-case data set `aq`:

```
aq <- na.omit(airquality)
fit1 <- randomForest(Ozone ~ Solar.R + Wind + Temp, data=aq)
fit2 <- svm(Ozone ~ Solar.R + Wind + Temp, data=aq)
visreg(fit1, "Temp", ylab="Ozone", ylim=c(0, 150))
visreg(fit2, "Temp", ylab="Ozone", ylim=c(0, 150))
```

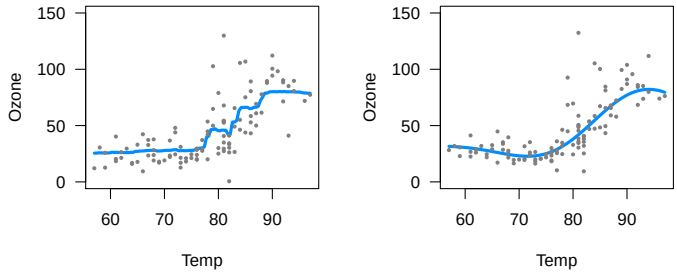


Figure 13: Left: Random forest. Right: Support vector machine.

Both of the results in Figure 13 appear reasonable with the default settings employed, although neither of these models is able to provide confidence bands for fitted values, so no shaded bands appear. A useful feature of plotting the model's predictions, however, is that it illustrates the effect of changing those settings. For example, consider the application of gradient boosting machines to this same data using the **gbm** package (Ridgeway, 2017). First, it is worth noting that the **gbm** package does not offer a `residuals` method. This would normally cause **visreg** to omit plotting the partial residuals. However, we can supply our own user-defined `residuals` method, which enables visreg to produce the plots in Figure 14.

```
residuals.gbm <- function(fit) fit$data$y - fit$fit
fit3 <- gbm(Ozone ~ Solar.R + Wind + Temp, data=aq)
fit4 <- gbm(Ozone ~ Solar.R + Wind + Temp, data=aq, n.trees=5000)
visreg(fit3, "Temp", ylab="Ozone", ylim=c(0, 150))
visreg(fit4, "Temp", ylab="Ozone", ylim=c(0, 150))
```

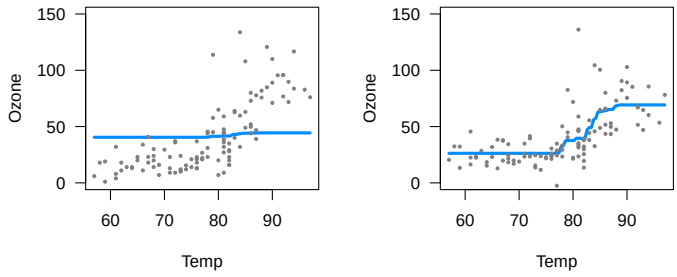


Figure 14: Visualizations of gradient boosting machine. Left: Default setting (100 trees) Right: 5,000 trees.

Note that the default settings for **gbm** do not produce a very good fit here. In particular, the default number of trees (100) is too low to capture the relationship between temperature and ozone. By increasing the number of trees (to 5,000), we obtain a much more reasonable fit.

Hierarchical and random effect models

The ability of **visreg** to visualizing mixed effect models is hindered by the fact that incorporating uncertainty about random effects into predictions is difficult from a frequentist perspective and most R packages for such models do not offer confidence intervals for such estimates. Nevertheless, **visreg** is still useful for visualizing the effects of fixed effects in such models using contrast plots, as well as plotting effects without confidence intervals.

As an illustration, we consider a study involving the protein content of cows' milk in the weeks following calving (Diggle et al., 2002). Consider the following random-intercept, random-slope model, fit using the **lme4** package (Bates et al., 2012), which also contains a fixed effect for the type of diet each cow was fed.

```
data(Milk, package="nlme")
fit <- lmer(protein ~ Diet + Time + (Time|Cow), Milk)
```

In the **lme4** package, the `predict` method does not return standard errors. This means that any conditional plots constructed by visreg will lack confidence intervals, like those in Figures 13 and 14. This is another example of a situation where a contrast plot is useful: by considering the effect of changing diet while other terms remain constant, the random effects drop out of the model and standard errors/confidence intervals are straightforward, as illustrated in Figure 15. The following code also illustrates how to change graphical options, as there is considerable overplotting of the partial residuals under the default settings.

```
visreg(fit, "Diet", type="contrast", ylab=expression(Delta*'Protein'),
       points.par=list(col="#55555540", cex=0.25))
```

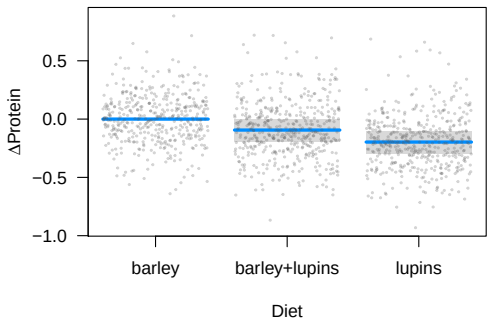


Figure 15: Contrast plot illustrating the fixed effect of diet in the `Milk` example.

The **visreg** package can also be used to plot random effects, although as mentioned earlier, the plots will not include intervals. Below, we provide code to plot the modeled relationship between protein content and time. Two aspects of the code are worth pointing out. First, note that according to the object-oriented design of **visreg**, the `predict` method supplied by **lme4** will used. It has its own option, `re.form`, to control how random effects are used in the prediction, and this must be passed through **visreg** accordingly. Second, for the sake of space we subset the plot to ten cows rather than all 79. This can be accomplished by returning, then subsetting, the raw **visreg** object prior to plotting. Returning the data frames, estimates, confidence intervals, and residuals used in the construction of its plots like this allows users to write their own extensions and modifications of **visreg** plots.

```
v <- visreg(fit, "Time", by="Cow", re.form=~(Time|Cow), plot=FALSE)
subCow <- sample(Milk$Cow, 10)
vv <- subset(v, Cow %in% subCow)
plot(vv, ylab="Protein", layout=c(10,1))
```

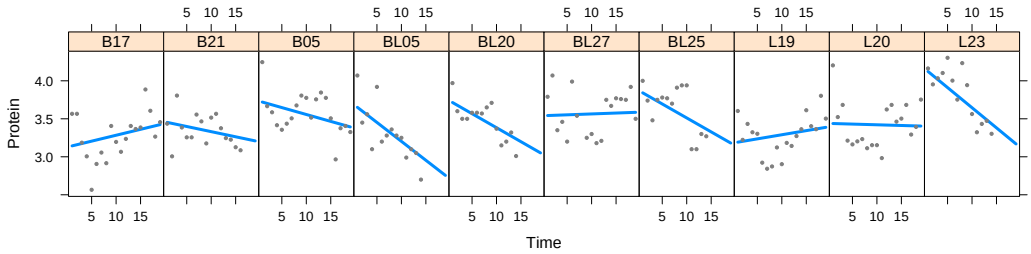


Figure 16: Subject-specific conditional plots for ten randomly chosen cows from the `Milk` example illustrating the change in protein content over time.

Conclusion

Partial residuals and how useful they are in detecting influential observations and departures from model assumptions depends on the model. Other types of plots, such as added variable plots

(Atkinson, 1985), are also helpful for visualizing regression models and their fit. We feel that the approach provided by **visreg** is reasonable and the best that can be expected from an object-oriented tool that can be applied generically to a wide variety of models, although we certainly acknowledge that other types of plots and visualizations may offer useful additional information for certain types of models.

The **visreg** package provides a very useful set of tools for simultaneously visualizing the estimated relationship between an explanatory variables and the outcome, the variability of that estimate, and the observations from which the estimates derive. These tools have a simple interface and are readily applied in an object-oriented manner to wide variety of models. We have found the development of this package to provide a convenient and versatile tool to assist with regression modeling, both for model exploration and for communicating modeling results.

More information about **visreg**, illustrating its various options with numerous examples can be found at http://pbreheny.github.io/visreg.

Bibliography

D. Adler and D. Murdoch. *Rgl: 3D Visualization Device System (OpenGL)*, 2011. R package version 0.92.798. [p65]

A. C. Atkinson. Regression diagnostics, transformations and constructed variables. *Journal of the Royal Statistical Society B*, 44:pp. 1–36, 1982. [p58]

A. C. Atkinson. *Plots, Transformations, and Regression*. Oxford University Press, 1985. [p70]

D. Bates, M. Maechler, and B. Bolker. *Lme4: Linear Mixed-Effects Models Using S4 Classes*, 2012. R package version 0.999999-0. [p68]

D. A. Belsley, E. Kuh, and R. E. Welsch. *Regression Diagnostics*. John Wiley & Sons, 1980. [p56]

L. Breiman, A. Cutler, A. Liaw, and M. Wiener. *randomForest: Breiman and Cutler's Random Forests for Classification and Regression*, 2015. R package version 4.6-12. [p67]

R. D. Cook. Exploring partial residual plots. *Technometrics*, 35:351–362, 1993. URL https://doi.org/10.2307/1270269. [p56]

R. D. Cook and S. Weisberg. *Residuals and Influence in Regression*. Chapman and Hall, 1982. [p56]

P. J. Diggle, P. J. Heagerty, K.-Y. Liang, and S. L. Zeger. *Analysis of Longitudinal Data*. Oxford University Press, 2002. [p68]

B. Efron. Better bootstrap confidence intervals. *Journal of the American Statistical Association*, 82:171–185, 1987. URL https://doi.org/10.2307/2289144. [p59]

M. Ezekiel. A method of handling curvilinear correlation for any number of variables. *Journal of the American Statistical Association*, 19:431–453, 1924. URL https://doi.org/10.2307/2281561. [p58]

J. Fox. Effect displays in R for generalised linear models. *Journal of Statistical Software*, 8:1–27, 2003. URL https://doi.org/10.18637/jss.v008.i15. [p57]

J. Fox and S. Weisberg. *An R Companion to Applied Regression*. Sage, Thousand Oaks CA, 2nd edition, 2011. [p57]

P. M. Grambsch and T. M. Therneau. Proportional hazards tests and diagnostics based on weighted residuals. *Biometrika*, 81:pp. 515–526, 1994. URL https://doi.org/10.1093/biomet/81.3.515. [p56]

F. Harrell. *Regression Modeling Strategies: With Applications to Linear Models, Logistic and Ordinal Regression, and Survival Analysis*. Springer-Verlag, 2015. [p57]

T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer-Verlag, 2009. [p57]

T. J. Hastie and R. J. Tibshirani. *Generalized Additive Models*. Chapman & Hall/CRC, 1990. [p56]

D. W. Hosmer and S. Lemeshow. *Applied Logistic Regression*. John Wiley & Sons, 2000. [p66]

P. E. Johnson. *Rockchalk: Regression Estimation and Presentation*, 2016. R package version 1.8.101. [p57]

M. Kutner, C. Nachtsheim, J. Neter, and W. Li. *Applied Linear Statistical Models*. McGraw-Hill, 2004. [p58]

J. M. Landwehr, D. Pregibon, and A. C. Shoemaker. Graphical methods for assessing logistic regression models. *Journal of the American Statistical Association*, 79:61–71, 1984. URL https://doi.org/10.2307/2288334. [p67]

W. A. Larsen and S. J. McCleary. The use of partial residual plots in regression analysis. *Technometrics*, 14:781–790, 1972. URL https://doi.org/10.2307/1267305. [p58]

C. Loader. *Locfit: Local Regression, Likelihood and Density Estimation.*, 2010. R package version 1.5-6. [p66]

A. Loy and H. Hofmann. Diagnostic tools for hierarchical linear models. *Wiley Interdisciplinary Reviews: Computational Statistics*, 5:48–61, 2013. URL https://doi.org/10.1002/wics.1238. [p56]

C. L. Mallows. Augmented partial residuals. *Technometrics*, 28:313–319, 1986. URL https://doi.org/10.2307/1268980. [p56]

D. Meyer, E. Dimitriadou, K. Hornik, A. Weingessel, and F. Leisch. *E1071: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien*, 2017. R package version 1.6-8. [p67]

R. J. O'Hara Hines and E. M. Carter. Improved added variable and partial residual plots for the detection of influential observations in generalized linear models. *Journal of the Royal Statistical Society C*, 42:pp. 3–20, 1993. URL https://doi.org/10.2307/2347405. [p67]

D. Pregibon. Logistic regression diagnostics. *The Annals of Statistics*, 9:705–724, 1981. URL https://doi.org/10.1214/aos/1176345513. [p56, 67]

G. Ridgeway. *Gbm: Generalized Boosted Regression Models*, 2017. R package version 2.1.3. [p68]

D. Sarkar. *Lattice: Multivariate Data Visualization with R*. Springer-Verlag, 2008. [p62]

T. Therneau. *A Package for Survival Analysis in S*, 2012. R package version 2.36-12. [p65]

W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S*. Springer-Verlag, 2002. [p65]

H. Wickham. *Ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag, 2009. [p63]

C. S. Withers and S. Nadarajah. Improved confidence regions based on edgeworth expansions. *Computational Statistics & Data Analysis*, 56:4366 – 4380, 2012. URL https://doi.org/10.1016/j.csda.2012.03.019. [p59]

F. S. Wood. The use of individual effects and residuals in fitting equations to data. *Technometrics*, 15:677–695, 1973. URL https://doi.org/10.2307/1267381. [p58]

S. Wood. *Mgcv: Mixed GAM Computation Vehicle with GCV/AIC/REML Smoothness Estimation*, 2012. R package version 1.7-17. [p66]

S. N. Wood. *Generalized Additive Models: An Introduction with R*. Chapman & Hall/CRC, 2006. [p56]

Patrick Breheny
Department of Biostatistics
University of Iowa
ORCiD: 0000-0002-0650-1119
patrick-breheny@uiowa.edu

Woodrow Burchett
Department of Statistics
University of Kentucky
woodrow.burchett@uky.edu

fourierin: An R package to compute Fourier integrals

by Guillermo Basulto-Elias, Alicia Carriquiry, Kris De Brabanter and Daniel J. Nordman

Abstract We present the R package **fourierin** (Basulto-Elias, 2017) for evaluating functions defined as Fourier-type integrals over a collection of argument values. The integrals are finitely supported with integrands involving continuous functions of one or two variables. As an important application, such Fourier integrals arise in so-called "inversion formulas", where one seeks to evaluate a probability density at a series of points from a given characteristic function (or vice versa) through Fourier transforms. This paper intends to fill a gap in current R software, where tools for repeated evaluation of functions as Fourier integrals are not directly available. We implement two approaches for such computations with numerical integration. In particular, if the argument collection for evaluation corresponds to a regular grid, then an algorithm from Inverarity (2002) may be employed based on a fast Fourier transform, which creates significant improvements in the speed over a second approach to numerical Fourier integration (where the latter also applies to cases where the points for evaluation are not on a grid). We illustrate the package with the computation of probability densities and characteristic functions through Fourier integrals/transforms, for both univariate and bivariate examples.

Introduction

Continuous Fourier transforms commonly appear in several subject areas, such as physics and statistics. In probability theory, for example, continuous Fourier transforms are related to the characteristic function of a distribution and play an important role in evaluating probability densities from characteristic functions (and vice versa) through inversion formulas (cf. Athreya and Lahiri (2006)). Similar Fourier-type integrations are also commonly required in statistical methods for density estimation, such as kernel deconvolution (cf. Meister (2009)).

At issue, the Fourier integrals of interest often cannot be solved in elementary terms and typically require numerical approximations. As a compounding issue, the oscillating nature of the integrands involved can cause numerical integration recipes to fail without careful consideration. However, Bailey and Swarztrauber (1994) present a mid-point integration rule in terms of appropriate discrete Fourier transforms, which can be efficiently computed using the Fast Fourier Transform (FFT). Inverarity (2002) extended this characterization to the multivariate integral case. These works consequently offer targeted approaches for numerically approximating types of Fourier integrals of interest (e.g., in the context of characteristic or density functions).

Because R is one of the most popular programming languages among statisticians, it seems worthwhile to have general tools available for computing such Fourier integrals in this software platform. However, we have not found any R package that specifically performs this type of integral in general, though this integration does intrinsically occur in some statistical procedures. See Stirnemann et al. (2012) for an application in kernel deconvolution where univariate Fourier integrals are required. Furthermore, beyond the integral form, the capacity to handle repeated calls for such integrals is another important consideration. This need arises when computing a function, that is itself defined by a Fourier integral, over a series of points. Note that this exact aspect occurs when determining a density function from characteristic function (or vice versa), so that the ability to efficiently compute Fourier integrals over a collection of arguments is crucial.

The intent of the package **fourierin** explained here is to help in computing such Fourier-type integrals within R. The main function of the package serves to calculate Fourier integrals over a range of potential arguments for evaluation and is also easily adaptable to several definitions of the continuous Fourier transform and its inverse (cf. Inverarity (2002)). (That is, the definition of a continuous Fourier transform may change slightly from one context to another, often up to normalizing constants, so that it becomes reasonable to provide a function that can accommodate any given definition through scaling adjustments.) If the points for evaluating Fourier integrals are specified on regular grid, then the package allows use of the FFT for particularly fast numerical integration. However, the package also allow the user to evaluate such integrals at arbitrary collections of points that need not constitute a regular grid (though, in this case, the FFT cannot be used and computations naturally become slower). The latter can be handy in some situations; for example, evaluations at zero can provide moments of a random variable when computing derivatives of a characteristic function from the probability density. The heavy computations in **fourierin** are performed in C++ via the **RcppArmadillo** package (cf. Eddelbuettel and Sanderson (2014)).

The rest of the paper has four sections. We first describe the Fourier integral for evaluation and its numerical approximation in "Fourier Integrals and Fast Fourier Transform (FFT)." We then illustrate how package **fourierin** may be used in univariate and bivariate cases of Fourier integration. In "Speed Comparison," we demonstrate the two approaches (FFT-based or not) for computing Fourier integrals, both in one and two dimensions and at several grid sizes. We provide evidence that substantial time savings occur when using the FFT-based method for evaluation at points on a grid. Finally, in "Summary," we present conclusions and upcoming extensions to the R package.

Fourier integrals and fast Fourier transform

For $\boldsymbol{w} = (w_1, \ldots, w_n), \boldsymbol{t} = (t_1, \ldots, t_n) \in \mathbb{R}^n$, define the vector dot product $\langle \boldsymbol{w}, \boldsymbol{t} \rangle = w_1 t_1 + \cdots + w_n t_n$ and recall the complex exponential function $\exp\{\imath x\} = \cos(x) + \imath \sin(x)$, $x \in \mathbb{R}$, where $\imath = \sqrt{-1}$.

This package aims to compute Fourier integrals at several points simultaneously, which namely involves computation of the integral

$$\left[\frac{|s|}{(2\pi)^{1-r}}\right]^{n/2} \int_{a_1}^{b_1} \int_{a_2}^{b_2} \cdots \int_{a_n}^{b_n} f(\boldsymbol{t}) \exp\{\imath s \langle \boldsymbol{w}, \boldsymbol{t} \rangle\} \mathrm{d}\boldsymbol{t}, \tag{1}$$

at more than one potential argument $\boldsymbol{w} \in \mathbb{R}^n$, where f is a generic continuous n-variate function that takes real or complex values, for $n \in \{1, 2\}$, and the above limits of integration are defined by real values $a_j < b_j$ for $j = 1, \ldots, n$. Note that s and r in (1) denote real-valued constants, which are separately included to permit some flexibility in the definition of continuous Fourier transforms to be used (e.g., $s = 1$, $r = 1$). Hence, (1) represents a function of $\boldsymbol{w} \in \mathbb{R}^n$, defined by a Fourier integral, where the intention is to evaluate (1) over a discrete collection of $\boldsymbol{w}$-values, often defined by a grid in $\mathbb{R}^n$. For example, if $[c_1, d_1) \times \cdots \times [c_n, d_n) \subset \mathbb{R}^n$ denotes a rectangular region specified by some real constants $c_j < d_j$, $j = 1, \ldots, n$, one may consider evaluating (1) at points $\boldsymbol{w}$ lying on a regular grid of size $m_1 \times m_2 \times \cdots \times m_n$ within $[c_1, d_1) \times \cdots \times [c_n, d_n)$, say, at points $\boldsymbol{w}^{(j_1,\ldots,j_n)} = (w_{j_1}, \ldots, w_{j_n})$ for $w_{j_k} = c_k + j_k(d_k - c_k)/m_k$ with $j_k \in \{0, 1, \ldots, m_k - 1\}$, $k = 1, \ldots, n$ (where m_k denotes the number of grid points in each coordinate dimension). Argument points on a grid are especially effective for fast approximations of integrals (as in 1), as we discuss in the following.

At given argument $\boldsymbol{w} \in \mathbb{R}^n$, we numerically approximate the integral (1) with a discrete sum using the mid-point rule, whereby the approximation of the j-th slice of the multiple integral involves l_j partitioning rectangles (or equi-spaced subintervals) for $j = 1, \ldots, n$ and $n \in \{1, 2\}$; that is, for $l_1, \ldots, l_n$ representing a selection of the numbers of approximating nodes to be used in the coordinates of integration (i.e., a resolution size), the integral (1) is approximated as

$$\left(\prod_{j=1}^{n} \frac{b_j - a_j}{l_j}\right) \cdot \left[\frac{|s|}{(2\pi)^{1-r}}\right]^{n/2} \sum_{i_1=0}^{l_1-1} \sum_{i_2=0}^{l_2-1} \cdots \sum_{i_n=0}^{l_n-1} f(\boldsymbol{t}^{(i_1,\ldots,i_n)}) \exp\{\imath s \langle \boldsymbol{w}, \boldsymbol{t}^{(i_1,\ldots,i_n)} \rangle\}, \tag{2}$$

with nodes $\boldsymbol{t}^{(i_1,\ldots,i_n)} = (t_{i_1}, \ldots, t_{i_n})$ defined by coordinate midpoints $t_{i_k} = a_k + (2i_k + 1)/2 \cdot (b_k - a_k)/l_k$ for $i_k \in \{0, 1, \ldots, l_k - 1\}$ and $k = 1, \ldots, n$. Note that a large grid size $l_1 \times \cdots \times l_n$ results in higher resolution for the integral approximation (2), but at a cost of increased computational effort. On the other hand, observe that when a regular grid is used, the upper evaluation limits, $d_1, \ldots, d_n$ are not included in such grid, however, the higher the resolution, the closer we get to these bounds.

To reiterate, the goal is then to evaluate the Fourier integral (1) over some set of argument points $\boldsymbol{w} \in \mathbb{R}^n$ by employing the midpoint approximation (2), where the latter involves a $l_1 \times \cdots \times l_n$ resolution grid (of midpoint nodes) for $n \in \{1, 2\}$. It turns out that when the argument points $\boldsymbol{w}$ fall on a $m_1 \times \cdots \times m_n$-sized regular grid and this grid size matches the size of the approximating node grid from (2), namely $l_j = m_j$ for each dimension $j = 1, \ldots, n$, then the sum (2) may be written in terms of certain discrete Fourier transforms and inverses of discrete Fourier transforms that can be conveniently computed with a FFT operation. Details of this derivation can be found in Inverarity (2002). It is well known that using FFT greatly reduces the computational complexity of discrete Fourier transforms from $O(m^2)$ to $O(m \log m)$ in the univariate case, where m is the resolution or grid size. The complexity of computing the multivariate discrete Fourier transform of an n-dimensional array using the FFT is $O(M \log M)$, where $M = m_1 \cdots m_n$ and m_j is the grid/resolution size in the j-th coordinate direction, $j = 1, \ldots, n$.

The R package **fourierin** can take advantage of such FFT representations for the fast computation of Fourier integrals evaluated on a regular grid. The package can also be used to evaluate Fourier integrals at arbitrary discrete sets of points. The latter becomes important when one wishes to evaluate the a continuous Fourier transform at only a few specific points (that may not necessarily constitute a regular grid). We later compare evaluation time of Fourier integrals on a regular grid, both using the

FFT and without using it in **fourierin**.

Examples

In this section we present examples to illustrate use of the **fourierin** package. We begin with a univariate example which considers how to compute continuous Fourier transforms evaluated on a regular grid (therefore using the FFT operation) as well as how the computations proceed at three specified points not on a regular grid (where the FFT is not be used). The second example considers a two dimensional, or bivariate, case of Fourier integration.

The code that follows shows how the package can be used in univariate cases. The example we consider is to recover a χ^2 density f with five degrees of freedom from its characteristic function ϕ, where the underlying functions are given by

$$f(x) = \frac{1}{2^{5/2}\Gamma\left(\frac{5}{2}\right)} x^{\frac{5}{2}-1} e^{-\frac{x}{2}} \qquad \text{and} \qquad \phi(t) = (1-2\imath t)^{-5/2}, \tag{3}$$

for all $x > 0$ and $t \in \mathbb{R}$. We also show how to use the package on non-regular grids. Specifically, we generate sample of three points from a χ^2 distribution with five degrees of freedom and evaluate the density in Formula 3 at these three points where the density has been computed using the Fourier inversion formula approximated at four different resolutions. Results are presented in Table 1.

For illustration, the limits of integration are set from -10 to 10 and we compare several resolutions (64, 256 or 512) or grid node sizes for numerically performing integration (cf. (2)), recalling that the higher the resolution, the better the integral approximation. To evaluate the integrals at argument points on a regular grid, we choose $[-3, 20]$ as an interval for specifying a collection of equi-spaced points, where the number of such points equals the resolution specified (as needed when using FFT).

```
## ------------------------------------------------------------------
## Univariate example
## ------------------------------------------------------------------

## Load packages
library(fourierin)
library(dplyr)
library(purrr)
library(ggplot2)

## Set functions
df <- 5
cf <- function(t) (1 - 2i*t)^(-df/2)
dens <- function(x) dchisq(x, df)


## Set resolutions
resolutions <- 2^(6:8)

## Compute integral given the resoltion
recover_f <- function(resol){
    ## Get grid and density values
    out <- fourierin(f = cf, lower_int = -10, upper_int = 10,
                     lower_eval = -3, upper_eval = 20,
                     const_adj = -1, freq_adj = -1,
                     resolution = resol)
    ## Return in dataframe format
    out %>%
        as_data_frame() %>%
        transmute(
            x = w,
            values = Re(values),
            resolution = resol)
}

## Density approximations
vals <- map_df(resolutions, recover_f)
```

```
## True values
true <- data_frame(x = seq(min(vals$x), max(vals$x), length = 150),
                   values = dens(x))

univ_plot <-
    vals %>%
    mutate(resolution = as.character(resolution),
           resolution = gsub("64", "064", resolution)) %>%
    ggplot(aes(x, values)) +
    geom_line(aes(color = resolution)) +
    geom_line(data = true, aes(color = "true values"))

univ_plot

## Evaluate in a nonregular grid
set.seed(666)
new_grid <- rchisq(n = 3, df = df)
resolutions <- 2^(6:9)

fourierin(f = cf, lower_int = -10, upper_int = 10,
          eval_grid = new_grid,
              const_adj = -1, freq_adj = -1,
          resolution = 128) %>%
    c () %>% Re() %>%
    data_frame(x = new_grid, fx = .)


## Function that evaluates the log-density on new_grid at different
## resolutions (i.e., number of points to approximate the integral in
## the Fourier inversion formula).
approximated_fx <- function (resol) {
    fourierin(f = cf, lower_int = -10, upper_int = 12,
              eval_grid = new_grid,
              const_adj = -1, freq_adj = -1,
              resolution = resol) %>%
        c() %>% Re() %>%
        {data_frame(x = new_grid,
                    fx = dens(new_grid),
                    diffs = abs(. - fx),
                    resolution = resol)}
}

## Generate table
tab <-
    map_df(resolutions, approximated_fx) %>%
    arrange(x) %>%
    mutate(diffs = round(diffs, 7))  %>%
    rename('f(x)' = fx,
           'absolute difference' = diffs)

tab
```

Observe that the first call of the `fourierin` function above has the default argument `use_fft = TRUE`. Therefore, this computation uses the the FFT representation described in Inverarity (2002) for regular grids, which is substantially fast (Figure 5, as described later, provides timing comparisons without the FFT for contrast). Also note that, when a regular evaluation grid is used, `fourierin` returns a list with both the Fourier integral values and the evaluation grid. Figure 1 shows the resulting plot generated. A low resolution (64) for numerical integration has been included in order to observe differences between the true density and its recovered version using Fourier integrals.

At the bottom of the code above, we also show how `fourierin()` works when a non-regular "evaluation grid" is provided. Observe that, in this case, one directly specifies separate points for evaluation of the integral in addition to separately specifying a resolution level for integration. This aspect is unlike the evaluation case on a regular grid. Consequently, only the Fourier integral values

x	$f(x)$	absolute difference	resolution
3.0585883	0.1541368	0.0002072	64
		0.0000476	128
		0.0000472	256
		0.0000471	512
6.4144242	0.0874281	0.0000780	64
		0.0000155	128
		0.0000148	256
		0.0000147	512
11.7262677	0.0151776	0.0000097	64
		0.0000025	128
		0.0000022	256
		0.0000022	512

Table 1: Absolute differences of true density values at three random points and density values at these same three points obtained using the Fourier inversion formula approximated at different resolutions.

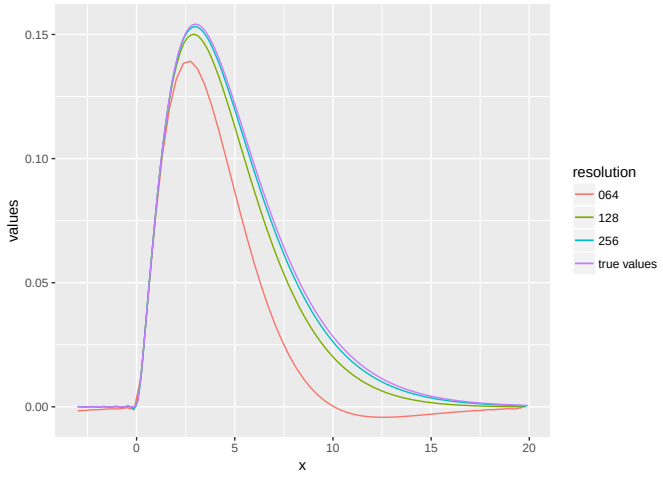


Figure 1: Example of `fourierin()` function for univariate function at resolution 64. Recovering a χ^2 density from its characteristic function. See Equation 3.

are returned, which is also unlike the regular grid case (where the evaluation grid is returned with corresponding integrals in a list). Note that the function f from (1), when having a real-valued argument, should be able to be evaluated at vectors in R.

In a second example, to illustrate how the `fourierin()` function works for bivariate functions, we use a bivariate normal density f and find its characteristic function ϕ. In particular, we have these underlying functions as

$$f(\boldsymbol{x}) = \frac{1}{\sqrt{2\pi|\Sigma|}} \exp\left[-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})'\Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right] \qquad \text{and} \qquad \phi(\boldsymbol{t}) = \exp\left(\imath \boldsymbol{x}'\boldsymbol{\mu} - \frac{1}{2}\boldsymbol{t}'\Sigma\boldsymbol{t}\right), \quad (4)$$

for all $\boldsymbol{t}, \boldsymbol{x} \in \mathbb{R}^2$, with $\boldsymbol{\mu} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$ and $\Sigma = \begin{bmatrix} 3 & -1 \\ -1 & 3 \end{bmatrix}$.

Below is the code for this bivariate case using a regular evaluation grid, where the output is a complex matrix whose components are Fourier integrals corresponding to the gridded set of bivariate arguments. As illustration, the limits of integration are set from $(-8, -6)$ to $(6, 8)$ (a square) and we consider a resolution 128, where the range $[-4, 4] \times [-4, 4]$ is also chosen to define a collection of evaluation points on a grid, where the number of such points again equals the resolution specified (i.e., for applying FFT).

```
## ------------------------------------------------------------------
## Bivariate example
## ------------------------------------------------------------------

## Load packages
```

```
library(fourierin)
library(tidyr)
library(dplyr)
library(purrr)
library(lattice)
library(ggplot2)

## Set functions to be tested with their corresponding parameters.
mu <- c(-1, 1)
sig <- matrix(c(3, -1, -1, 2), 2, 2)

## Multivariate normal density, x is n x d
f <- function(x) {
    ## Auxiliar values
    d <- ncol(x)
    z <- sweep(x, 2, mu, "-")
    ## Get numerator and denominator of normal density
    num <- exp(-0.5*rowSums(z * (z %*% solve(sig))))
    denom <- sqrt((2*pi)^d*det(sig))
    return(num/denom)
}

## Characteristic function, s is n x d
phi <- function (s) {
    complex(modulus = exp(-0.5*rowSums(s*(s %*% sig))),
            argument = s %*% mu)
}

## Evaluate characteristic function for a given resolution.
eval <- fourierin(f,
                  lower_int = c(-8, -6), upper_int = c(6, 8),
                  lower_eval = c(-4, -4), upper_eval = c(4, 4),
                  const_adj = 1, freq_adj = 1,
                  resolution = 2*c(64, 64),
                  use_fft = T)

## Evaluate true and approximated values of Fourier integral
dat <- eval %>%
    with(crossing(y = w2, x = w1) %>%
         mutate(approximated = c(values))) %>%
    mutate(true = phi(matrix(c(x, y), ncol = 2)),
           difference = approximated - true) %>%
    gather(value, z, -x, -y) %>%
    mutate(real = Re(z), imaginary = Im(z)) %>%
    select(-z) %>%
    gather(part, z, -x, -y, -value)


## Surface plot
wireframe(z ~ x*y | value*part, data = dat,
          scales =
              list(arrows=FALSE, cex= 0.45,
                   col = "black", font = 3, tck = 1),
          screen = list(z = 90, x = -74),
          colorkey = FALSE,
          shade=TRUE,
          light.source= c(0,10,10),
          shade.colors = function(irr, ref,
                                  height, w = 0.4)
              grey(w*irr + (1 - w)*(1 - (1 - ref)^0.4)),
          aspect = c(1, 0.65))

## Contours of values
biv_example1 <-
```

```
    dat %>%
    filter(value != "difference") %>%
    ggplot(aes(x, y, z = z)) +
    geom_tile(aes(fill = z)) +
    facet_grid(part ~ value) +
    scale_fill_distiller(palette = "Reds")

biv_example1

## Contour of differences
biv_example2 <-
    dat %>%
    filter(value == "difference") %>%
    ggplot(aes(x, y, z = z)) +
    geom_tile(aes(fill = z)) +
    facet_grid(part ~ value) +
    scale_fill_distiller(palette = "Spectral")

biv_example2
```

The result of `fourierin()` was stored above in `eval`, which is a list with three elements: two vectors with the corresponding evaluation grid values in each coordinate direction and a complex matrix containing the Fourier integral values. If we do not wish to evaluate the Fourier integrals on a regular grid and instead wish to evaluate these at, say l bivariate points, then we must pass a $l \times 2$ matrix in the argument `w` and the function will return a vector of size l with the Fourier integrals, rather than a list. In the bivariate situation here, the function f must be able to receive a two-column matrix with m rows, where m is the number of points where the Fourier integral will be evaluated.

Corresponding to this bivariate example, we have generated three plots to compare the approximation from Fourier integrals to the underlying truth (i.e., compare the approximated and true characteristic functions of the bivariate normal distribution). In Figure 2, we present the surface plots of the approximated and the true values, as well as their differences for both the real and imaginary parts. One observes that differences are small, indicating the adequacy of the numerical integration.

For a different perspective of the resulting Fourier integration, Figure 3 presents a contour plot showing the approximated and true values of the bivariate normal characteristic function, for both real and imaginary parts. We show a tile plot of the differences in Figure 4. Observe that the range of differences in Figure 4 is relatively much smaller than the values in Figure 3.

Speed comparison

Through a small numerical study, here we compare the differences in execution times using `fourierin()` for integration at points on a regular grid, both with or without FFT steps, considering univariate and bivariate Fourier integrals. Figure 5 shows timing results for a univariate example of the integral in (1) evaluated on a grid, while Figure 6 presents timing results for a bivariate example. Note that the reported time units differ between these figures, as the bivariate case naturally requires more time. These figures provide evidence that, for evaluating integrals on a regular grid, the FFT option in `fourierin()` creates large advantages in time.

The code that was used to generate Figure 5 and 6 is below.

```
## ------------------------------------------------------------------
## Univariate speed test
## ------------------------------------------------------------------

library(fourierin)
library(dplyr)
library(purrr)
library(ggplot2)
library(microbenchmark)

## Test speed at several resolutions
resolution <- 2^(3:8)

## Function to be tested
```

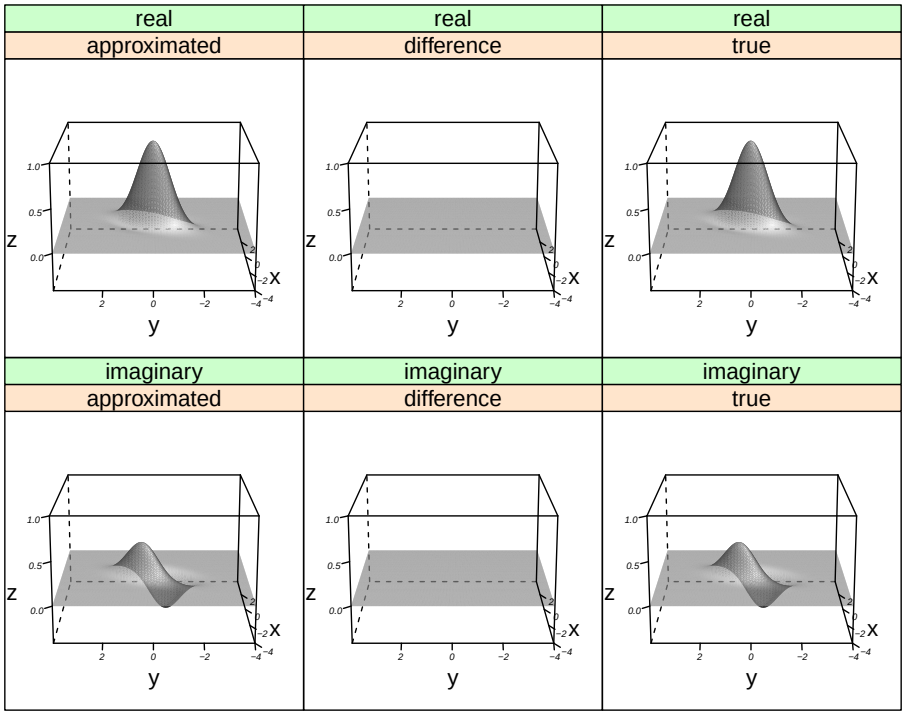


Figure 2: Example of `fourierin` function for univariate function at resolution 128×128: Obtaining the characteristic function of a bivariate normal distribution from its density. See Equation 4. This panel contains every combination of approximation-true-difference with real-imaginary parts.

```
myfnc <- function(t) exp(-t^2/2)

## Aux. function
compute_times <- function(resol){
    out <-
        microbenchmark(
            fourierin_1d(f = myfnc, -5, 5, -3, 3, -1, -1, resol),
            fourierin_1d(f = myfnc, -5, 5, -3, 3, -1, -1, resol,
                         use_fft = FALSE),
            times = 5) %>%
        as.data.frame()
    ## Rename levels
    levels(out$expr) <- c("yes", "no")
    ## Obtain median of time.
    out %>%
        group_by(expr) %>%
        summarize(time = median(time*1e-6),
                  resolution = resol) %>%
        rename(FFT = expr)
}

speed1 <- resolution %>%
    map_df(compute_times) %>%
    mutate(resolution = as.factor(resolution)) %>%
    ggplot(aes(resolution, log(time), color = FFT)) +
    geom_point(size = 2, aes(shape = FFT)) +
```

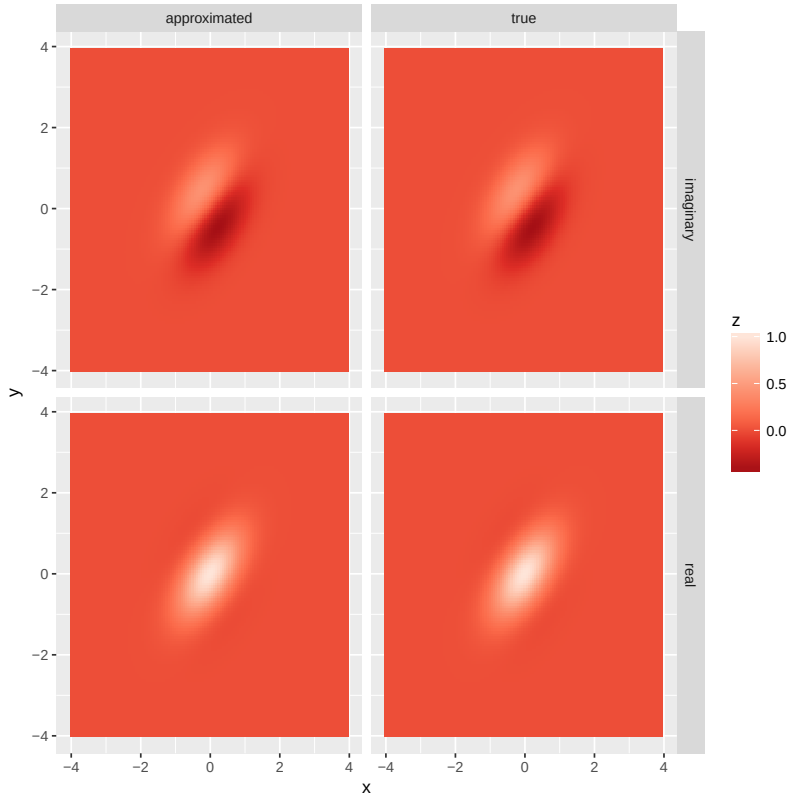


Figure 3: Example of `fourierin` function for univariate function at resolution 128×128: Obtaining the characteristic function of a bivariate normal distribution from its density. See Equation 4. Each combination the approximated and true values are shown for both the real and imaginary parts.

```
    geom_line(aes(linetype = FFT, group = FFT)) +
    ylab("time (in log-milliseconds)")

speed1

## ------------------------------------------------------------------
## Bivariate test
## ------------------------------------------------------------------

## Load packages
library(fourierin)
library(dplyr)
library(purrr)
library(ggplot2)
library(microbenchmark)

## Test speed at several resolutions
resolution <- 2^(3:7)

## Bivariate function to be tested
myfnc <- function(x) dnorm(x[, 1])*dnorm(x[, 2])

## Aux. function
compute_times <- function(resol){
    resol <- rep(resol, 2)
    out <-
        microbenchmark(
            fourierin(myfnc,
                      lower_int = c(-8, -6), upper_int = c(6, 8),
                      lower_eval = c(-4, -4), upper_eval = c(4, 4),
                      const_adj = 1, freq_adj = 1,
```

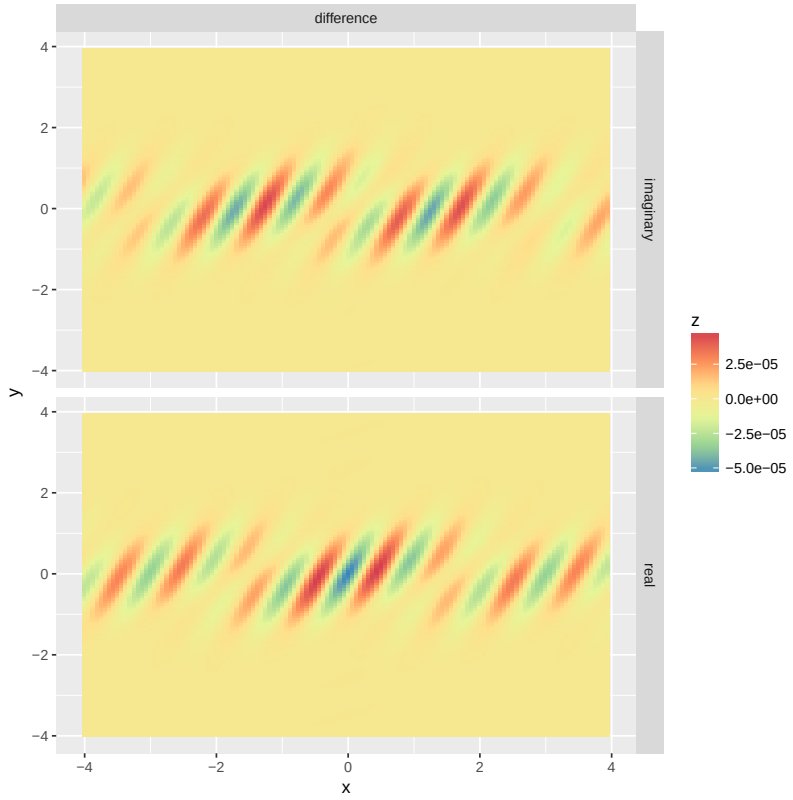


Figure 4: Example of `fourierin` function for univariate function at resolution 128×128: Obtaining the characteristic function of a bivariate normal distribution from its density. See Equation 4. This plot show the difference between the approximated and true values for the real and imaginary parts.

```
                  resolution = resol),
            fourierin(myfnc,
                  lower_int = c(-8, -6), upper_int = c(6, 8),
                  lower_eval = c(-4, -4), upper_eval = c(4, 4),
                  const_adj = 1, freq_adj = 1,
                  resolution = resol, use_fft = FALSE),
            times = 3) %>%
        as.data.frame()

    ## Rename levels
    levels(out$expr) <- c("yes", "no")
    ## Obtain median of time.
    out %>%
        group_by(expr) %>%
        summarize(time = median(time*1e-9),
                  resolution = resol[1]) %>%
        rename(FFT = expr)
}

## Values
comparison <-
    resolution %>%
    map_df(compute_times)

fctr_order <-
    unique(comparison$resolution) %>%
    paste(., ., sep = "x")

## Plot
speed2 <- comparison %>%
    mutate(resolution = paste(resolution, resolution, sep = "x"),
```

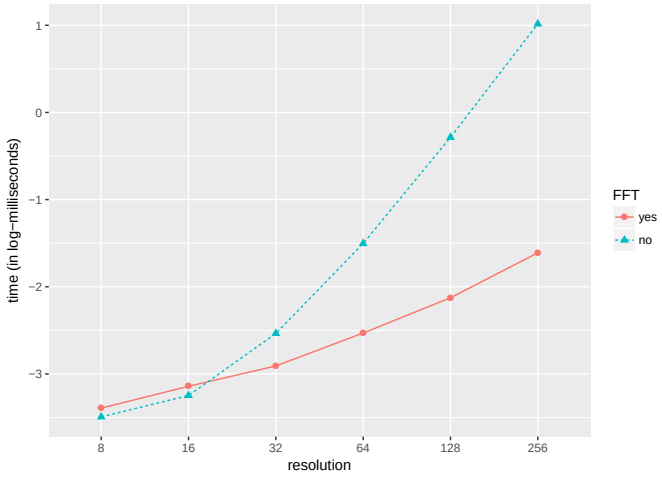


Figure 5: Example of a univariate Fourier integral over grids of several (power of two) sizes. Specifically, the standard normal density is being recovered using the Fourier inversion formula. Time is in log-milliseconds. The Fourier integral has been applied five times for every resolution and each dot represents the mean for the corresponding grid size and method. Observe that both, x and y axis are in logarithmic scale.

```
        resolution = ordered(resolution, levels = fctr_order)) %>%
    ggplot(aes(resolution, log(time), color = FFT)) +
    geom_point(size = 2, aes(shape = FFT)) +
    geom_line(aes(linetype = FFT, group = FFT)) +
    ylab("time (in log-seconds)")

speed2
```

Summary

Continuous Fourier integrals/transforms are useful in statistics for computation of probability densities from characteristic functions, as well as the reverse, when describing probability structure; see the "Examples" section for some demonstrations. The usefulness and potential application of Fourier integrals, however, also extends to other contexts of physics and mathematics, as well as to statistical inference (e.g., types of density estimation). For this reason, we have developed the **fourierin** package as a tool for computing Fourier integrals over collections of evaluation points, where repeat evaluation steps and often complicated numerical integrations are involved. When evaluation points fall on a regular grid, **fourierin** allows use of a Fast Fourier Transform as a key ingredient for rapid numerical approximation of Fourier-type integrals.

In "Speed Comparison," we presented evidence of the gain in time when using this fast implementation of `fourierin()` on regular grids, while we also illustrated the versatility of **fourierin** in "Examples" section. At present (version 0.2.1), the **fourierin** package performs univariate and bivariate Fourier integration. An extension of the package to address higher dimensional integration will be included in future versions.

Bibliography

K. B. Athreya and S. N. Lahiri. *Measure theory and probability theory*. Springer Science & Business Media, 2006. [p72]

D. H. Bailey and P. N. Swarztrauber. A fast method for the numerical evaluation of continuous Fourier and Laplace transforms. *SIAM Journal on Scientific Computing*, 15(5):1105–1110, 1994. [p72]

G. Basulto-Elias. *fourierin: Computes Numeric Fourier Integrals*, 2017. URL https://CRAN.R-project.org/package=fourierin. R package version 0.2.2. [p72]

D. Eddelbuettel and C. Sanderson. Rcpparmadillo: Accelerating R with high-performance C++ linear algebra. *Computational Statistics and Data Analysis*, 71:1054–1063, March 2014. URL http://dx.doi.org/10.1016/j.csda.2013.02.005. [p72]

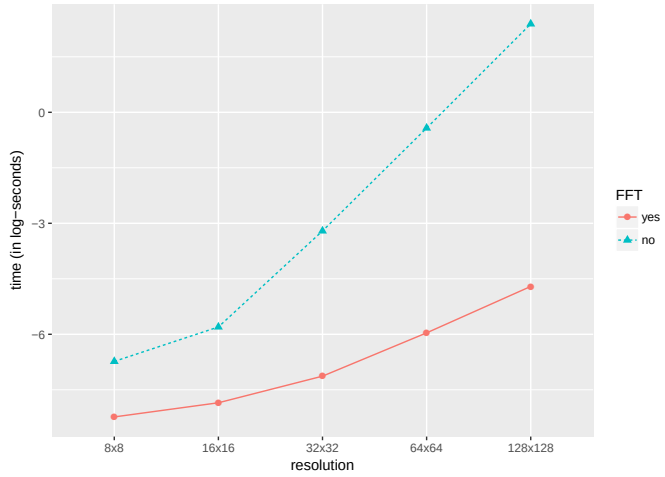


Figure 6: Example of a bivariate Fourier integral over grids of several (power of two) sizes. Both axis have the same resolution. Specifically, the characteristic function of a bivariate normal distribution is being computed. Time is in log-seconds (unlike 5). The Fourier integral has been applied five times for every resolution and each dot represents the mean for the corresponding grid size and method. Observe that both, x and y axis are in logarithmic scale.

G. Inverarity. Fast computation of multidimensional Fourier integrals. *SIAM Journal on Scientific Computing*, 24(2):645–651, 2002. [p72, 73, 75]

A. Meister. *Deconvolution problems in nonparametric statistics*, volume 193. Springer, 2009. [p72]

J. Stirnemann, A. Samson, and F. C. C. from Claire Lacour. *deamer: Deconvolution density estimation with adaptive methods for a variable prone to measurement error*, 2012. URL https://CRAN.R-project.org/package=deamer. R package version 1.0. [p72]

Guillermo Basulto-Elias
Iowa State University
Ames, IA
United States
basulto@iastate.edu

Alicia Carriquiry
Iowa State University
Ames, IA
United States
alicia@iastate.edu

Kris De Brabanter
Iowa State University
Ames, IA
United States
kbrabant@iastate.edu

Daniel J. Nordman
Iowa State University
Ames, IA
United States
dnordman@iastate.edu

Discrete Time Markov Chains with R

by Giorgio Alfredo Spedicato

Abstract The **markovchain** package aims to provide S4 classes and methods to easily handle Discrete Time Markov Chains (DTMCs), filling the gap with what is currently available in the CRAN repository. In this work, I provide an exhaustive description of the main functions included in the package, as well as hands-on examples.

Introduction

DTMCs are a notable class of stochastic processes. Although their basic theory is not overly complex, they are extremely effective to model categorical data sequences (Ching et al., 2008). To illustrate, notable applications can be found in linguistic (see Markov's original paper Markov (1907)), information theory (Google original algorithm is based on Markov Chains theory, Lawrence Page et al. (1999)), medicine (transition across HIV severity states, Craig and Sendi (2002)), economics and sociology (Jones (1997) shows an application of Markov Chains to model social mobility).

The **markovchain** package (Spedicato, Giorgio Alfredo, 2016) provides an efficient tool to create, manage and analyse Markov Chains (MCs). Some of the main features include the possibility to: validate the input transition matrix, plot the transition matrix as a graph diagram, perform structural analysis of DTMCs (e.g. classification of transition matrices and states, analysis of the stationary distribution, etc ...), perform statistical inference (such as fitting transition matrices from various input data, simulating stochastic processes trajectories from a given DTMC, etc..). The author believes that no R package provides a unified infrastructure to easily manage DTMCs as **markovchain** does at the time this paper is being drafted.

The package targets data scientists using DTMC, Academia members, supporting faculty instructors, as well as students of undergraduate courses on Stochastic Processes.

The paper will be organized as follows: Section 14.2 gives a brief overview on R packages and alternative software that provide similar functionalities, Section 14.3 reviews DTMC basic theory, Section 14.4 discusses the package design and structure, Section 14.5 shows how to create and manipulate homogeneous DTMCs, Section 14.6 and Section 14.7 respectively present the functions created to perform structural analysis, and statistical inference on DTMCs. A brief overview of the functionalities written to deal with non - homogeneous discrete dime Markov chains (NHDTMCs) is provided in Section 14.8. A discussion on numerical reliability and computational performance is provided in Section 14.9. Finally, Section 14.10 draws final conclusions and briefly discusses future potential developments of the package.

Analysis of existing DTMC-related software

As reviewed later in more details, a DTMC is defined by a stochastic matrix known as transition matrix (TM), which is a square matrix satisfying Equation 1.

$$\begin{cases} P_{ij} \in [0,1] \, \forall i,j \\ \sum_i P_{ij} = 1 \end{cases} \tag{1}$$

Although defining a stochastic matrix is trivial in any mathematical or statistical software, a DTMC dedicated infrastructure can provide object oriented programmed methods to verify the validity of the input data (i.e. if the input matrix is a stochastic one) , as well as to perform structural analysis on DTMC objects.

Various packages mention MCs - related models in the CRAN repository, whereby a few of them will be now reviewed. The **clickstream** package (Scholz, 2016), on CRAN since 2014, aims to model websites click stream using higher order Markov Chains. It provides a `MarkovChain` S4 class that is similar to the `markovchain` class. Further, **DTMCPack** (Nicholson, William, 2013) and **MTCM** (Bessi, Alessandro, 2015) also work with DTMCs but provide even more limited functions: the first one focuses on creating simulations from a given DTMC, whilst the second contains only one function for estimating the underlying transition matrix for a given categorical sequence. Moreover, none of them appears to have been updated since 2015. The coverage of functionalities provided by **markovchain** package for analysing DTMCs appears to be more complete than the above mentioned packages, since none of them provides methods for importing or coercing transition matrices from other objects, such as R `matrices` or `data.frames`. Furthermore, **markovchain** is the only package providing a quick graph plotting facility for DTMC objects. The same applies when considering the functionalities used

to perform structural analysis of transition matrices and to fit DTMCs from various kind of input data. More interestingly, the **FuzzyStatProb** package (Pablo J. Villacorta and José L. Verdegay, 2016) gives an alternative approach for estimating the parameters of DTMCs using "fuzzy logic".

This review voluntarily omits discussing packages that are not specifically focused on DTMC. Nonetheless, the **depmixS4** (Visser and Speekenbrink, 2010) and the **HMM** (Himmelmann, 2010) packages deal with Hidden Markov Models (HMMs). In addition, the number of R packages focused on the estimation of statistical models using the Markov Chain Monte Carlo simulation approach is sensibly bigger. Finally, the **msm** (Jackson, 2011), **heemod** (Antoine Filipovi et al., 2017) and the **TPmsm** packages (Artur Araújo et al., 2014) focus on health applications of multi - state analysis using different kinds of models, including Markov-related ones among them.

Finally, among other well known software used in Mathematics and Statistics, only Mathematica (Wolfram Research, Inc., 2013) provides routines specifically written to deal with Markov processes at the author's knowledge. Nevertheless, the analysis of DTMCs could be easily handled within the Matlab programming language (MATLAB, 2017) due to its well known linear algebra capabilities.

Review of underlying theory

In this section a brief review of the theory of DTMCs is presented. Readers willing to dive deeper can inspect Cassandras (1993) and Grinstead and Snell (2012).

A *DTMC* is a stochastic process whose domain is a discrete set of states, $\{s_1, s_2, \ldots, s_k\}$. The chain starts in a generic state at time zero and moves from a state to another by steps. Let p_{ij} be the probability that a chain currently in state s_i moves to state s_j at the next step. The key characteristic of DTMC processes is that p_{ij} does not depend upon the previous state in the chain. The probability p_{ij} for a (finite) DTMC is defined by a transition matrix previously introduced (see Equation 1). It is also possible to define the TM by column, under the constraint that the sum of the elements in each column is 1.

To illustrate, a few toy - examples on transition matrices are now presented; the "Land of Oz" weather Matrix, Kemeny et al. (1974). Equation 2 shows the transition probability between (R)ainy, (N)ice and (S)now weathers.

$$\begin{bmatrix} & R & N & S \\ R & 0.5 & 0.25 & 0.25 \\ N & 0.5 & 0 & 0.5 \\ S & 0.25 & 0.25 & 0.5 \end{bmatrix} \tag{2}$$

Further, the Mathematica Matrix 3, taken from the Mathematica 9 Computer Algebra System manual (Wolfram Research, Inc., 2013), that will be used when discussing the analysis the structural proprieties of DTMCs, is as follows:

$$\begin{bmatrix} & A & B & C & D \\ A & 0.5 & 0.5 & 0 & 0 \\ B & 0.5 & 0.5 & 0 & 0 \\ C & 0.25 & 0.25 & 0.25 & 0.25 \\ D & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3}$$

Simple operations on TMs allow to understand structural proprieties of DTMCs. For example, the $n-th$ power of P is a matrix whose entries represent the probabilities that a DTMC in state s_i at time t will be in state s_j at time $t+n$. In particular, if u_t is the probability vector for time t (that is, a vector whose $j-th$ entries represent the probability that the chain will be in the $j-th$ state at time t), then the distribution of the chain at time $t+n$ is given by $u_n = u * P^n$. Main properties of Markov chains are now presented.

A state s_i is *reachable* from state s_j if $\exists n \rightarrow p_{ij}^n > 0$. If the inverse is also true then s_i and s_j are said to *communicate*. For each MC, there always exists a unique decomposition of the state space into a sequence of disjoint subsets in which all the states within each subset communicate. Each subset is known as a *communicating class* of the MC. It is possible to link this decomposition to graph theory, since the communicating classes represent the strongly connected components of the graph underlying the transition matrix (Jarvis and Shier, 1999).

A state s_j of a DTMC is said to be *absorbing* if it is impossible to leave it, meaning $p_{jj} = 1$. An *absorbing Markov chain* is a chain that contains at least one absorbing state which can be reached, not necessarily in a single step. Non - absorbing states of an absorbing MC are defined as *transient states*. In addition, states that can be visited more than once by the MC are known as *recurrent states*.

If a DTMC contains $r \geq 1$ absorbing states, it is possible to re-arrange their order by separating

transient and absorbing states such that the t transient states come before the r absorbing ones. Such re-arranged matrix is said to be in *canonical form* (see Equation 4), where its composition can be represented by sub - matrices.

$$\begin{pmatrix} Q_{t,t} & R_{t,r} \\ 0_{r,t} & I_{r,r} \end{pmatrix} \tag{4}$$

Such matrices are: Q (a t-square sub - matrix containing the transition probabilities across transient states), R (a nonzero t-by-r matrix containing transition probabilities from non-absorbing to absorbing states), 0 (an r-by-t zero matrix), and I_r (an r-by-r identity matrix). It is possible to use these matrices to calculate various structural proprieties of the DTMC. Since $\lim_{n\to\infty} Q^n = 0$, it can be shown that in every absorbing matrix the probability to be eventually absorbed is 1, regardless of the state where the MC is initiated.

Further, in Equation 5 the *fundamental matrix* is presented, where the generic n_{ij} entry expresses the expected number of times the process will transit in state s_j, given that it started in state s_i. Also, the i-th entry of vector $t = N * \bar{1}$, being $\bar{1}$ a t-sized vector of ones, expresses the expected number of steps before an absorbing DTMC, started in state s_i, is absorbed. The b_{ij} entries of matrix $B = N * R$ are the probabilities that a DTMC started in state s_i will eventually be absorbed in state s_j. Finally, the probability of visiting the transient state j when starting from the transient state i is the h_{ij} entry of the matrix $H = (N - I_t) * N_{dg}^{-1}$, being dg the diagonal operator.

$$N = (I - Q)^{-}1 = I + \sum_{i=0,1,\ldots,\infty} Q^i \tag{5}$$

A DTMC is said to be *ergodic* if there exist a number N such that it is possible to reach every state in at most N steps. If $P^n > 0$ for some n, then P is a *regular* DTMC.

Fixed row vectors $\bar{w}$, also known as *steady state vectors*, are vectors such that $\bar{w}P = \bar{w}$. Mathematically, they correspond to eigenvectors associated to unitary eigenvalues of the TM. It can be shown that $\lim_{n\to\infty} v * P^n = w$ and that $\lim_{n\to\infty} P^n = W$, where v is a generic stochastic vector and w is a matrix where all rows are $\bar{w}$.

The *mean first passage time* m_{ij} is the expected the number of steps needed to reach state s_j starting from state s_i, where $m_{ii} = 0$ by convention. For ergodic MCs, r_i is the mean recurrence time, that is the expected number of steps to return to s_i from s_i. It is possible to prove that $r_i = \frac{1}{w_i}$, where w_i is the i-th entry of $\bar{w}$. Further, let D be a diagonal matrix, in which the diagonal elements come from r_i, and let C be a matrix filled with ones. It is then possible to get the mean first passage matrix M from Equation 6.

$$(I - P) = C - M \tag{6}$$

Let $Z = (I - P + M)^{-1}$ be the fundamental matrix for an ergodic MC. It is possible to write m_{ij} as a function of Z and $\bar{w}$, as Equation 7 shows.

$$m_{ij} = \frac{z_{jj} - z_{ij}}{w_j} \tag{7}$$

A further topic in structural analysis of irreducible DTMCs is periodicity. The period of a state s_i, denoted as $d(i)$, is the greatest common divisor of n for which $p_{ii}^n > 0$. If the period is 1, the state is aperiodic, while if the period is greater than 2, the state is periodic; all states in the same class share the same period.

Given a generic DTMC, it is possible to simulate stochastic trajectories following the underlying MC from the TM. Given an initial state $s(t) = j$, the $s(t+1)$ state is sampled from the multinomial distribution whose probabilities are expressed by the j-th row. The sampled state indicates from which row the probabilities to sample $s(t+2)$ are taken from. Also, given a sample sequence, it is possible to estimate the TM of the underlying DTMC. Equation 8 shows the maximum likelihood estimator (MLE) of the TM p_{ij} entry, being the n_{ij} elements the number of sequences $\left(X_t = s_i, X_{t+1} = s_j\right)$ counted in the sample, that is:

$$\hat{p}_{ij}^{MLE} = \frac{n_{ij}}{\sum_{u=1}^{k} n_{iu}}. \tag{8}$$

Equation 10 shows asymptotic confidence intervals for p_{ij}. The bootstrap approach allows to define non - parametric ones.

$$LowerEndpoint_{ij} = p_{ij} - 1.96 * SE_{ij} \quad (9)$$

$$UpperEndpoint_{ij} = p_{ij} + 1.96 * SE_{ij} \quad (10)$$

The mode of the X_{t+1} conditional distribution given $X_t = s_j$ represents the prediction from a given DTMC and the current chain state $X_t = s_j$.

In conclusion, the **markovchain** package allows to perform statistical analysis on NHDTMCs, in the special case where they can be treated as sequential lists of DTMCs.

Implementation design and details

The **markovchain** package has been originally written in "native" R. Most functions have been therefore ported in **Rcpp** (Eddelbuettel, Dirk, 2013) since 2015, yielding sensible improvements in computational time. Other dependencies of **markovchain** are: **igraph** Csardi, Gabor and Nepusz, Tamas (2006), **matlab** Roebuck (2014), **Matrix** Bates and Maechler (2016) and **expm** Goulet et al. (2015) (for operation on matrices), and the **method** package for defining S4 classes.

Homogeneous DTMCs are defined by a dedicated S4 class, `"markovchain"`. Such class is defined by the following slots:

1. `states`: a `character` vector, listing the states for which transition probabilities are defined.
2. `byrow`: a `logical` variable, indicating whether transition probabilities are shown by row or by column.
3. `transitionMatrix`: a `matrix` variable defining the TM.
4. `name`: an optional `character` variable to name the DTMC.

A `"markovchain"` S4 class has been designed based on Chambers, J.M. (2008) suggestions. For example, a S4 `setValidity` method checks the coherence of any newly created `markovchain` object, by verifying that either the rows or columns of the transition matrix sum to one, and that all elements are bounded between 0 and 1.
Another S4 class,`"markovchainList"`, has been created for handling non - homogeneous DTMCs. Finally, the package provides functions and S4 to analyse continuous MCs, as well as higher order MCs, although their discussion is beyond the scope of this paper.

Three vignettes documents the **markovchain** package. The first one broadly describes the functionalities of the package and it also presents real - world applications of DTMCs using the package. The second one, written using **knit** and **rmarkdown**, is a beamer presentation that quickly introduces the key functionalities of the package. The third one presents experimental functions for higher order and multivariate MCs. Finally, the `www.github.com/spedygiorgio/markovchain` GitHub page hosts the package's wiki as well as its development version.

Creating and manipulating markovchain objects

The package is loaded within R as follows:

```
library("markovchain")
```

Creating a `markovchain` object is easy, and can be done with provided code.

```
#using "long" approach for mcWeather

weatherStates <- c("rainy", "nice", "sunny")
weatherMatrix <- matrix(data = c(0.50, 0.25, 0.25,
0.5, 0.0, 0.5, 0.25, 0.25, 0.5), byrow = TRUE,
nrow = 3,dimnames = list(weatherStates, weatherStates))
mcWeather <- new("markovchain", states = weatherStates,
byrow = TRUE, transitionMatrix = weatherMatrix,
name = "Weather")

#using "quick" approach on Mathematica's DTMC

mathematicaMatr <- matrix(c(1/2, 1/2, 0, 0, 1/2, 1/2,
```

```
0, 0, 1/4, 1/4, 1/4, 1/4, 0, 0, 0, 1),byrow=TRUE, nrow=4)
mathematicaMc<-as(mathematicaMatr, "markovchain")

#both are markovchain objects
is(mcWeather,"markovchain")
[1] TRUE
is(mathematicaMc,"markovchain")
[1] TRUE
```

Commenting on the code snippet, the first part shows the "standard" approach to create a `markovchain`, by calling the `new` S4 method, while the second part shows the "quick" method, by coercing a `matrix` object into a `markovchain` one.

Specific methods allow to print and plot `markovchain` objects:

```
plot(mcWeather, main="Weather Markov Chain")
```

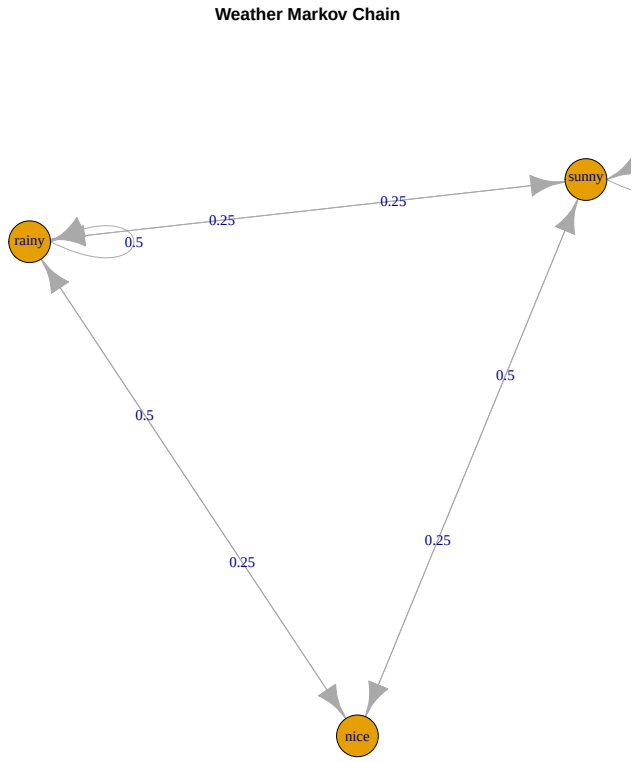


Figure 1: Plotting a `markovchain` object.

In particular, the `plot` method makes use of **igraph** package to draw the TM by default. It is possible to modify the plot either by passing further parameters via `...` or by choosing another plotting devices, as further specified in the package vignette.

Algebraic operations have been defined in `"markovchain"` classes, as of the following example:

```
initialState <- c(0, 1, 0)

#multiplication

after2Days <- initialState * (mcWeather * mcWeather)
after2Days
```

```
     rainy nice sunny
[1,] 0.375 0.25 0.375
```

in which multiplications by vectors and exponentiation are intuitively performed, making easy to find the distribution of states at the n-th step.

A power operator also exists, ^, and it is based on the **expm** package (Goulet et al., 2015), providing efficient matrix exponentiation.

```
#after two days (by square power)

mcWeather^2

Weather^2
 A  3 - dimensional discrete MC defined by the following states:
 rainy, nice, sunny
 The transition matrix  (by rows)  is defined as follows:
       rainy   nice  sunny
rainy 0.4375 0.1875 0.3750
nice  0.3750 0.2500 0.3750
sunny 0.3750 0.1875 0.437
```

Finally, logical operators have been defined as well.

```
#logical equality and inequality
mcWeather==mcWeather
[1] TRUE
mcWeather!=mathematicaMc
[1] TRUE
```

Both the algebraic and logical operators have been defined by overriding standard R operators, providing a more concise and "natural" code, which can bring the advantage of being more appealing to a novice user, by executing certain operations on TM in an efficient way. Such approach has been stressed in both the class help file and the package vignette code to make the final user fully aware of any potential drawbacks of such choice.

Various convenience S4 methods have been defined to easily manipulate and manage `markovchain` objects. In the following examples, some of the implemented methods in the "`markovchain`" class are presented, allowing to: get and set names, return the MC dimension, transpose the transition matrix, and directly access the transition probabilities.

```
#some markovchain specific methods

#naming
name(mcWeather)
[1] "Weather"

name(mathematicaMc) <- "Mathematica Markov Chain"
#list of defined states
states(mcWeather)
[1] "rainy" "nice"  "sunny"

#the dimension
dim(mcWeather)
[1] 3

#transpose operator
t(mcWeather)

Unnamed Markov chain
A  3 - dimensional discrete Markov Chain defined by the following states:
rainy, nice, sunny
The transition matrix  (by cols)  is defined as follows:
      rainy nice sunny
rainy  0.50  0.5  0.25
nice   0.25  0.0  0.25
```

```
sunny  0.25  0.5  0.50

#two ways to get transition probabilities
transitionProbability(mcWeather, "nice", "sunny")
[1] 0.5
mcWeather[2,3]
 [1] 0.5
```

Finally, coerce methods allow to both import and export `markovchain` classes. Following, a brief example on how to transform a `markovchain` object into a `data.frame` one.

```
#exporting to data.frame and matrix

as(mcWeather, "data.frame")
      t0    t1 prob
 1 rainy rainy 0.50
 2 rainy  nice 0.25
 3 rainy sunny 0.25
 4  nice rainy 0.50
 5  nice  nice 0.00
 6  nice sunny 0.50
 7 sunny rainy 0.25
 8 sunny  nice 0.25
 9 sunny sunny 0.50
```

Structural properties of finite Markov chains

The **markovchain** package embeds functions to analyse the structural proprieties of DTMC. For example, it is possible to find the stationary distribution, as well as classify the states. Feres, Renaldo (2007) and Montgomery, James (2009) provide a full description of the algorithms underlying these functions, whilst a more theoretical perspective can be found in Brémaud, Pierre (1999). The Mathematica MC will be used to illustrate such features.

The `summary` method provides an overview of the structural properties of the DTMC process underlying the `markovchain` object.

```
#plotting and summarizing
plot(mathematicaMc)
```

```
summary(mathematicaMc)
 Mathematica Markov Chain  Markov chain that is composed by:
 Closed classes:
 s1 s2
 s4
 Recurrent classes:
 {s1,s2},{s4}
 Transient classes:
 {s3}
 The Markov chain is not irreducible
 The absorbing states are: s4
```

In the above example, closed and transient classes are identified, irreducibility checks are executed, and a list of absorbing states is returned. Further, it is known that a finite MC has at least one steady-state distribution, and the `steadyStates` method can be used to obtain it. To illustrate, for the `mcWeather` matrix there exist a one - dimensional solution, since the underlying TM is irreducible. A higher dimensional solution is given when the irreducibility property does not hold, as of the second example.

```
#probability with DTMC: stationary distribution
## when the TM is irreducibile
steadyStates(mcWeather)
      rainy nice sunny
```

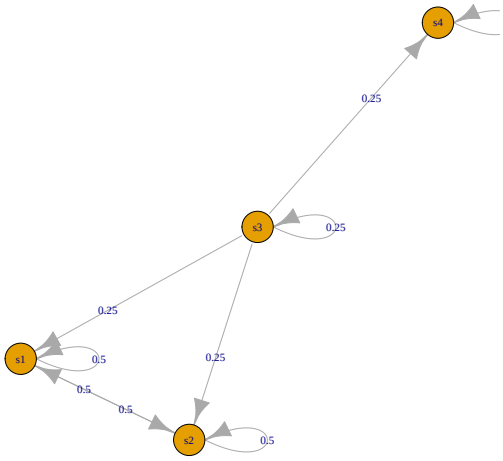


Figure 2: Plot of the Mathematica MC DTMC process.

```
 [1,]   0.4  0.2   0.4
## when reducibility applies
steadyStates(mathematicaMc)
       s1  s2 s3 s4
 [1,] 0.5 0.5  0  0
 [2,] 0.0 0.0  0  1
```

Specific methods and functions return transient and absorbing states, and check whether any state is accessible from another. Recurrent and communicating classes can be easily identified as well.

```
#probability with DTMC: classifying states

transientStates(mathematicaMc)
 [1] "s3"

absorbingStates(mathematicaMc)
 [1] "s4"

is.accessible(mathematicaMc, from = "s1",to="s4")
 [1] FALSE

#identifying recurrent and transient classes

recurrentClasses(mathematicaMc)
 [[1]]
 [1] "s1" "s2"

 [[2]]
 [1] "s4"

communicatingClasses(mathematicaMc)
 [[1]]
 [1] "s1" "s2"

 [[2]]
```

```
 [1] "s3"

 [[3]]
 [1] "s4"
```

The communicating classes are the strongly connected components of the graph underlying the DTMC. It is possible to convert a `markovchain` object into an `igraph` one, in order to use **igraph**'s package clustering function to identify the strongly connected components as the following example displays:

```
library(igraph)
#converting to igraph

mathematica.igraph<-as(mathematicaMc,"igraph")

#finding and formatting the clusters
SCC <- clusters(mathematica.igraph, mode="strong")
V(mathematica.igraph)$color <- rainbow(SCC$no)[SCC$membership]

#plotting
plot(mathematica.igraph, mark.groups = split(1:vcount(mathematica.igraph), SCC$membership),
main="Communicating classes - strongly connected components")
```

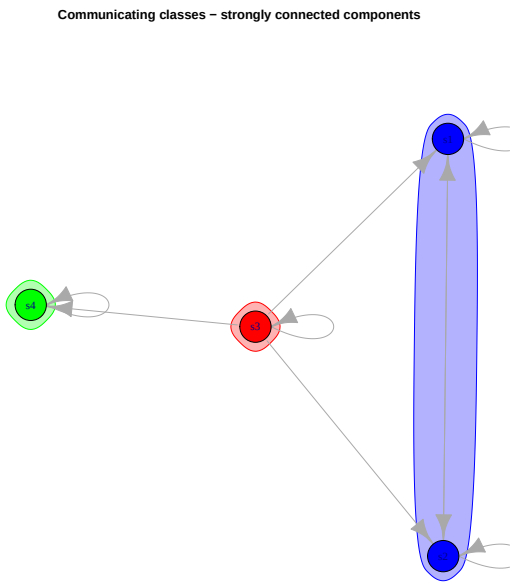


Figure 3: The communicating classes are the strongly connected components of the graph underlying the DTMC.

The three distinct clusters identified with different colors by the **igraph** package match with the partition of the transition matrix into communicating classes given by **markovchain** package's `communicatingClasses` function.

We now illustrate the Canonical Form and the Fundamental Matrix concepts using another example taken from classical theory: The Flipping Coin problem. Specifically, consider repeatedly flipping a fair coin until the sequence (heads, tails, heads) appears; it is possible to model such process using a DTMC with four states: "E" empty initial sequence, "H" head, "HT" head followed by tail, "HTH" head followed by tail and head.

```
# Flipping Coin Problem
```

```
## defining the matrix

flippingMatr <- matrix(0, nrow=4, ncol=4)
flippingMatr[1,1:2] <- 0.5
flippingMatr[2,2:3] <- 0.5
flippingMatr[3,c(1,4)] <- 0.5
flippingMatr[4,4] <- 1
rownames(flippingMatr) <-
colnames(flippingMatr) <- c("E","H","HT","HTH")

## creating the corresponding DTMC
flippingMc <- as(flippingMatr,"markovchain")
```

The following function returns the Q, R, and I matrices by properly combining functions and methods from the **markovchain** package.

```
#function to extract matrices

extractMatrices <- function(mcObj) {

  require(matlab)
  mcObj <- canonicForm(object = mcObj)

  #get the indices of transient and absorbing

  transIdx <- which(states(mcObj) %in% transientStates(mcObj))
  absIdx <- which(states(mcObj) %in% absorbingStates(mcObj))

  #get the Q, R and I matrices

  Q <- as.matrix(mcObj@transitionMatrix[transIdx,transIdx])
  R <- as.matrix(mcObj@transitionMatrix[transIdx,absIdx])
  I <- as.matrix(mcObj@transitionMatrix[absIdx, absIdx])

  #get the fundamental matrix

  N <- solve(eye(size(Q)) - Q)

  #computing final absorbion probabilities

  NR <- N %*% R

  #return
  out <- list(
    canonicalForm = mcObj,
    Q = Q,
    R = R,
    I = I,
    N=N,
    NR=NR
  )
  return(out)
}
```

The expected number of visits to transient state j starting from state i can be found in the corresponding entries of the *fundamental matrix* $N = (I_t - Q)^{-1}$. Therefore, the fundamental matrix for the above DTMC is:

```
#decompose the matrix

flipping.Dec <- extractMatrices(mcObj = flippingMc)
```

```
flipping.Fund <- flipping.Dec$N

#showing the fundamental matrix

flipping.Fund
```

```
   E H HT
E  4 4  2
H  2 4  2
HT 2 2  2
```

```
#expected number of steps before being absorbed

flipping.Fund%*%c(1,1,1)
```

```
   [,1]
E    10
H     8
HT    6
```

```
#calculating B matrix
#the probability to being absorbed in HTH state as a function of the starting transient state

flipping.B <- flipping.Fund%*%flipping.Dec$R
flipping.B
```

```
   [,1]
E     1
H     1
HT    1
```

```
#calculating H, probability of visiting transient state j starting in transient state i

flipping.H <- (flipping.Fund - matlab::eye(ncol(flipping.Fund))) *  solve(diag(diag(flipping.Fund)))
flipping.H
```

```
       E    H  HT
E  0.75 0.00 0.0
H  0.00 0.75 0.0
HT 0.00 0.00 0.5
```

The calculated fundamental matrix shows that the number of times the chain is in state HT, starting from state H is two. Also, the $N * \bar{1}$ vector indicates that if the chains starts in HT, the expected number of steps before being absorbed is eight. Since there is only one absorbing state, HTH, the probability to be absorbed in HTH is one, whichever the starting transient state is. Also, matrix H shows that the probability that a chain in state H will eventually visit again state H is 0.75.

It is possible to compute the distribution of first passage time, as the code that follows shows:

```
#first passage time

fptMc <- new("markovchain", transitionMatrix=matrix(c(0, 1/2, 1/2,1/2,0, 1/2,
1/2, 1/2, 0), byrow = TRUE,ncol=3), name="FistPassageTimeExample", states=c("a"  ,"b","c"))

firstPassage(fptMc,state = "a",5)
```

```
       a       b       c
1 0.0000 0.50000 0.50000
2 0.5000 0.25000 0.25000
3 0.2500 0.12500 0.12500
4 0.1250 0.06250 0.06250
5 0.0625 0.03125 0.03125
```

The output of `firstPassage` function shows that the probability that the first hit of state "b" occurs at the second step is 0.25.

Periodicity analysis is shown in the following last example, in which the output shows that the DTMC has a period of 2.

```
#defining a toy - model matrix for periodicity

periodicMc<-as(matrix(c(0,1,1,0),nrow=2),"markovchain")
periodicMc
```

```
 Unnamed Markov chain
  A  2 - dimensional discrete Markov Chain defined by the following states:
  s1, s2
  The transition matrix  (by rows)  is defined as follows:
    s1 s2
 s1  0  1
 s2  1  0
```

```
#computing periodicity

period(periodicMc)
```

```
 [1] 2
```

Statistical inference using markovchain package

Statistical analysis functions allow to estimate a DTMC from data and to simulate a DTMC, and can be done through the `rmarkovchain` function:

```
weathersOfDays <- rmarkovchain(n = 30, object = mcWeather, t0 = "sunny")
weathersOfDays
```

```
  [1] "sunny" "sunny" "rainy" "rainy" "rainy" "nice"  "rainy" "rainy"
  [9] "rainy" "rainy" "nice"  "rainy" "rainy" "nice"  "sunny" "nice"
 [17] "rainy" "rainy" "sunny" "rainy" "rainy" "rainy" "sunny" "rainy"
 [25] "sunny" "sunny" "sunny" "sunny" "sunny" "rainy"
```

The code shown above simulates 30 observations from the weather DTMC previously introduced.

Next, the function `createSequenceMatrix` is used to obtain the *sequence matrix*, that is the empirical transition matrix between the preceding and subsequent state, for a given sequence, whilst the function `markovchainFit` fits DTMCs. We will exemplify the use of such functions on the `rain` data set (recorded daily rainfall volume in Alofi island) bundled within the package.

```
#loading the Alofi's rain data set

data(rain)
rain$rain[1:10]
```

```
  [1] "6+"  "1-5" "1-5" "1-5" "1-5" "1-5" "1-5" "6+"  "6+"  "6+"


#obtaining the empirical transition matrix

createSequenceMatrix(stringchar = rain$rain)


      0 1-5  6+
0   362 126  60
1-5 136  90  68
6+   50  79 124



#fitting the DTMC by MLE

alofiMcFitMle <- markovchainFit(data = rain$rain, method = "mle", name = "Alofi")
alofiMcFitMle


$estimate
Alofi
 A  3 - dimensional discrete Markov Chain defined by the following states:
 0, 1-5, 6+
 The transition matrix  (by rows)  is defined as follows:
            0       1-5        6+
0   0.6605839 0.2299270 0.1094891
1-5 0.4625850 0.3061224 0.2312925
6+  0.1976285 0.3122530 0.4901186


$standardError
             0        1-5         6+
0   0.03471952 0.02048353 0.01413498
1-5 0.03966634 0.03226814 0.02804834
6+  0.02794888 0.03513120 0.04401395

$confidenceInterval
$confidenceInterval$confidenceLevel
[1] 0.95

$confidenceInterval$lowerEndpointMatrix
            0       1-5         6+
0   0.6034754 0.1962346 0.08623909
1-5 0.3973397 0.2530461 0.18515711
6+  0.1516566 0.2544673 0.41772208

$confidenceInterval$upperEndpointMatrix
            0       1-5        6+
0   0.7176925 0.2636194 0.1327390
1-5 0.5278304 0.3591988 0.2774279
6+  0.2436003 0.3700387 0.5625151


$logLikelihood
[1] -1040.419
```

Clearly, the `markovchainFit` function returns not only the pointwise estimate of the transition matrix, but also its standard error and confidence intervals. MLE estimates are provided by default, but a bootstrap one Efron, B. (1979) can also be obtained as the following code shows.

```
#estimating Alofi TM

alofiMcFitBoot <- markovchainFit(data = rain$rain, method = "bootstrap",
name = "Alofi",nboot=100)



#point estimate of the TM

alofiMcFitBoot$estimate


 Alofi
  A  3 - dimensional discrete Markov Chain defined by the following states:
  0, 1-5, 6+
  The transition matrix  (by rows)  is defined as follows:
             0       1-5        6+
 0   0.6605457 0.2314278 0.1080264
 1-5 0.4646651 0.3071925 0.2281424
 6+  0.1976978 0.3115299 0.4907723



#95 CIs

alofiMcFitBoot$standardError


               0         1-5          6+
 0   0.001957644 0.001793261 0.001318923
 1-5 0.002733252 0.002712275 0.002273845
 6+  0.002647255 0.002949244 0.003075143
```

Subsequently, the three-days forward predictions from `alofiMcFitMle` object are generated, assuming that the last two days were "1-5" and "6+" respectively. Clearly only the last state matters for a MC stochastic process.

```
#obtain a prediction

predict(object = alofiMcFitMle$estimate, newdata = c("1-5", "6+"),n.ahead = 3)


 [1] "6+" "6+" "6+"


#obtain a prediction changing t-2 state

predict(object = alofiMcFitMle$estimate, newdata = c("0", "6+"),n.ahead = 3)


 [1] "6+" "6+" "6+"
```

Non homogeneous Markov chains

Non homogeneous DTMCs (NHDTMCs) can be handled using the `"markovchainList"` S4 class, which consists in a list of `markovchain` objects.

```
#define three DTMC
```

```
matr1<-matrix(c(0.2,.8,.4,.6),byrow=TRUE,ncol=2);mc1<-as(matr1, "markovchain")
matr2<-matrix(c(0.1,.9,.2,.8),byrow=TRUE,ncol=2);mc2<-as(matr2, "markovchain")
matr3<-matrix(c(0.5,.5,.2,.8),byrow=TRUE,ncol=2);mc3<-as(matr2, "markovchain")

#create the markovchainList to store NHDTMCs

mcList<-new("markovchainList", markovchains=list(mc1,mc2,mc3), name="My McList")
mcList
```

```
 My McList  list of Markov chain(s)
 Markovchain  1
 Unnamed Markov chain
  A  2 - dimensional discrete Markov Chain defined by the following states:
  s1, s2
  The transition matrix  (by rows)  is defined as follows:
     s1  s2
 s1 0.2 0.8
 s2 0.4 0.6

 Markovchain  2
 Unnamed Markov chain
  A  2 - dimensional discrete Markov Chain defined by the following states:
  s1, s2
  The transition matrix  (by rows)  is defined as follows:
     s1  s2
 s1 0.1 0.9
 s2 0.2 0.8

 Markovchain  3
 Unnamed Markov chain
  A  2 - dimensional discrete Markov Chain defined by the following states:
  s1, s2
  The transition matrix  (by rows)  is defined as follows:
     s1  s2
 s1 0.1 0.9
 s2 0.2 0.8
```

The example above shows that creating a `markovchainList` S4 object is very simple. Moreover, the `rmarkovchain` function also works on objects from the `"markovchainList"` class.

```
#simulating a NHDTMC

mysim<-rmarkovchain(n=100, object=mcList,include.t0=TRUE,what="matrix")
head(mysim,n = 10)
```

```
       [,1] [,2] [,3] [,4]
  [1,] "s2" "s2" "s2" "s2"
  [2,] "s2" "s1" "s2" "s2"
  [3,] "s2" "s1" "s2" "s2"
  [4,] "s2" "s1" "s2" "s2"
  [5,] "s2" "s2" "s1" "s2"
  [6,] "s1" "s2" "s2" "s1"
  [7,] "s1" "s2" "s2" "s2"
  [8,] "s1" "s2" "s2" "s2"
  [9,] "s2" "s2" "s2" "s1"
 [10,] "s1" "s1" "s2" "s2"
```

Finally, it is possible to infer a non - homogeneous sequence of DTMC, that is a `markovchainList` object from a given matrix, where each row represents a single trajectory and each column stands for a different period.

```
#using holson data set

data(holson)
head(holson,n = 3)
```

```
   id time1 time2 time3 time4 time5 time6 time7 time8 time9 time10 time11
 1  1     1     1     1     1     1     1     1     1     1      1      1
 2  2     1     1     1     1     1     1     1     1     1      1      1
 3  3     1     1     1     1     1     1     1     1     1      1      1
```

```
#fitting a NHDTMCs on holson data set

nhmcFit<-markovchainListFit(holson[,2:12])

#showing estimated DTMC for time 1 -> time 2 transitions

nhmcFit$estimate[[1]]
```

```
 time1
  A  3 - dimensional discrete Markov Chain defined by the following states:
  1, 2, 3
  The transition matrix  (by rows)  is defined as follows:
            1          2         3
 1 0.94609164 0.05390836 0.0000000
 2 0.26356589 0.62790698 0.1085271
 3 0.02325581 0.18604651 0.7906977
```

```
#showing estimated DTMC for time 2 -> time 3 transitions

nhmcFit$estimate[[2]]
```

```
 time2
  A  3 - dimensional discrete Markov Chain defined by the following states:
  1, 2, 3
  The transition matrix  (by rows)  is defined as follows:
           1         2         3
 1 0.9323410 0.0676590 0.0000000
 2 0.2551724 0.5103448 0.2344828
 3 0.0000000 0.0862069 0.9137931
```

Numerical reliability and computational performance

Numerical reliability

Finding the stationary distribution is a computational - intensive task that could raise numerical issues. The **markovchain** package relies on the R linear algebra facilities (built on LAPACK routines) when the `eigen` function is called to find the stationary distribution. An initial analysis of the numerical stability of the `markovchain` matrix computation has been performed estimating the error rate when calculating the stationary distribution on a large sample of simulated DTMC of a given size k (range set between 2 and 32). Initially, dense matrices were simulated. The following algorithm was used for a given k:

1. generate N random k-sized DTMCs, where each row $\bar{r}$ has been independently sampled from a Dirichlet distribution, $\bar{r} \sim Dir(\bar{\alpha})$. The Dirichlet parameters' vector, $\bar{\alpha}$ is itself assumed to follow an Uniform distribution (sampled independently for each row).
2. try to compute the steady - state distribution for the simulated DTMC.

3. calculate the success rate as the relative frequency of previous step non - failures at size k .

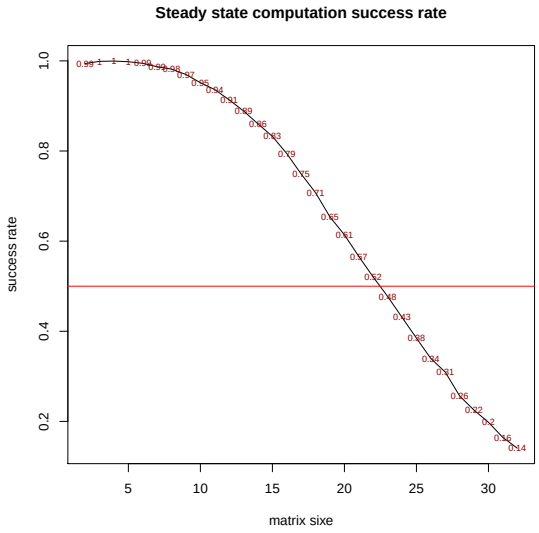


Figure 4: Steady state computation success rate.

The figure shown above displays the success rate observed by TM size. The success rate is higher than 95% for matrices no greater than 10 unit, then it decreases markedly and becomes lower than 50% for matrices bigger then 22. A deeper analysis allowed to identify that the failure reason was due to inaccuracy in the Dirichlet sampling function (row sums numerically different from zero). The TM simulation process was therefore revised normalizing the sum of each row to be numerically equal to one. The experiment was repeated at $2^3, 2^4, \ldots, 2^8$ TM sizes (wider matrices were not tested due to computational timing issues). The observed success rate was always 100% for the sampled TM sizes.

The first example deserves few more words, even if it does not demonstrate any shortcomings in the computational part of the package. Instead, it shows how easy it is to analyze numerically - incorrect TMs as the size of the problems dealt with increases. Various posts have been raised on this topic on the package Github address since the package was published on CRAN.

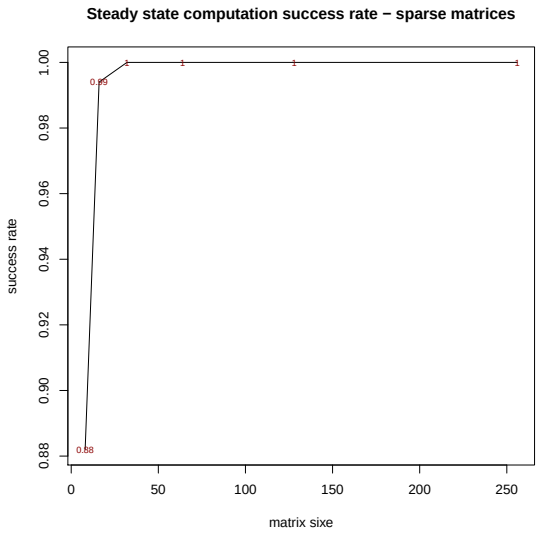


Figure 5: Steady state computation success rate, sparse matrices.

A final test has been performed using TMs with a sparsity factor of 75%. The observed success rate is 100% for matrices wider than 2^5, inexplicably lower (around 90%) for smaller matrices matrices.

The previous examples are clearly far to exhaustively assess the numerical reliability of the implemented algorithms that would require an much deeper analysis and beyond the scope of

the paper. In fact, the numerical reliability is likely to be significantly affected by particular TM structures. Nevertheless they can provide an initial insight about the dimension of the problems that the **markovchain** R package can "safely" handle. The R code used to generate the numerical reliability assessment herewith discussed is available in the "reliability.R" file within the demo folder of `markovchain` R package.

Computational performance

The computation time needed to estimate the TM from input data sequence depends by the size of input data, as the following example displays:

```
#using the rain data sequence
data(rain)
rainSequence<-rain$rain

#choosing different sample size
sizes<-c(10,50,100,250,500,1096)

#timing assessiment
microseconds<-numeric(length(sizes))

for(i in 1:length(sizes)) {

  mydim<-sizes[i]
  mysequence<-rainSequence[1:mydim]
  out<-microbenchmark(
    myFit<-markovchainFit(data=mysequence)
  )
  microseconds[i]<-mean(out$time)

}

plot(sizes, microseconds,type="o",col="steelblue",
xlab="character sequence size",ylab="microseconds",
main="Computational time vs size")
```

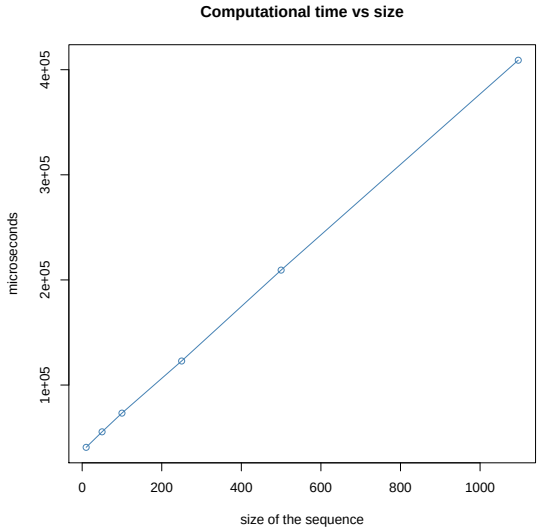


Figure 6: Computation time by size of input data sequence.

The plot shows that the computation time increases linearly with the size of input data sequence, as expected.

The last numeric example presented in the section discussing NHDTMCs shows the computational advantages of rewriting the kernel of core functions using **Rcpp** and **RcppParallel** snippets generated by (Allaire et al., 2016). The `rmarkovchain` function allows the final user to choose whether to use the C++ implementation and a parallel backend, by setting the boolean parameters `useRcpp` and `parallel` respectively.

```
microbenchmark(
rmarkovchain(n=100,object=nhmcFit$estimate,what = "matrix",useRCpp = F),
rmarkovchain(n=100,object=nhmcFit$estimate,what = "matrix",useRCpp = T,parallel = F),
rmarkovchain(n=100,object=nhmcFit$estimate,what = "matrix",useRCpp = T,parallel = T)
)
```

The omitted output of the code snippet shown above demonstrates that the joint use of **Rcpp** and **RcppParallel** fastens the simulations around 10x with respect to the pure R sequential implementation.

Conclusions, discussion and acknowledgements

The **markovchain** package has been designed in order to make common operations on DTMCs as easy as possible for statisticians. The package allows to create, manipulate, import and export DTMCs. Further, the author believes that the current version of the package fully satisfies standard needs such as inference of underlying TM from empirical data, and states classification of a given DTMC.

The author believes that no other R package provides a set of classes, methods, and functions as wide as the one provided in **markovchain**, as of May 2017.

The package's main vignette gives a complete descriptions of its capabilities, including bayesian estimation, statistical tests, classes and methods for continuous time MCs. Also, a separate vignette describes the functions designed to deal with higher order and multivariate MCs, and should still be considered experimental. In fact, such techniques are generally less used than standard DTMCs, and consequently much less literature, applied examples, and coded algorithms are available.

Clearly, an expanded version of the package's capabilities in that area is expected to be released in the future. Current development efforts target optimizing computation speed and reliability, and increasing the analysis capabilities regarding statistical inference. Rewriting core functions using **Rcpp** gave a major boost in terms of computing speed, as exemplified in previous sections. Moreover, the rewriting of the internal core parts of the code affected many functions, such as `markovchainFit` and `markovchainFitList`. Feedbacks provided by the users of the package at https://github.com/spedygiorgio/markovchain/issues have been extremely useful for improving the package. To illustrate, bugs due to numerical issues have been found when analyzing relatively big MCs and have led to revising the `steadyStates` function to be computationally more robust. A known limitation of the package is the lack of a deep assessment of the performance of the package's routines for a relatively large TM. In fact, improving the numerical reliability of the package for large DTMCs is an area on which efforts will be certainly allocated in the near future. At this regard, the implementation of numerical methods methods shown in Stewart (1994) will be explored.

Finally, the package has been available on CRAN since Summer 2013. Notably, it has been granted a funding slot in both 2015, 2016 and 2017 Google Summer of Code (GSOC) editions. In particular, during 2015 GSOC a material part of R code has been ported in **Rcpp** coding, yielding considerable fastening in computational time. The author is extremely grateful to Tae Seung Kang, Sai Bhargav Yalamanchi and Deepak Yadav for their contribution in improving the package. A special thank should be given to the RJournal referees for their constructive comments.

Giorgio Alfredo Spedicato
UnipolSai Assicurazioni
Piazza della Costituzione 2
Bologna 40128, Italy
spedicato_giorgio@yahoo.it

Bibliography

J. Allaire, R. Francois, K. Ushey, G. Vandenbrouck, M. Geelnard, and Intel. *RcppParallel: Parallel Programming Tools for Rcpp*, 2016. URL https://CRAN.R-project.org/package=RcppParallel. R package version 4.3.20. [p102]

Antoine Filipovi, C., Pierucci, Kevin, Zarca, and Isabelle Durand-Zaleski. Markov models for health economic evaluation: The r package heemod. *ArXiv e-prints*, 2017. URL https://pierucci.org/heemod. R package version 0.9.0. [p85]

Artur Araújo, Luís Meira-Machado, and Javier Roca-Pardiñas. **TPmsm**: Estimation of the transition probabilities in 3-state models. *Journal of Statistical Software*, 62(4):1–29, 2014. URL http://www.jstatsoft.org/v62/i04/. [p85]

D. Bates and M. Maechler. *Matrix: Sparse and Dense Matrix Classes and Methods*, 2016. URL http://CRAN.R-project.org/package=Matrix. R package version 1.2-6. [p87]

Bessi, Alessandro. *MCTM: Markov Chains Transition Matrices*, 2015. URL http://CRAN.R-project.org/package=MCTM. R package version 1.0. [p84]

Brémaud, Pierre. *Discrete-Time Markov Models*. Springer-Verlag, 1999. [p90]

C. G. Cassandras. *Discrete Event Systems: Modeling and Performance Analysis*. CRC, 1993. [p85]

Chambers, J.M. *Software for Data Analysis: Programming with R*. Statistics and computing. Springer-Verlag, 2008. ISBN 9780387759357. [p87]

W.-K. Ching, M. K. Ng, and E. S. Fung. Higher-order multivariate markov chains and their applications. *Linear Algebra and its Applications*, 428(2):492–507, 2008. [p84]

B. A. Craig and P. P. Sendi. Estimation of the transition matrix of a discrete-time markov chain. *Health Economics*, 11(1), 2002. [p84]

Csardi, Gabor and Nepusz, Tamas. The **igraph** software package for complex network research. *InterJournal*, Complex Systems:1695, 2006. URL http://igraph.sf.net. [p87]

Eddelbuettel, Dirk. *Seamless R and C++ Integration with* **Rcpp**. Springer-Verlag, New York, 2013. ISBN 978-1-4614-6867-7. [p87]

Efron, B. Bootstrap methods: Another look at the jackknife. *Ann. Statist.*, 7(1):1–26, 1979. URL http://dx.doi.org/10.1214/aos/1176344552. [p96]

Feres, Renaldo. Notes for math 450 matlab listings for markov chains, 2007. URL http://www.math.wustl.edu/~feres/Math450Lect04.pdf. [p90]

V. Goulet, C. Dutang, M. Maechler, D. Firth, M. Shapira, M. Stadelmann, and expm-developers@lists.R-forge.R-project.org. *Expm: Matrix Exponential*, 2015. URL http://CRAN.R-project.org/package=expm. R package version 0.999-0. [p87, 89]

C. M. Grinstead and J. L. Snell. *Introduction to Probability*. American Mathematical Soc., 2012. [p85]

L. Himmelmann. *HMM: HMM - Hidden Markov Models*, 2010. URL https://CRAN.R-project.org/package=HMM. R package version 1.0. [p85]

C. H. Jackson. Multi-state models for panel data: The **msm** package for r. *Journal of Statistical Software*, 38(8):1–29, 2011. URL http://www.jstatsoft.org/v38/i08/. [p85]

J. Jarvis and D. R. Shier. Graph-theoretic analysis of finite markov chains. *Applied mathematical modeling: a multidisciplinary approach*, 1999. [p85]

C. I. Jones. On the evolution of the world income distribution. *Available at SSRN 59412*, 1997. [p84]

J. G. Kemeny, J. L. Snell, and G. L. Thompson. Finite mathematics. *DC Murdoch, Linear Algebra for Undergraduates*, 1974. [p85]

Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, 1999. URL http://ilpubs.stanford.edu:8090/422/. Previous number = SIDL-WP-1999-0120. [p84]

A. A. Markov. Issledovanie zamechatel nogo sluchaya zavisimyh ispytanij. *Izvestiya Akademii Nauk, SPb, VI seriya*, 1(93):61–80, 1907. [p84]

MATLAB. *Version 9.2.0 (R2017a)*. The MathWorks Inc., Natick, Massachusetts, 2017. [p85]

Montgomery, James. Communication classes, 2009. URL http://www.ssc.wisc.edu/~jmontgom/commclasses.pdf. [p90]

Nicholson, William. **DTMCPack***: Suite of Functions Related to Discrete-Time Discrete-State Markov Chains*, 2013. URL http://CRAN.R-project.org/package=DTMCPack. R package version 0.1-2. [p84]

Pablo J. Villacorta and José L. Verdegay. **FuzzyStatProb**: An r package for the estimation of fuzzy stationary probabilities from a sequence of observations of an unknown Markov chain. *Journal of Statistical Software*, 71(8):1–27, 2016. URL https://doi.org/10.18637/jss.v071.i08. [p85]

P. Roebuck. *Matlab: MATLAB Emulation Package*, 2014. URL http://CRAN.R-project.org/package=matlab. R package version 1.0.2. [p87]

M. Scholz. The R package **clickstream**: Analyzing clickstream data with markov chains. *Journal of Statistical Software*, 74(4):1–17, 2016. URL https://doi.org/10.18637/jss.v074.i04. [p84]

Spedicato, Giorgio Alfredo. **markovchain***: An R Package to Easily Handle Discrete Markov Chains*, 2016. R package version 0.6.5. [p84]

W. J. Stewart. *Introduction to the Numerical Solutions of Markov Chains*. Princeton Univ. Press, 1994. [p102]

I. Visser and M. Speekenbrink. **depmixS4**: An r package for hidden markov models. *Journal of Statistical Software*, 36(7):1–21, 2010. URL http://www.jstatsoft.org/v36/i07/. [p85]

Wolfram Research, Inc. *Mathematica*. Wolfram Research, Inc., ninth edition, 2013. [p85]

NA

CRTgeeDR: an R Package for Doubly Robust Generalized Estimating Equations Estimations in Cluster Randomized Trials with Missing Data

by Melanie Prague, Rui Wang, and Victor De Gruttola

Abstract Semi-parametric approaches based on generalized estimating equations (GEE) are widely used to analyze correlated outcomes in longitudinal settings. In this paper, we present a package **CRTgeeDR** developed for cluster randomized trials with missing data (CRTs). For use of inverse probability weighting to adjust for missing data in cluster randomized trials, we show that other software lead to biased estimation for non-independence working correlation structure. **CRTgeeDR** solves this problem. We also extend the ability of existing packages to allow augmented Doubly Robust GEE estimation (DR). Simulation studies demonstrate the consistency of estimators implemented in **CRTgeeDR** compared to packages such as **geepack** and the gains associated with the use of the DR for analyzing a binary outcome using a logistic regression. Finally, we illustrate the method on data from a sanitation CRT in developing countries.

Introduction

We describe the R package **CRTgeeDR**, for estimating coefficients of regression in a marginal mean model. The method is designed to analyze data collected in cluster randomized trials (CRTs) where 1) observations within a cluster may be correlated, 2) observations in separate clusters are independent, 3) a monotone transformation of expectation of the outcome is linearly related to the explanatory variables, and 4) treatment is randomized at a cluster level. The estimation approach generalizes the Generalized Estimating Equation (GEE) (Zeger and Liang, 1986) for fitting marginal generalized linear models to clustered data with possibly informative missingness of the outcome. It combines existing methods for accommodating missing data that use inverse probability weighting (IPW) (Robins et al., 1995) and for increasing precision of estimation by appropriate use of baseline covariates (AUG) (Stephens et al., 2012). We have developed a method for estimating the intervention effect in cluster randomized trials that combines the IPW and the AUG and is doubly robust (DR), meaning that the resulting estimator is consistent if either the model predicting the outcome or the model predicting the missing data is correctly specified—that is, they reflect the true data generation processes (Prague et al., 2016). Below we illustrate the use of the software on a real dataset and clarify its benefits.

The package **CRTgeeDR** not only implements the DR estimator but also the standard GEE, the IPW and the AUG. Regarding IPW, our package differs from most of those currently available in that it avoids the bias that can result from conventional implementation applied to CRTs. Lin et al. (2015) pointed out that implementation of GEE for complete longitudinal data in the current version of SAS (`GENMOD procedure`) requires use of an independence correlation structure if the observation of the outcome at one time point depends on covariates obtained at another time point; this problem had been corrected in the new `GEE procedure` in SAS/STAT 13.2 (SAS Institute Inc., 2015). Tchetgen Tchetgen et al. (2012) made a similar comment regarding the analysis of incomplete longitudinal data in which time-varying covariates and previous outcome values are needed to model the missingness process. This article clarifies this issue for CRTs and proposes an implementation in R that allows for unbiased IPW (and thus DR) estimation with non-independence working correlation structure.

GEE-based approaches for estimating the coefficients in marginal models, in particular the marginal effect of an intervention, have been implemented in only a limited number of R packages and other software for general use. Of note, most of the available software was initially developed to deal with correlated longitudinal data rather than data from CRTs. There are three R packages on CRAN, which will solve GEEs and produce standard errors: whereas **gee** (Carey et al., 2012) and **geepack** (Jun, 2002; Halekoh et al., 2006; Højsgaard and Halekoh, 2016) are computationally demanding, the package **geeM** allows a fast estimation through the use of sparse matrix representation (McDaniel et al., 2013). When interest lies in adjusting for missing outcomes using the IPW, all the packages mentioned above require specification of weights. These weights can be computed using packages such as **ipw** (van der Wal and Geskus, 2011; Geskus and van der Wal, 2015) or directly assigned from a user-defined function. These approaches require the missing data process to be known or correctly specified. Some packages, such as **drgee** (Zetterqvist and Sjölander, 2015), implement doubly robust approaches for uncorrelated data arising from observational studies. These packages provide estimates that are doubly robust in

the sense that the consistency of the parameter estimator from the marginal models is guaranteed if the model linking the outcome to covariates and treatment or the model linking the treatment assignment to covariates correctly reflects the true data generation process. These methods have been extended to deal with missing data with IPW approaches in **CausalGAM** (Glynn and Quinn, 2010a,b), but these packages are intended for analysis of observational studies, not CRTs. Finally, the targeted maximum likelihood estimation (tMLE) method allows estimation of the marginal additive effect of a treatment (van der Laan, 2014a). It is implemented in the packages **tmle** (Gruber, 2014) and **tmlenet** (Sofrygin and van der Laan, 2015) for longitudinal and correlated data. Except for Porter et al. (2011), there has been little published discussion about the differences between GEE-based and tMLE estimation, and we do not delve into a comparison of the two methods. The focus of this article is only on software implementation of the doubly robust GEE for CRTs.

The paper is organized as follows. Section 16.2 introduces the theory of the doubly robust estimator and Section 16.3 describes the features of the **CRTgeeDR** and the estimating function denoted `GeedrEstimation`. Section 16.4 compares the performance of **CRTgeeDR** to **geepack** for the IPW in CRTs and illustrates that the DR is consistent and more efficient than the IPW. Section 16.5 illustrates the analysis of a dataset on sanitation in developing countries (Guiteras et al., 2015a) and illustrates the benefit of using the DR approach compared to standard GEE. Section 16.6 presents a discussion.

IPW in CRTs and doubly robust estimation

Notation

Consider a CRT comprised of n clusters or communities, each with n_i individuals. The cluster sample sizes are assumed fixed and non-informative. Let $\boldsymbol{Y}_i = [Y_{ij}]_{j=1,\ldots,n_i}$ denote the outcome vector for cluster i, some elements of which may be unobserved. Let $R_{ij} = 1$ if Y_{ij} is observed and $R_{ij} = 0$ otherwise. Let $\boldsymbol{X}_{ij} = [X_{ij}^r]_{j=1,\ldots,n_i;r=1,\ldots,p}$ denote the P baseline covariates for subject j in cluster i, which is fully observed. Let A_i be the treatment assigned to cluster i; the indicator for treated condition is $A_i = 1$, and $A_i = 0$ for control condition. We assume that the probability of treatment assignment is known and fixed to $p_A = P(A_i = 1)$. The conditional mean of Y_{ij} is denoted $\mu_{ij} = E(Y_{ij}|X_{ij}, A_i)$, and we let $\boldsymbol{\mu}_i = [\mu_{ij}]_{j=1,\ldots,n_i}$ denote the full vector of means in the i^{th} cluster. We assume that the mean structure of Y_{ij} depends on the covariate vector for subject j in cluster i (Robins et al., 1999), and consider a model for the mean as follow:

$$g(\mu_{ij}) = \boldsymbol{X}_{ij}\boldsymbol{\beta}_X + A_i\beta_A,$$

where $g(.)$ is a monotone differentiable link function and $\boldsymbol{\beta} = (\beta_A, \boldsymbol{\beta}_X)$ is a $(P+1) \times 1$ is a vector of regression coefficients of interest. In this article, we focus on estimation of the marginal effect of an intervention β_A for a binary outcome using the logit link. We assume the variance is $v_{ij} = \text{var}(Y_{ij}|X_{ij}, A_i) = \phi h(\mu_{ij})$, where $h(.)$ is the variance function and ϕ is the dispersion parameter. Thus for our specific example, $v_{ij} = \phi\mu_{ij}(1-\mu_{ij})$ When data are missing at random (MAR), the observation indicator R_{ij} is a function of covariates, treatment condition, and observed outcomes. For CRTs, we assume a restricted version of MAR (rMAR), which requires that R_{ij} cannot be a function of observed outcomes. Although all the theory would hold for classical MAR assumption, it is most of the time difficult to specify the function linking the observation indicator and the observed outcomes of other individuals in the same cluster because there is no ordering. Thus, the probability of being observed π_{ij} for individual j in cluster i, called the propensity score (PS), is: $\pi_{ij}(\boldsymbol{X}_{ij}, A_i, \eta_W) = P(R_{ij} = 1|\boldsymbol{X}_{ij}, A_i)$. The parameters η_W are nuisance parameters and must be estimated.

IPW in CRTs

In presence of rMAR outcome, as in Robins et al. (1995), we estimate $\boldsymbol{\beta}$ by using inverse probability weighted generalized estimating equation (IPW). Therefore, we must include a weight matrix $\boldsymbol{W}_i$ to the usual GEE, that is:

$$\boldsymbol{W}_i(\boldsymbol{X}_{ij}, A_i, \eta_W) = \text{diag}\Big(\frac{R_{ij}}{\pi_{ij}(\boldsymbol{X}_{ij}, A_i, \eta_W)}\Big)_{j=1,\ldots,n_i}.$$

This matrix $\boldsymbol{W}_i(\boldsymbol{X}_{ij}, A_i, \eta_W)$, denoted simply as $\boldsymbol{W}_i$ in the following, adjusts the contribution of each individual in a given cluster by upweighting the contribution of individuals who are less likely to be observed according to their characteristics. Thus, if the propensity score is correctly specified, i.e.,

correspond to the true missingness process, the IPW equation provides consistent estimates:

$$0 = \sum_{i=1}^{n} \boldsymbol{D}_i^\top \boldsymbol{V}_i^{-1} \boldsymbol{W}_i (\boldsymbol{Y}_i - \boldsymbol{\mu}_i), \tag{1}$$

where $D_i = \partial \boldsymbol{\mu}_i / \partial \boldsymbol{\beta}$ is a derivative matrix and $\boldsymbol{V}_i$ is the working covariance matrix for the response $\boldsymbol{Y}_i$. In particular, $\boldsymbol{V}_i = \phi \boldsymbol{F}_i^{1/2} \boldsymbol{C}(\boldsymbol{\alpha}) \boldsymbol{F}_i^{1/2}$, where $\boldsymbol{F}_i^{1/2} = \text{diag}(h(\mu_{ij}))_{j=1,\ldots,n_i}$ and $\boldsymbol{C}(\boldsymbol{\alpha})$ is the working correlation structure with non-diagonal terms $\boldsymbol{\alpha}$. For example, for an independence correlation structure $\boldsymbol{\alpha}$ is zero; for exchangeable structure, all the elements of $\boldsymbol{\alpha}$ are identical. Parameters $\boldsymbol{\alpha}$ could also depend on the treatment assignment $\boldsymbol{C}(\boldsymbol{\alpha}(A_i))$ but we do not consider this possibility in our implementation. In the package **CRTgeeDR**, we estimate the $\boldsymbol{\alpha}$ and ϕ parameters using moment estimators from the Pearson residuals and the Pearson Chi-Square statistic as in **geeM** (McDaniel and Henderson, 2015) also described in McDaniel et al. (2013). In the absence of missing data, $\boldsymbol{W}_i = \boldsymbol{I}$ is set to identity, and the standard GEE is performed by **CRTgeeDR**.

In existing packages such as **geepack**, the Equation 1 is implemented as $0 = \sum_{i=1}^{n} \boldsymbol{D}_i \boldsymbol{V}_i^{-1} (\boldsymbol{Y}_i - \boldsymbol{\mu}_i)$, with $\boldsymbol{V}_i^{-1} = \phi \boldsymbol{F}_i^{1/2} \boldsymbol{W}_i^{1/2} \boldsymbol{C}(\boldsymbol{\alpha}) \boldsymbol{W}_i^{1/2} \boldsymbol{F}_i^{1/2}$ to ensure the fast invertibility of $\boldsymbol{V}_i$. It is easy to verify that when an independence correlation structure is used, $\boldsymbol{C}(\boldsymbol{\alpha}) = I$, and the two implementations are identical. Therefore, one can always use **geepack** with an independence working correlation structure. In contrast, if a non-independence working correlation structure is used, the consistency of IPW estimators do not hold. See the Web-Supplementary Material for a demonstration. Regarding other packages such as **geeM**, although the implementation was the same as in **geepack** up to version 0.8.0, it is now implemented as in Equation 1 in version 0.10.0. In the SAS `GEE` procedure, one can use the option `"type=obslevel"` (in the `missing` statement) in order to use the same implementation as in Equation 1. In general, it is necessary to check the formula used for implementation of the estimating equation in any desired software to avoid confusion.

Augmentation and doubly robust estimation

Recent advances in methods for analysis of data from CRTs have used augmented GEE to improve efficiency of inferences by incorporating baseline covariates (Stephens et al., 2012); we denote this estimator the AUG. They have also been extended to accommodate missing data using an approach based on the IPW which is doubly robust GEE (DR). The DR properties are described in Prague et al. (2016) and the estimating equation is given by :

$$\begin{aligned} 0 = \sum_{i=1}^{M} \Bigg[& \boldsymbol{D}_i^\top \boldsymbol{V}_i^{-1} \boldsymbol{W}_i \Big(\boldsymbol{Y}_i - \boldsymbol{B}_i(\boldsymbol{X}_{ij}, A_i, \boldsymbol{\eta}_B) \Big) \\ & + \sum_{a=0,1} p_A^a (1 - p_A)^{1-a} \boldsymbol{D}_i^\top \boldsymbol{V}_i^{-1} \Big(\boldsymbol{B}_i(\boldsymbol{X}_{ij}, A_i = a, \boldsymbol{\eta}_B) - \boldsymbol{\mu}_i(\boldsymbol{\beta}, A_i = a) \Big) \Bigg] \\ = \boldsymbol{\Phi}(\boldsymbol{Y}_i, \boldsymbol{R}_i, A_i, \boldsymbol{X}_{ij}, \boldsymbol{\beta}, \boldsymbol{\eta}_W, \boldsymbol{\eta}_B). & \end{aligned} \tag{2}$$

Each element of the vector $\boldsymbol{B}_i(\boldsymbol{X}_i, A_i = a, \boldsymbol{\eta}_B) = [B_{ij}(\boldsymbol{X}_i, A_i = a, \boldsymbol{\eta}_B)]_{j=1,\ldots,n_i}$ is an arbitrary function linking Y_{ij} with $\boldsymbol{X}_{ij}$ for each treatment arm, which we refer to as the outcome model (OM) The $\boldsymbol{\eta}_B$ are nuisance parameters. The estimator in Equation 2 is most efficient if $B_{ij}(\boldsymbol{X}_i, A_i = a, \boldsymbol{\eta}_B) = E(Y_{ij} | \boldsymbol{X}_{ij}, A_i = a)$ (Zhang et al., 2008), that is, the OM is correctly specified. If the OM is not correctly specified, i.e., does not correspond to the true data generation process, the estimation remains consistent provided that the PS model is correctly specified, but one may have a loss in efficiency. Without missing data, $\boldsymbol{W}_i = \boldsymbol{I}$ is set to identity, and the AUG is performed by **CRTgeeDR**.

Without missing data or with data missing completely at random, the use of augmentation may allow a gain in efficiency by incorporating information on baseline covariates. The PS should not be used because it will be misspecified and therefore may lead to an increase of the variance of the estimates. In presence of rMAR data, IPW alone can be used but DR should be preferred in order to increase the chances to have an unbiased estimator. Finally, as mentioned above, for data missing not at random, none of the methods implemented in **CRTgeeDR** are adequate.

The R package CRTgeeDR

The main function for estimation in the package `CRTgeeDR`

The call function for performing estimation is `geeDREstimation`:

```
R> geeDREstimation(formula, id, data = parent.frame(), family = gaussian,
+    corstr = "independence", Mv = 1, corr.mat = NULL, init.beta = NULL,
+    init.alpha = NULL, init.phi = 1, scale.fix = FALSE, maxit = 20,
+    tol=1e-05, print.log = FALSE, nameTRT = "TRT", nameMISS = "MISSING",
+    nameY = "OUTCOME", sandwich = TRUE, sandwich.nuisance = FALSE,
+    fay.adjustment = FALSE, fay.bound = 0.75, aug = NULL, pi.a = 1/2,
+    model.augmentation.trt = NULL, model.augmentation.ctrl = NULL,
+    stepwise.augmentation = FALSE, weights = NULL, typeweights = "VW",
+    model.weights = NULL, stepwise.weights = FALSE)
```

The marginal model, to be estimated on the R dataframe `data`, is given in `formula`. The link function, g, depends on the nature of the outcome, which is specified in the argument `family`. The name of the outcome `nameY`, the clustering variable `id`, the binary treatment `nameTRT` (with the convention 1 is treated and 0 is control), and the missing indicator `nameMISS` must be specified if they differ from default values. The algorithm iterates between the estimation the working correlation structure and regression parameters with a stopping rule based on stabilisation of estimates (tolerance can be set by the user; default is `tol`$= 10^{-5}$ or `max.iter=20`). Depending on the specification or not of the PS and the OM, `geeDREstimation` allows the implementation of standard GEE, the IPW, the AUG and the DR approaches. The algorithm is defined as follow:

1. *Determine the PS:* $\pi_{ij}(\boldsymbol{X}_{ij}, A_i, \eta_W) = P(R_{ij}|\boldsymbol{X}_{ij}, A_i)$, π_{ij} for short. Either the π_{ij} are known from prior analysis or by design and the weights can be specified directly in the `weights` argument. Alternatively one can compute the PS by fitting a logistic regression of R_{ij} on $(\boldsymbol{X}_{ij}, A_i)$. In this case, the PS regression formula can be directly entered in `model.weights`. A `glm` with logit link function is internally processed with or without variable selection, depending on the value of the `stepwise.weights` argument. If all of the above are set to NULL or default, no IPW adjustment will be made—GEE or AUG will be used. Finally, if despite our concern about the implementation of weights, one wants to use the same implementation as in packages **geepack** or `proc GENMOD` in SAS, then one can set `typeweights="GENMOD"`.
2. *Determine group-specific OM:* $B_{ij}(X_{ij}, A_i = a) = E\left[\boldsymbol{Y}_{ij}|A_i = a, X_{ij}\right]$. When the $\boldsymbol{B}_i$ are known from prior analysis, they can be directly entered in `aug=c(ctrl=`$B_{ij}(X_{ij}, A_i = 0)$`, trt=`$B_{ij}(X_{ij}, A_i = 1)$`)`. Alternatively, we can regress Y_{ij} on $\boldsymbol{X}_{ij}$ within each treatment group. In this case, the OM regression formulas can be directly entered in `model.augmentation.trt` and `model.augmentation.ctrl`. A `glm` is then internally processed with or without variable selection depending on the value of the argument `stepwise.augmentation`. If all of the above are set to NULL or default, no augmentation adjustment will be made—GEE or IPW will be used. The probability of treatment assignment, which is known in CRTs, must be specified in the argument `pi.a`. Of note for steps 1 and 2, when using the stepwise option to compute the OM or the PS, one runs the risk of overfitting (van der Laan, 2014b). Avoiding this is possible by sparsely including only relevant variables in the selection and also by running a bootstrap diagnostic using outputs (`ps.model`, `om.model.trt` and `om.model.ctrl`). The underlying assumption is that the true OM or PS are selected at the end of the stepwise selection and then held fixed in the estimating equation in further steps.
3. *Determine the working correlation structure.* Available structures are `independence`, `exchangeable`, `M-dependent` (using `Mv`), `unstructured`, or `user-defined` (using `corr.mat`). Using the `scale.fix` argument, the dispersion parameter ϕ can be either estimated or held fixed to a specified value.
4. *Obtain initial values.* They are either specified by the user (`init.beta`, `init.alpha`, and `init.phi`) or internally defined by fitting a glm under independence to obtain initial values for $\hat{\boldsymbol{\beta}}^{(0)}$ and by setting $\phi^{(0)} = 1$ and $\boldsymbol{\alpha}^{(0)} = 0$.
5. *Enter/continue the iterative procedure* :
   (a) Use the fit from $\hat{\boldsymbol{\beta}}^{(n)}$ to compute Pearson residuals. Use Pearson residuals based formulas to compute the scale parameter ($\phi^{(n+1)}$, except if `scale.fix=TRUE`) and the parameters in the working correlation matrix ($\boldsymbol{\alpha}^{(n+1)}$).
   (b) Construct the augmented equation given in Equation 2 and solve it numerically using Newton-Raphson algorithm for $\hat{\boldsymbol{\beta}}^{(n+1)}$:

$$\hat{\boldsymbol{\beta}}^{(n+1)} = \hat{\boldsymbol{\beta}}^{(n)} - \left[\frac{\partial \boldsymbol{\Phi}(\boldsymbol{Y}_i, \boldsymbol{R}_i, A_i, \boldsymbol{X}_{ij}, \boldsymbol{\beta}, \eta_W, \boldsymbol{\eta}_B)}{\partial \boldsymbol{\beta}}\right]^{-1}_{\hat{\boldsymbol{\beta}}^{(n)}} \boldsymbol{\Phi}(\boldsymbol{Y}_i, \boldsymbol{R}_i, A_i, \boldsymbol{X}_{ij}, \hat{\boldsymbol{\beta}}^{(n)}, \eta_W, \boldsymbol{\eta}_B).$$

   (c) If $\max\left|\frac{\hat{\beta}^{(n+1)} - \hat{\beta}^{(n)}}{\hat{\beta}^{(n)} + prec.machine}\right| >$ `tol` and $n + 1 \leq$ `max.iter` go back to 5 else go to 6, where $prec.machine \sim 10^{-16}$.

6. Compute the requested variances of $\hat{\beta}^{(n+1)}$. If, `sandwich` and `sandwich.nuisance` are set to TRUE, classical and nuisance-adjusted (for the estimation of parameters η_W in the PS and η_B in the OM) sandwich estimators of the variance are provided, see Prague et al. (2016) for their definition. The nuisance-adjusted version is computed using numerical derivatives of score equations for PS, OM and estimating equations jointly, which are obtained by using the `jacobian` function of the package **numDeriv** (Gilbert and Varadhan, 2015); this is recommended if the AUG, the IPW or the DR estimator are considered. Finally, a small-sample-adjusted sandwich estimator of the variance can also be computed using Fay's adjustment (Fay and Graubard, 2001) setting the argument `fay.adjustment` to TRUE. Its implementation is derived from the function `gee.var.fg` in the package **geesmv** (Wang, 2015).

Adequacy of the PS and the OM to data

Consistency and efficiency of the DR estimator depend on the correct specification of the PS and the OM, see Prague et al. (2016) for theoretical demonstrations. The user may want to check the adequacy of the selected OM model to the data by using the function `getOMPlot`, which provides plots to check the `glm` model assumption. The "Residuals vs. Fitted" and the "Scale-location" graphics allow verification of the homogeneity of the variance and the adequacy of the link function. The "Normal Q-Q" checks for the normal distribution of the residuals. The "Residuals vs Leverage" plot allows detection of points that have high leverage on the regression coefficients and that should be investigated as outliers. In the same spirit, the "Cook's distance" and the "Cook's distance vs leverage" provide measures of the effect of deleting a given observation. Of note, these graphs are only interpretable for a continuous outcome. In addition, for the PS model the function `getPSPlot` provides a histogram of the weights. If weights are too large then the IPW and DR approaches are likely to be unstable. In this case, the user should compute weights externally using, for example, stabilized weights with the associated package **ipw** (van der Wal and Geskus, 2011) or other approaches such as described in Wang and Paik (2011). Finally, the user can access the `glm` objects created during the PS and OM initial steps as objects named `ps.model`, `om.model.trt`, and `om.model.ctrl` from the main function `geeDREstimation`.

Simulations

The properties of DR to accommodate complex correlation structure, rMAR outcomes, and the presence of imbalance in baseline covariates have already been demonstrated in Prague et al. (2016). In this article, we focus on the superiority of implementation of weights in the package **CRTgeeDR** compared to package **geepack**. We focus on a simple example to illustrate that, even in very simple cases, estimators implemented in broadly used R package **geepack** for IPW can be inconsistent when using an exchangeable working correlation structure. This is the case when $\boldsymbol{V}_i^{-1} = \phi \boldsymbol{F}_i^{1/2} \boldsymbol{W}_i^{1/2} \boldsymbol{C}(\boldsymbol{\alpha}) \boldsymbol{W}_i^{1/2} \boldsymbol{F}_i^{1/2}$ is used in the estimating equation. We simulate data from a CRT with 100 communities of 90, 100, or 110 individuals with probability 1/3 for each. The treatment A is randomly assigned with probability $p_A = 1/2$. One covariate is of interest: $X_{ij} \sim \mathcal{N}(2,1)$. We simulate correlated outcome with exchangeable structure, and correlation between individuals is set to 0.05. This is done by using a cluster-level bridge distribution $b_i \sim \mathcal{B}(0.05)$. Data generation process is as follow:

$$\begin{array}{rcl} \text{logit}[P(Y_{ij}=1|A_i,X_{ij})] & = & -0.5+0.3A_i+0.4X_{ij}+0.4X_{ij}A_i+b_i, \\ \text{logit}[P(R_{ij}=1|A_i,X_{ij})] & = & 4.0-0.3A_i-0.8X_{ij}-0.8X_{ij}A_i. \end{array} \tag{3}$$

We simulated R=10,000 replicates. The observed average proportion of missing observations is around 25% and the observed average intraclass correlation is 0.08. Missingness is associated strongly with individual covariates and, therefore, the weights differ between individuals in the same cluster. The true value of the odds-ratio for the marginal effect of treatment is computed for each dataset k without missing data by obtaining the counterfactual values with and without treatment under this model:

$$\text{OR}_k = \frac{E(Y_{ij}=1|A_i=1)/E(Y_{ij}=0|A_i=1)}{E(Y_{ij}=1|A=0)/E(Y_{ij}=0|A_i=0)}.$$

The true OR is given by $\frac{1}{R}\sum_{k=1}^{R} OR_k$=2.56 with associated parameter for marginal intervention effect in the marginal regression $\beta_A = 0.941$. For each dataset, we first ran the analysis on the dataset without missing data for the standard GEE and the AUG using **CRTgeeDR**. Then we ran the analysis on the dataset with missing data for the IPW using **geepack** and

for the standard GEE, the IPW, the AUG, and the DR using **CRTgeeDR**. Two types of DR are presented here: DR1 is the estimator using the correct models for the OM and the PS, and DR2 omits treatment-covariate interaction terms in the PS. The models for the PS and OM for analysis are described in the Table 1. Table 1 shows the bias, empirical standard error, sandwich standard error, and coverages for each analysis using independence (-I) and exchangeable (-E) working correlation structure. The code to replicate this study is available in Web-Supplementary Material.

	Independence (-I)				Exchangeable (-E)			
Method	Bias	Emp. SE	SE	Cov.	Bias	Emp. SE	SE	Cov.
No missing data:								
GEE **CRTgeeDR**	0.002	0.102	0.099	94.3	0.002	0.108	0.099	93.2
GEE **geepack**	0.003	0.102	0.101	94.6	0.003	0.102	0.101	94.6
AUG **CRTgeeDR**	0.002	0.101	0.099	94.3	0.002	0.109	0.114	95.8
With missing data:								
GEE **CRTgeeDR**	-0.257	0.103	0.177	82.0	-0.256	0.104	0.081	18.1
AUG **CRTgeeDR**	0.249	0.092	0.109	35.7	0.307	0.115	0.139	37.1
With missing data and adjustment for it:								
IPW **CRTgeeDR**	0.003	0.108	0.106	95.0	0.003	0.118	0.110	93.7
IPW **geepack**	0.008	0.107	0.104	94.8	0.582	0.577	0.357	19.4
DR1 **CRTgeeDR**	0.003	0.107	0.104	94.5	0.004	0.120	0.125	96.1
DR2 **CRTgeeDR**	0.003	0.105	0.102	94.4	0.004	0.118	0.123	96.0

Marginal mean model:
$\text{logit}(\mu_{ij}) = \beta_0 + \beta_A A_i$.
PS used for IPW and DR (true):
$\text{logit}(P(R_{ij} = 1|A_i, X_{ij})) = \gamma_0 + \gamma_A A_i + \gamma X_{ij} + \gamma_I X_{ij} A_i$.
PS used for DR2 (omitting interactions in PS):
$\text{logit}(P(R_{ij} = 1|A_i, X_{ij})) = \gamma_0 + \gamma_A A_i + \gamma X_{ij}$.
OM used for AUG, DR1 and DR2 (fitted for each group a):
$\text{logit}(P(Y_{ij} = 1|A_i = a, X_{ij})) = \xi + \xi_A A_i + \xi X_{ij}$.

Table 1: Comparison of the standard GEE, the IPW, the AUG and the DR analysis with the packages **CRTgeeDR**, **geepack**, and **geeM** using independence and exchangeable working correlation structure. True value for the parameter β_A is 0.91 (OR=2.56). The bias, the empirical and the estimated standard errors (SE), and the coverages for parameter $\widehat{\beta_A}$ are computed over 10,000 replicates. The true data generation process for outcome and missingness is provided in Equation 3. The PS and OM models for analysis are correctly specified and given in the footnote of the table.

The results for standard GEE are unbiased in the absence of missing data (<0.003 for GEE-I and GEE-E with all packages) and biased in presence of rMAR outcomes reflecting the fact that the missingness is informative. Using the IPW-I corrects for this bias (0.008 for **geepack**). All packages give a similar estimated standard error leading to acceptable coverage close to their nominal value of 95%. When using an exchangeable correlation structure, the coverage (93.7%) remains close to the nominal value for IPW-E using **CRTgeeDR**, but it drops to 19.4% using **geepack**. This is mainly driven by an increase in the bias from 0.003 for **CRTgeeDR** to 0.582 for **geepack** for IPW-E. Using the DR1 version of **CRTgeeDR** provides consistent estimates (bias $\leq$0.004 for DR1-I and DR1-E). DR-1 yields coverage that is close to or greater than 95% and gains, on average, in efficiency. For example, the empirical standard error is 0.108 for IPW-I and 0.107 for DR-I. DR2, which omits the term $X_{ij}A_i$ in the PS, yields consistent and efficient estimates even when the treatment-covariate interactions are not explicitly specified in the PS. As demonstrated in Prague et al. (2016), DR1 and DR2 have similar properties.

Illustration on the sanitation data

In this section, we present a step-by-step analysis of data from a CRT to investigate the efficacy of alternatives policies on the investment in hygienic latrines in developing coun-

tries. A total of 380 communities in rural Bangladesh were assigned to different marketing interventions—community motivation, subsidies, supply side-market, a combination of the three, and a control group. Results of this study were published in (Guiteras et al., 2015a). All the code and data associated with this study are available on dataverse, see url in Guiteras et al. (2015b).

	Side-Market supply		Control		All	
Cluster structure						
M	36 (n = 1651)		66 (n = 3186)		100 (n = 4837)	
N_i	49 (15)		48 (16)		48 (16)	
Outcome Y_{ij}	Mean	Missing %	Mean	Missing %	Mean	Missing %
Hygienic Latrine Ownership	34.8%	4.2%	30.3%	3.1%	31.8%	3.5%
Individual-level $\boldsymbol{X}_{ij}^{\text{IND}}$	Mean	Missing %	Mean	Missing %	Mean	Missing %
Report diarrhea	4.3%	0%	4.8%	0%	4.6%	0%
Male	91.1%	<0.01%	90.0%	<0.01%	90.1%	<0.01%
Education	49.2%	0%	45.8%	0%	46.9%	0%
Muslim	83.2%	<0.01%	86.3%	<0.01%	85.2%	<0.01%
Bengali	85.6%	<0.01%	88.5%	<0.01%	87.6%	<0.01%
Agricultor	75.0%	<0.01%	70.2%	<0.01%	71.9%	<0.01%
Stoves	58.2%	<0.01%	62.9%	<0.01%	61.3%	<0.01%
Water Pipes	89.9%	<0.01%	91.3%	<0.01%	90.8%	<0.01%
Phone	64.1%	<0.01%	57.2%	<0.01%	59.5%	<0.01%
Age	39 (13)	<0.01%	39 (14)	<0.01%	39 (14)	<0.01%
Cluster-level $\boldsymbol{X}_{ij}^{\text{C}}$	Mean	Missing %	Mean	Missing %	Mean	Missing %
Village size	230 (120)	0%	270 (190)	0%	256 (170)	0%
Nb doctors	7 (7)	0%	9 (18)	0%	8 (15)	0%
% Landless	41.6 (12)	0%	34.4 (15)	0%	36.9 (15)	0%
% Almost Landless	19.3 (11)	0%	24.0 (8)	0%	22.4 (9)	0%
% Access electricity	59.9 (26)	0%	59.1 (20)	0%	59.4 (22)	0%

Table 2: Description of the Sanitation dataset from (Guiteras et al., 2015a) considering only the Side-Market supply and the Control group. Percentages are given for qualitative covariates. Means and standard deviations in parentheses are provided for continuous covariates.

We consider only the comparison of a supply side-market versus control. The published analysis used a mixed effect model and showed that the supply side-market alone did not increase the hygienic latrine ownership (+0.3 percentage points, p-value=0.90). We reanalyze the dataset using the GEE approaches in order to get the marginal effect of intervention. Description of the outcome and variables for adjustment are available in Table 2. Because covariates were missing in less than 0.01% of the observations, we assume that covariates are missing completely at random and exclude individuals with missing covariates. The final dataset contains 4774 individuals and 380 clusters. We assume the outcomes are rMAR. As there is some evidence of imbalance in baseline covariates across arms, i.e., the descriptive distributions of covariates in Table 2 are different between treated and control groups, we use the DR approach. We assume that the correlation between any pair of individuals in the same cluster is the same and hence use an exchangeable working correlation structure. In this example, the PS and OM are fitted using a logistic regression with a linear combination of all the individual-level and cluster-level covariates described in Table 2. Variables for these models are selected using a forward stepwise regression before solving the estimating equation. Adequacy of the model has been verified. The code for analysis is available in the Web-Supplementary Material. To illustrate the use of the package **CRTgeeDR**, we provide instructions for the DR estimator:

```
R> DR <- geeDREstimation(OUTCOME ~ TRT, id = CLUSTER, data = Sanitation,
+    family = binomial("logit"), corstr = "exchangeable", typeweights = "VW",
+    model.weights = MISSING ~ TRT + DIARRHEA + ... + ELEC_ACCESS,
+    model.augmentation.trt = OUTCOME ~ DIARRHEA + ... + ELEC_ACCESS,
+    model.augmentation.ctrl = OUTCOME ~ DIARRHEA + ... + ELEC_ACCESS,
```

```
+    stepwise.weights = TRUE, stepwise.augmentation = TRUE)
R> summary(DR)
```

The output displays statistics for estimated coefficients $\boldsymbol{\beta}$, $\boldsymbol{\alpha}$ and ϕ, the number of Newton-Raphson iterations before convergence, and some description of the size of the clusters.

```
            Estimates Model SE Robust SE   wald        p
(Intercept)   -0.8106  0.09396    0.1088 -7.452 0.000000
TRT            0.4365  0.12890    0.1425  3.062 0.002198

 Est. Correlation:  0.07306
 Correlation Structure:  exchangeable
 Est. Scale Parameter:  0.9955

 Number of GEE iterations: 2
 Number of Clusters:  100    Maximum Cluster Size:  87
 Number of observations with nonzero weight:  4612
```

Table 3 presents the PS and OM for analysis, the estimates, the nuisance-adjusted sandwich estimates of the variance, the confidence intervals for the odd-ratios, the p-values, and the computation times for each of these analysis. For DR the computation time is 20 seconds, most of which is required for the computation of the nuisance-adjusted sandwich estimator of the variance (the estimation is < 3 seconds otherwise). Whereas GEE and IPW lead to non-significant effect of supply side-market, the DR estimates are significantly different from 0 at the 0.05 level (p=0.025). Using the DR, we conclude that there is 55% [8% - 121%] greater chance of owning hygienic latrine after one year if there is a supply side-market. This effect is significant (p<0.05) even using a nuisance-adjusted SE, which is generally larger than the standard sandwich SE due to incorporation of additional variability from estimation of the nuisance parameters in the PS and the OM (η_W and η_B). Information about the PS and the OM can be obtained by using the following commands:

```
R>  summary(DR$ps.model)
R>  summary(DR$om.model.trt)
R>  summary(DR$om.model.ctrl)
R>  getPSPlot(DR)
```

		Sandwich	Nuis-adj.	$exp(\beta_A)$			p-value		time
	β_A	SE	SE	OR	IC_{min}	IC_{max}	Unadj.	Nuis-adj.	(sec.)
GEE	0.19	0.171	-	1.21	0.87	1.69	0.262	-	1
IPW	0.19	0.182	0.219	1.21	0.79	1.86	0.290	0.386	32
AUG	0.45	0.141	0.176	1.57	1.12	2.22	0.001	0.010	11
DR	0.44	0.143	0.183	1.55	1.08	2.21	0.002	0.016	20

Marginal mean model: $logit(\mu_{ij}) = \beta_0 + \beta_A A_i$.
PS: $\text{logit}(P(R_{ij}|A_i, \boldsymbol{X}_{ij}^{\text{IND}}, \boldsymbol{X}_{ij}^{\text{C}}) = \gamma_0 + \gamma_A A_i + \sum_{k=1}^{10} \gamma_k^{\text{IND}} X_{ijk}^{\text{IND}} + \sum_{k=1}^{5} \gamma_k^{\text{C}} X_{ijk}^{\text{C}}$.
OM: $\text{logit}(P(Y_{ij}|A_i = a, \boldsymbol{X}_{ij}^{\text{IND}}, \boldsymbol{X}_{ij}^{\text{C}}) = \xi_0 + \sum_{k=1}^{10} \xi_k^{a\text{IND}} X_{ijk}^{\text{IND}} + \sum_{k=1}^{5} \xi_k^{a\text{C}} X_{ijk}^{\text{C}}$, for each group a.

Table 3: Effects of the supply side-market vs. control on the probability of hygienic latrine ownership in the sanitation data analysis (Guiteras et al., 2015a) using the standard GEE, the IPW adjustment (IPW and DR), and the augmentation for imbalance (AUG and DR) assuming outcomes are rMAR.

Description of models for OM, PS and histogram of weights are given in the Web-Supplementary Material Table 1 and Figure 1. As noted in Table 3, the estimates for IPW are close to those for GEE, reflecting the fact that only 3.5% of data are missing. We also note that all of the non-null weights are close to 1 (1.035 [1.02; 1.04]) showing that no covariate of the PS explains the missingness pattern. Thus, the increased significance of the intervention in the DR analysis compared to GEE is mainly driven by the augmentation. In both groups, households with higher education and economic status (as evidenced by stoves, water

pipes, phones, and other factors) are more likely to have a hygienic latrine. For cluster-level covariates the patterns differ by intervention group: a high number of doctors is positively associated with the hygienic latrine ownership only in the intervention group indicating a potential synergy between the number of doctors and the presence of side-supply markets.

Conclusion

We demonstrated that the IPW can be biased in CRTs if the weights are not implemented as described in Robins et al. (1995) and a non-independence working correlation structure is chosen. In particular, we discuss problems that arise in the package **geepack** implemented in R. These concerns apply not only for outcome data in CRTs but also to longitudinal outcome data, when the probability that an observations is missing at a given time depends on time-varying covariates measured at other times. We recommend to always check the implementation in the software that has been chosen for analysis. The **CRTgeeDR** package protects against this bias and allows for adjustment in imbalance in baseline covariates in CRTs. The package can accommodate a wide range of outcome types, link functions, and working correlation structures. The **CRTgeeDR** package is easy to use and does not require extensive programming. It therefore makes the augmented GEE (AUG) and the Doubly robust (DR) methodology for CRTs more accessible to applied researchers. Of note, although the **CRTgeeDR** package had been designed for CRTs, it can also be used for analysis of correlated longitudinal data from a randomized trial. The use of version 2.0 of the **CRTgeeDR** package to analyze observational clustered data (in which treatment attribution may be informative) is not straightforward, but updates with these capabilities are under development.

Acknowledgement

We thank R. Guiteras for sharing the Sanitation study on the dataverse website. This work was founded by NIH grants R37 AI 51164 and R01 MH100974. Portions of this research were conducted on the Cluster at Harvard Medical (NIH grant NCRR 1S10RR028832-01).

Bibliography

V. J. Carey, T. Lumley, and B. Ripley. *gee: Generalized Estimation Equation Solver*, 2012. URL http://CRAN.R-project.org/package=gee. R package version 4.13-19. [p105]

M. P. Fay and B. I. Graubard. Small-sample adjustments for Wald-type tests using sandwich estimators. *Biometrics*, 57(4):1198–1206, 2001. [p109]

R. B. Geskus and W. M. van der Wal. *ipw: Estimate Inverse Probability Weights*, 2015. URL http://CRAN.R-project.org/package=ipw. R package version 1.0-11. [p105]

P. Gilbert and R. Varadhan. *numDeriv: Accurate Numerical Derivatives*, 2015. URL http://CRAN.R-project.org/package=numDeriv. R package version 2014.2.1. [p109]

A. Glynn and K. Quinn. *CausalGAM: Estimation of Causal Effects with Generalized Additive Models*, 2010a. URL http://CRAN.R-project.org/package=CausalGAM. R package version 0.1-3. [p106]

A. N. Glynn and K. M. Quinn. An introduction to the augmented inverse propensity weighted estimator. *Political Analysis*, 18(1):36–56, 2010b. [p106]

S. Gruber. *tmle: Targeted Maximum Likelihood Estimation*, 2014. URL http://CRAN.R-project.org/package=tmle. R package version 1.2.0-4. [p106]

R. Guiteras, J. Levinsohn, and A. M. Mobarak. Encouraging sanitation investment in the developing world: a cluster-randomized trial. *Science*, 348(6237):903–906, 2015a. [p106, 111, 112]

R. Guiteras, J. Levinsohn, and M. Mobarak. Encouraging sanitation investment in the developing world: A cluster-randomized trial. *Harvard Dataverse; online data*, 2015b. doi: doi/10.7910/DVN/GJDUTV. URL http://dx.doi.org/10.7910/DVN/GJDUTV. [p111]

U. Halekoh, S. Højsgaard, and J. Yan. The R package geepack for generalized estimating equations. *Journal of Statistical Software*, 15(2):1–11, 2006. [p105]

S. Højsgaard and R. Halekoh. *geepack: Generalized Estimating Equations Package*, 2016. URL http://CRAN.R-project.org/package=geepack. R package version 1.2-0.1. [p105]

Y. Jun. geepack: Yet another R package for generalized estimating equations. *R-News*, 2(3): 12–14, 2002. [p105]

G. Lin, R. Rodriguez, and I. I. SAS. Weighted methods for analyzing missing data with the GEE Procedure. *Proceedings of SAS Global Forum*, Washington DC(2014 March 23th-26th): paper 166, 2015. [p105]

L. S. McDaniel and N. Henderson. *geeM: Solve Generalized Estimating Equations*, 2015. URL http://CRAN.R-project.org/package=geeM. R package version 0.7.4. [p107]

L. S. McDaniel, N. C. Henderson, and P. J. Rathouz. Fast pure R implementation of gee: Application of the Matrix package. *The R journal*, 5(1):181, 2013. [p105, 107]

K. E. Porter, S. Gruber, M. J. van der Laan, and J. S. Sekhon. The relative performance of targeted maximum likelihood estimators. *The International Journal of Biostatistics*, 7(1):1–34, 2011. [p106]

M. Prague, R. Wang, A. Stephens, E. Tchetgen Tchetgen, and V. De gruttola. Accounting for interactions and complex inter-subject dependency for estimating treatment effect in cluster randomized trials with missing at random outcomes. *Biometrics*, 72(4):1066–1077, 2016. [p105, 107, 109, 110]

J. M. Robins, A. Rotnitzky, and L. P. Zhao. Analysis of semiparametric regression models for repeated outcomes in the presence of missing data. *Journal of the American Statistical Association*, 90(429):106–121, 1995. [p105, 106, 113]

J. M. Robins, S. Greenland, and F.-C. Hu. Estimation of the causal effect of a time-varying exposure on the marginal mean of a repeated binary outcome. *Journal of the American Statistical Association*, 94(447):687–700, 1999. [p106]

SAS Institute Inc. *SAS/STAT Software, Version 13.2*. Cary, NC, 2015. URL http://www.sas.com/. [p105]

O. Sofrygin and M. van der Laan. *tmlenet: Targeted Maximum Likelihood Estimation for Network Data*, 2015. URL http://CRAN.R-project.org/package=tmlenet. R package version 0.1-0. [p106]

A. J. Stephens, E. J. Tchetgen Tchetgen, and V. DeGruttola. Augmented generalized estimating equations for improving efficiency and validity of estimation in cluster randomized trials by leveraging cluster-level and individual-level covariates. *Statistics in medicine*, 31 (10):915–930, 2012. [p105, 107]

E. J. Tchetgen Tchetgen, M. M. Glymour, J. Weuve, and J. Robins. A cautionary note on specification of the correlation structure in inverse-probability-weighted estimation for repeated measures. *Epidemiology*, 23(4):644–646, 2012. [p105]

M. van der Laan. Causal inference for a population of causally connected units. *Journal Causal Inference*, 2(1):1374–1380, 2014a. [p106]

M. van der Laan. Targeted estimation of nuisance parameters to obtain valid statistical inference. *The International Journal of Biostatistics*, 10(1):29–57, 2014b. [p108]

W. M. van der Wal and R. B. Geskus. ipw: an R package for inverse probability weighting. *Journal of Statistical Software*, 43(13):1–23, 2011. [p105, 109]

C. Wang and M. C. Paik. A weighting approach for gee analysis with missing data. *Communications in Statistics-Theory and Methods*, 40(13):2397–2411, 2011. [p109]

M. Wang. *geesmv: Modified Variance Estimators for Generalized Estimating Equations*, 2015. URL http://CRAN.R-project.org/package=geesmv. R package version 1.3. [p109]

S. L. Zeger and K.-Y. Liang. Longitudinal data analysis for discrete and continuous outcomes. *Biometrics*, 42(1):121–130, 1986. [p105]

J. Zetterqvist and A. Sjölander. *drgee: Doubly Robust Generalized Estimating Equations*, 2015. URL http://CRAN.R-project.org/package=drgee. R package version 1.1.3. [p105]

M. Zhang, A. A. Tsiatis, and M. Davidian. Improving efficiency of inferences in randomized clinical trials using auxiliary covariates. *Biometrics*, 64(3):707–715, 2008. [p107]

Melanie Prague
Department of Biostatistics
Harvard T.H. Chan School of Public Health
655 Huntington Ave
Boston, MA 02115
and
INRIA - INSERM U1219 - SISTM
164 rue Leo Saignat Room 23
33076 Bordeaux Cedex, France
(ORCiD:0000-0001-9809-7848)
melanie.prague@inria.fr

Rui Wang
Department of Biostatistics
Harvard T.H. Chan School of Public Health
655 Huntington Ave
Boston, MA 02115
(ORCiD:0000-0001-5007-193X)
rwang@hsph.harvard.edu

Victor De Gruttola
Department of Biostatistics
Harvard T.H. Chan School of Public Health
655 Huntington Ave
Boston, MA 02115
degrut@hsph.harvard.edu

queueing: A Package For Analysis Of Queueing Networks and Models in R

by Pedro Cañadilla Jiménez, Yolanda Román Montoya

Abstract queueing is a package that solves and provides the main performance measures for both basic Markovian queueing models and single and multiclass product-form queueing networks. It can be used both in education and for professional purposes. It provides an intuitive, straightforward way to build queueing models using S3 methods. The package solves Markovian models of the form M/M/c/K/M/FCFS, open and closed single class Jackson networks, open and closed multiclass networks and mixed networks. Markovian models are used when both the customer inter-arrival time and the server processing time are exponentially distributed. Queueing network solvers are useful for modelling situations in which more than one station must be visited.

Introduction

Queueing theory is a mathematical branch of operations research. A pioneering work in this field was *The Theory of Probabilities and Telephone Conversations* by *A. K. Erlang* (Erlang, Agner K., 1909).

Queueing exists when the demand for a service exceeds the available supply (Donald Gross and Carl M. Harris, 1974). Although the common image of queues is that of people waiting in line, for example at the supermarket checkout (Terry Green, 2012), queueing models are also used to analyse computer performance (I. Mitrani, 1987, 1998; E. Gelembe and I. Mitrani, 1980; Mor Harchol-Balter, 2013; Ramón Puigjaner et al., 1995; Edward D. Lazowska et al., 1984; Leonard Kleinrock, 1976), traffic jams (Tom Vanderbilt, 2009) and in many other areas of activity.

Although there are computer programs that are applicable to queueing theory, to date the statistical computing environment *R* has lacked specific packages for this purpose.

In 2015 and 2016 respectively, **simmer** (Iñaki Ucar and Bart Smeets, 2015) and **queuecomputer** (Anthony Ebert, 2016) were included at *CRAN*. Both apply simulation techniques to queueing models. Simulation techniques have the great advantage of flexibility, enabling a system to be represented at the level of detail desired. The disadvantages are the greater costs incurred, because many parameters must be defined, a large and complex program developed (and debugged) and significant computational resources deployed in order to obtain narrow confidence intervals (Edward D. Lazowska et al., 1984). **queueing** is a queueing model solver, which has the advantage of achieving a favourable balance between accuracy and efficiency (Edward D. Lazowska et al., 1984).

queueing provides **R** users with the most widely-used models: Markovian models, queueing networks and calculators. Although Markovian models or queueing network models may be viewed as very simple models with strong assumptions, they have actually been used to accurately model many real situations, because the accuracy of queueing models is robust with respect to deviations from prior assumptions (Edward D. Lazowska et al., 1984).

Markovian models include the familiar queues observed in real life at supermarket and airport checkouts, restaurant queues, etc. The performance of computer systems can be modelled in a similar way by assuming that the customers are operating system processes, database transactions, etc.
With queueing networks, several classes of customers can be considered, and therefore sizing, capacity planning and what-if scenarios can be addressed with robust assumptions (Edward D. Lazowska et al., 1984). For example, queueing networks can be observed at airports, where various stages, such as check-in, passport control and security checks must be completed in turn before passengers enter the aeroplane.

Finally, calculators are utilities that are used to obtain probabilities, such as that of a supermarket customer being obstructed, of a passenger being rejected after arrival, or of a caller being transferred to hear the operator voice message stating that all phone lines are busy.

These features of the **queueing** package make it a very valuable tool for the applied study of queuing theory, in diverse fields of application, including the following:

- In education, to facilitate the presentation of class content, for learning and practice;
- In research, to achieve a deeper understanding of the models and their principles, using this as a basis for designing more advanced models.
- In health centres, to determine the appropriate size for waiting areas and the average time that patients must wait.
- In database administration, to discover bottlenecks at database servers and to accomodate user load.

- In the design of inbound call-centres, to calculate the optimal number of agents required to maintain a given level of service.

Queueing Package

Nomenclature

To take into account different forms of queue organisation (limited or unlimited space, limited or unlimited population, etc.), a nomenclature of six positions describing the characteristics of the model is usually used (D. G. Kendall, 1953). Its structure is *A/B/C/K/M/Z*, where:

- *A* describes the inter-arrival time probability distribution,
- *B* describes the service time probability distribution,
- *C* is the number of servers or channels,
- *K* is the space limit of the service facility in the sense that no more than *K* customers can be in the system (*C* in service and *K* - *C* waiting in queue),
- *M* is the size of the customer population,
- *Z* is the queue discipline, i.e how the next customer is chosen from the queue when the server completes a service.

When *Z* is *FCFS*, $M = \infty$, and $K = \infty$, the last three positions are omitted.

Thus, *M/M/1/∞/∞/FCFS*, abbreviated to *M/M/1*, describes a model in which the inter-arrival and service times are both exponential (*M* is obtained from the Markovian property of the exponential distribution); there is a single server in a facility that does not impose any restriction on the number of customers; customers arrive from a population that is considered infinite in the sense that the arrival of an individual does not modify the probability of the next arrival; and *FCFS* (*F*irst *C*ome, *F*irst *S*erved) is the most frequent way in which the next customer to be served is chosen from the queue.

Package contents

The package built provides a solution for two families of models, and three commonly-used calculators:

- **Markovian models**: in which inter-arrival and service times are both distributed exponentially (Pazos Arias et al., 2003; Sixto Ríos Insúa et al., 2004; Donald Gross and Carl M. Harris, 1974; Leonard Kleinrock, 1975):

1. *M/M/1*
2. *M/M/c*
3. *M/M/∞*
4. *M/M/1/K*
5. *M/M/c/K*
6. *M/M/c/c*
7. *M/M/1/K/K*
8. *M/M/c/K/K*
9. *M/M/c/K/m*
10. *M/M/∞/K/K*

- **Operational models** (used mainly in modelling computer performance, see Edward D. Lazowska et al. (1984)):

a. *Multiple channel open Jackson networks*
b. *Multiple channel closed Jackson networks*
c. *Single channel multiple class open networks*
d. *Single channel multiple class closed networks*
e. *Single channel multiple class mixed networks*

- **Calculators**:

i. *B-Erlang*
ii. *C-Erlang*
iii. *Engset*

Structure of queueing package

The **queueing** package was developed taking into account the **S3** special class of functions in **R**. With this type of function, different queueuing models can be created in the same way, thus providing the user with a uniform and easy way to create the models. The model is created using the following steps:

1. Create the inputs for the model with the **NewInput** function;
2. Optionally check the inputs with the **CheckInput** function.
3. Create the model by calling **QueueingModel**
4. Print a summary of the model using **print**, or a specific model performance measure such as **W**.

Although step 2 is optional (as it is applied when the **QueueingModel** function is called), it is recommended that the inputs should always be checked, as this makes it easier to understand the data and, thus to correctly build the model.

The following code is an example of a model using **queueing**. It can be thought of as cars arriving at a petrol station, following an exponential distribution at the rate $\lambda = 2$. The cars are served exponentially distributed at the rate $\mu = 3$.

This situation is modelled in **queueing** using a single node in which the customer inter-arrival time and service time both follow an exponential distribution, at the rates $\lambda = 2$ and $\mu = 3$ respectively, as shown in Figure 1.

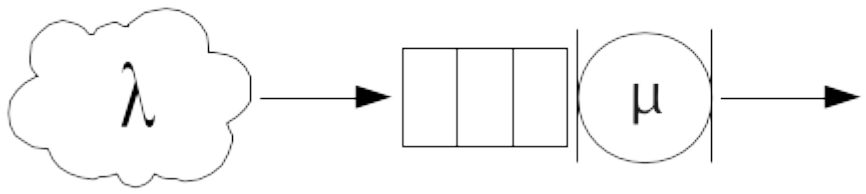


Figure 1: M/M/1 Infinite population, single server. Example applicable to service provided at a petrol station

```
# Load the package
library(queueing)

# Create the inputs for the model.
i_mm1 <- NewInput.MM1(lambda=2, mu=3)

# Optionally check the inputs of the model
CheckInput(i_mm1)

# Create the model
o_mm1 <- QueueingModel(i_mm1)

# Print on the screen a summary of the model
print(summary(o_mm1), digits=2)
```

```
#>   lambda mu c  k  m   RO   P0  Lq   Wq X L W Wqq Lqq
#> 1      2  3 1 NA NA 0.67 0.33 1.3 0.67 2 2 1   1   3
```

The output of the model also includes components, as the functions $F_{W_q}(t)$ and $F_W(t)$, which can be used to view the cumulative probability distribution of the random variables w_q (time waiting) and w (time in the system: time in queue + time being served), assuming *FIFO*(*F*ist *I*n, *F*irst *O*ut) or *FCFS*(*F*ist *C*ome, *F*irst *S*erved) as the queue discipline. Accordingly, $F_{W_q}(t)$ is the probability of a customer waiting for a time less than or equal to t for service (Donald Gross and Carl M. Harris, 1974).

As can be seen in Figure 2, the probability of a customer having to wait for at least 4 units of time is 1, and this coincides with the probability of a customer having to spend 4 total units of time in the system.

```
gTitle <- "Distribution function of w and wq"
fw   <- o_mm1$FW
fwq  <- o_mm1$FWq
n    <- 10
ty   <- "l"
ylab <- "FW(t), FWq(t)"
xlab <- "t"
cols <- c("black", "red")
leg  <- c("FW(t)", "FWq(t)")

curve(fw, from=0, to=n, type=ty, ylab=ylab, xlab=xlab, col=cols[1], main=gTitle)
curve(fwq, from=0, to=n, type=ty, col=cols[2], add=T)
legend("bottomright", leg, lty=c(1, 1), col=cols)
```

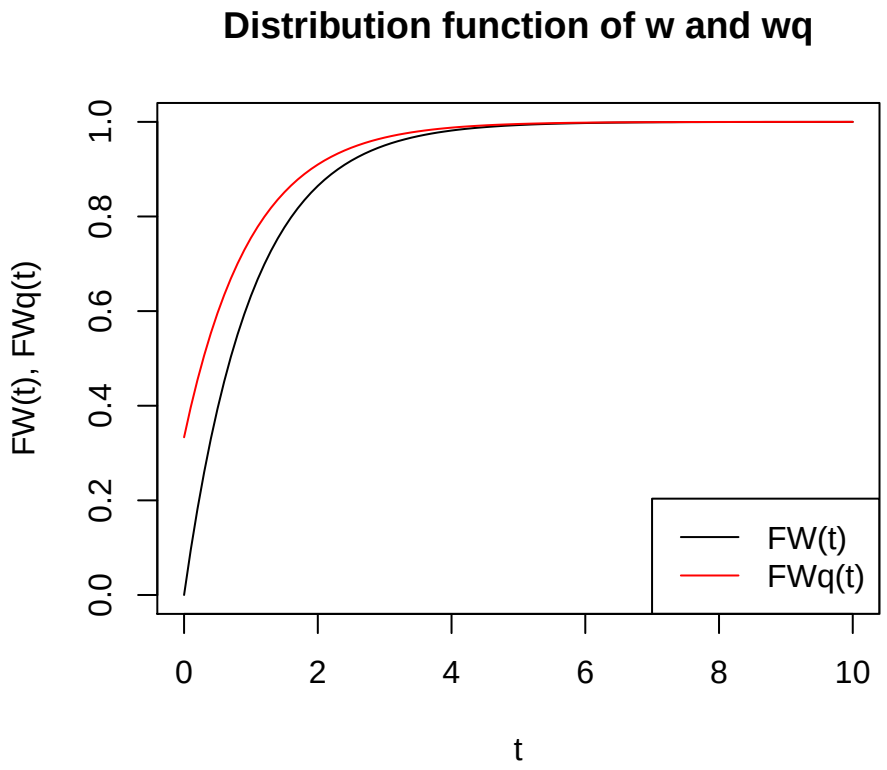


Figure 2: Distribution of random variables w and wq. As can be seen, from t=4, both variables has probability of 1

Performance metrics comparision

queueing provides functions returning the outputs of the model, and therefore additional **R** functions can be developed to study the effect of changing model inputs. For example:

- Change the input parameter λ in the model *M/M/1*:

```
L_f_aux   <- function(x){L  (QueueingModel(NewInput.MM1(lambda=x, mu=1, n=-1)))}
Lq_f_aux  <- function(x){Lq (QueueingModel(NewInput.MM1(lambda=x, mu=1, n=-1)))}
Lqq_f_aux <- function(x){Lqq(QueueingModel(NewInput.MM1(lambda=x, mu=1, n=-1)))}

L_f   <- function(v){sapply(v, L_f_aux)}
Lq_f  <- function(v){sapply(v, Lq_f_aux)}
Lqq_f <- function(v){sapply(v, Lqq_f_aux)}
```

```
gt     <- "L, Lq and Lqq"
ylab   <- "L, Lq, Lqq"
xlab   <- "rho"
n      <- 100
to     <- 0.99
ty     <- "l"
lty    <- c(1, 1, 1)
cols   <- c("blue", "red", "green")
leg    <- c("L", "Lq", "Lqq")

curve(L_f,   from=0, to=to, n=n, ylab=ylab, xlab=xlab, col=cols[1], type=ty, main=gt)
curve(Lq_f,  from=0, to=to, n=n, col=cols[2], add=T, type=ty)
curve(Lqq_f, from=0, to=to, n=n, col=cols[3], add=T, type=ty)
legend("topleft", leg, lty=c(1, 1, 1), col=cols)
```

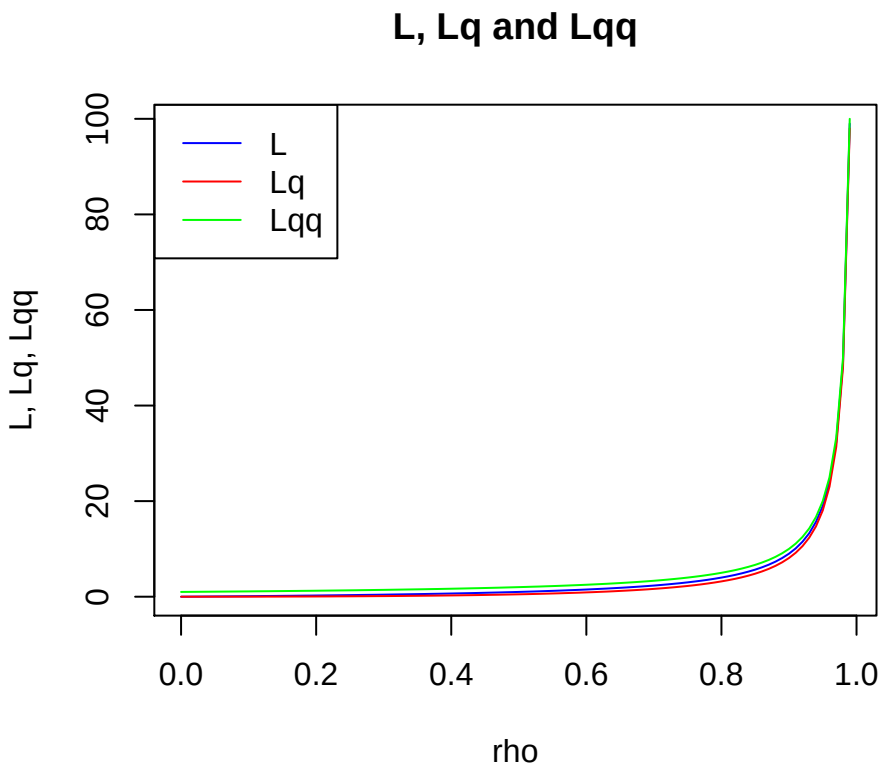


Figure 3: Evolution of the number of customers in system (L), in queue (Lq) and in queue knowing that queue exists (Lqq). The three functions exhibit a dramatic increase in the number of customers as rho tends to 1. Observe that in general, Lqq > L > Lq

As can be seen in Figure 3, the three functions exhibit a dramatic increase in the number of customers, from $\rho \to 1$. Observe that $L_{qq} > L > L_q$. $L > L_q$ because the number of customers in the system must include the customer currently receiving the service; the interesting insight here is that $L_{qq} > L$, and so merely knowing that there exists a queue raises the number of customers in the system.

```
x <- seq(from=0, to=0.99, by=0.01)
Lqq_f(x) - L_f(x)
```

```
#>   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
#>  [36] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
#>  [71] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

- The arrival rate changes in line with the number of servers in a *M/M/c* model:

```
W_f_aux   <- function(x){W  (QueueingModel(NewInput.MMC(lambda=x, mu=1.01, c=x)))}
Wq_f_aux  <- function(x){Wq (QueueingModel(NewInput.MMC(lambda=x, mu=1.01, c=x)))}
```

```
W_f   <- function(v){sapply(v, W_f_aux)}
Wq_f  <- function(v){sapply(v, Wq_f_aux)}

gt     <- "W and Wq"
ylab   <- "W, Wq"
xlab   <- "lambda, c"
n      <- 14
ty     <- "l"
leg    <- c("W", "Wq")
lty    <- c(1, 1, 1)
cols   <- c("blue", "red")

curve(W_f, from=1, to=n, n=n, ylab=ylab, xlab=xlab , col=cols[1], type=ty, main=gt)
curve(Wq_f,  from=1, to=n, n=n, col=cols[2], add=T, type=ty)
legend("topright", leg, lty=lty, col=cols)
```

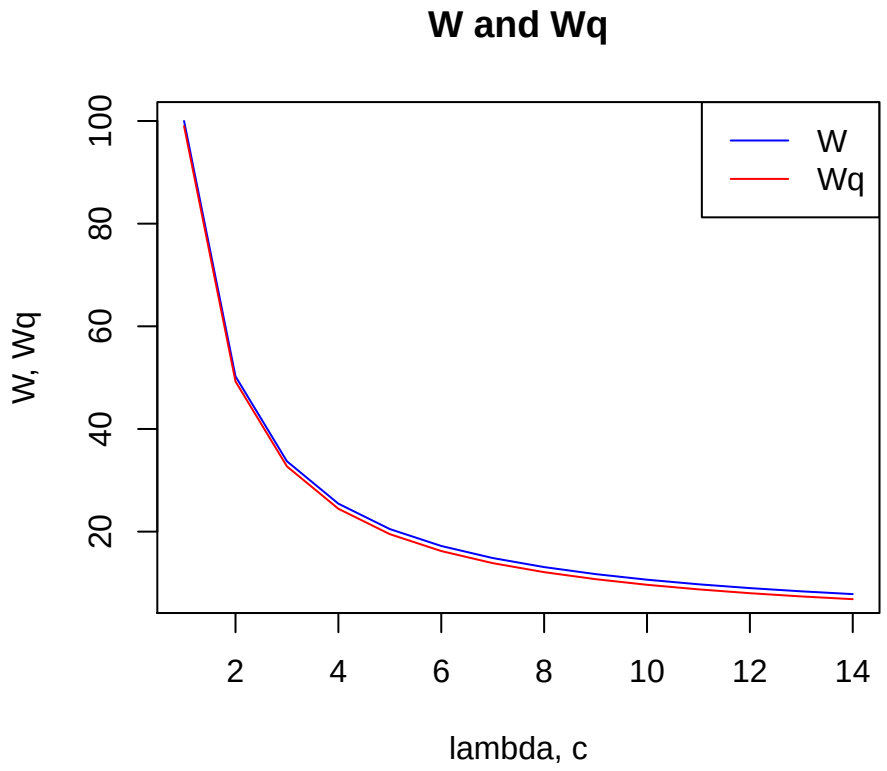


Figure 4: Evolution of the mean time in queue and in the system according to lambda and c increase equally. Observe that the time tends to zero as the number of servers increase

Observe in Figure 4 that although λ and c increase at the same rate, the time tends to zero as the number of servers increases, meaning that the capacity of the facility increases faster than the work pending attention.

- Sometimes it is useful to compare different models, for example to measure the impact of duplicating the number of servers or of restricting the queue area. **queueing** includes a useful function to compare different queueing models, so that the tradeoff between performace measures can be seen at a glance.

```
o_mm2  <- QueueingModel(NewInput.MMC(lambda=2, mu=3, c=2))
o_mm2k <- QueueingModel(NewInput.MM1K(lambda=2, mu=3, k=5))
CompareQueueingModels(o_mm1, o_mm2, o_mm2k)
```

```
#>   lambda mu c  k  m        RO        P0         Lq         Wq        X
#> 1      2  3 1 NA NA 0.6666667 0.3333333 1.33333333 0.66666667 2.000000
```

```
#> 2      2  3 2 NA NA 0.3333333 0.5000000 0.08333333 0.04166667 2.000000
#> 3      2  3 1  5 NA 0.6345865 0.3654135 0.78796992 0.41390205 1.903759
#>          L         W       Wqq      Lqq
#> 1 2.000000 1.0000000 1.0000000 3.000000
#> 2 0.750000 0.3750000 0.2500000 1.500000
#> 3 1.422556 0.7472354 0.6717949 2.015385
```

Queueing Network Models

In addition to the basic Markovian models, **queueing** also offers several routines with which to build different queueing networks.

Although simple queueing models have been used to correctly model different situations (Allan L. Scherr, 1967), a queueing network must be used when several stages or places need to be visited in order to obtain full service. There are some "special" queueing networks in the simple models: thus, *M/M/c/K* (J. Medhi, 2003; Hisashi Kobayashi, 1978) and *M/M/c/K/m* are two examples of closed queueing models.

For models with more than two stations or with different classes of customers, **queueing** offers single and multiple channel open and closed Jackson networks, single channel open and closed multiple classes of customers and single channel mixed networks.

The steps taken to create a queueing network are the same as those for a single queueing model: the first is to apply the corresponding *NewInput* function; then, optionally, the *CheckInput* function is called, to help with the input parameters of the model, and finally the model is built with the *QueueingModel* function.

The difference between this and the single node model is that there exist several different functions with which to create the inputs of the network. For example, for a single-class closed Jackson network, there are three ways to create the input, as long as the problem has a probability route or the problem has been defined as operational (Peter J. Denning and Jeffrey P. Buzen, 1978).

For example, imagine the urgent-treatment box of a hospital. Patients arrive at the rate of 10 patients/hour. A doctor examines each patient and may provide treatment and then send the patient home, or derive the patient to a specialist (orthopaedist, cardiologist, etc.), who in turn may request tests (blood analysis, radiography, etc.), and then send the patient home or recommend hospitalisation. In the latter case(assuming there are beds available), treatment will be provided, more checks and tests may be required and, eventually, the patient will leave (not always to go home).

In this case, the model used is a single-class Open Jackson network, with five nodes: general doctor, orthopaedist, cardiologist, checks and tests box (composed of 15 technicians) and hospitalised (with space for any number of patients). Except for visiting the general physician, in no case is a patient allowed to directly enter another node.

The probability route matrix is measured as:

$$\begin{pmatrix} 0 & 0.3 & 0.2 & 0 & 0 \\ 0 & 0 & 0 & 0.7 & 0.1 \\ 0 & 0 & 0 & 0.8 & 0.15 \\ 0 & 0.3 & 0.7 & 0 & 0 \\ 0 & 0 & 0 & 0.6 & 0.3 \end{pmatrix}$$

The model is built as follows, noting *gd* as general physician, *ort* as orthopaedist, *car* as cardiologist, *tb* as checks and tests box and *hos* as hospitalised:

```
data <- c(0, 0.3, 0.2, 0, 0, 0, 0, 0, 0.7, 0.1, 0, 0, 0, 0.8, 0.15, 0, 0.4, 0.3, 0, 0.3)
prob <- matrix(data=data, byrow = TRUE, nrow = 5, ncol=5)
gd  <- NewInput.MM1(lambda=10, mu=25, n=0)
ort <- NewInput.MM1(lambda=0,  mu=18, n=0)
car <- NewInput.MM1(lambda=0,  mu=20, n=0)
tb  <- NewInput.MMC(lambda=0, mu=12, c=15, n=0)
hos <- NewInput.MMInf(lambda=0, mu=0.012, n=0)
hospital <- NewInput.OJN(prob=prob, gd, ort, car, tb, hos)
```

The fact that no patient is allowed to visit any doctor other than the general physician is specified, by setting lambda to zero at the node.

The parameters can be checked using *CheckInput* and the model is built using *QueueingModel*

```
CheckInput(hospital)
m_hospital <- QueueingModel(hospital)
```

```
print(summary(m_hospital), digits = 2)
```

```
#>       L  W  X     Lk     Wk   Xk     ROk
#> Net 465 47 10     NA     NA   NA      NA
#> Nd1  NA NA NA   0.67  0.067 10.0   0.400
#> Nd2  NA NA NA   1.11  0.111  9.5   0.525
#> Nd3  NA NA NA   0.50  0.050  6.7   0.335
#> Nd4  NA NA NA   1.00  0.100 12.0   0.067
#> Nd5  NA NA NA 462.21 46.221  5.5 462.213
```

As can be seen from the output, results are calculated both for the complete network and for each node.

queueing also allows multiple classes of patients to be created, although currently only in the operational way (Peter J. Denning and Jeffrey P. Buzen, 1978).

At the hospital, the patients arriving at the urgent-treatment box are usually one of two types: high priority or normal priority.

By appropriately adjusting the input parameters, the problem can now be modelled as follows:

```
classes <- 2
vLambda <- c(2, 10)
nodes   <- 5
vType     <- c("Q", "Q", "Q", "Q", "D")
vHigh   <- c(2, 4, 2, 6, 2)
vNorm   <- c(1, 2, 1, 3, 0.5)
vVisit  <- matrix(data=c(vHigh, vNorm), nrow=2, ncol=5, byrow = TRUE)
sHigh    <- c(1/100, 1/250, 1/300, 1/600, 1/5)
sNorm    <- c(1/90, 1/150, 1/200, 1/300, 1/3)
vService <-matrix(data=c(sHigh, sNorm), nrow=2, ncol=5, byrow = TRUE)

cl_hosp <- NewInput.MCON(classes, vLambda, nodes, vType, vVisit, vService)
```

Some parameters have a different meaning in this model. Although the order of the classes and nodes is chosen by the modeller, once it is decided, consistency must be maintained.

- *classes*: number of classes of patients;
- *vLambda*: arrival rate of each class, setting high priority first;
- *nodes*: number of nodes;
- *vType*: except nodes like hospital (no limit to number of beds available), which is "D", the rest are "Q";
- *vHigh*: average number of visits by each *high priority* patient to nodes;
- *vNorm*: average number of visits by each *normal priority* patient to nodes;
- *vVisit*: recall that order consistency must be preserved;
- *sHigh*: average service time in each node for *high priority* patients;
- *sNorm*: average service time in each node for *normal priority* patients.

As before, the *CheckInput* function should be used before building the model with *QueueingModel*.

```
CheckInput(cl_hosp)
m_cl_hosp <- QueueingModel(cl_hosp)
print(summary(m_cl_hosp), digits = 2)
```

```
#>       L    W  X   Lc   Wc Xc    Lk     Wk Xk   ROk   Lck    Wck Xck  ROck
#> Net   3 0.25 12   NA   NA NA    NA     NA NA    NA    NA     NA  NA    NA
#> Cl1  NA   NA NA 0.92 0.46  2    NA     NA NA    NA    NA     NA  NA    NA
#> Cl2  NA   NA NA 2.12 0.21 10    NA     NA NA    NA    NA     NA  NA    NA
#> Nd1  NA   NA NA   NA   NA NA 0.178 0.0148 14 0.151    NA     NA  NA    NA
#> Nd2  NA   NA NA   NA   NA NA 0.198 0.0165 28 0.165    NA     NA  NA    NA
#> Nd3  NA   NA NA   NA   NA NA 0.068 0.0056 14 0.063    NA     NA  NA    NA
#> Nd4  NA   NA NA   NA   NA NA 0.136 0.0114 42 0.120    NA     NA  NA    NA
#> Nd5  NA   NA NA   NA   NA NA 2.467 0.2056  9 2.467    NA     NA  NA    NA
#> CN11 NA   NA NA   NA   NA NA    NA     NA NA    NA 0.047 0.0236   4 0.040
#> CN12 NA   NA NA   NA   NA NA    NA     NA NA    NA 0.038 0.0192   8 0.032
#> CN13 NA   NA NA   NA   NA NA    NA     NA NA    NA 0.014 0.0071   4 0.013
#> CN14 NA   NA NA   NA   NA NA    NA     NA NA    NA 0.023 0.0114  12 0.020
#> CN15 NA   NA NA   NA   NA NA    NA     NA NA    NA 0.800 0.4000   4 0.800
```

```
#> CN21 NA   NA NA   NA   NA NA    NA     NA NA    NA 0.131 0.0131  10 0.111
#> CN22 NA   NA NA   NA   NA NA    NA     NA NA    NA 0.160 0.0160  20 0.133
#> CN23 NA   NA NA   NA   NA NA    NA     NA NA    NA 0.053 0.0053  10 0.050
#> CN24 NA   NA NA   NA   NA NA    NA     NA NA    NA 0.114 0.0114  30 0.100
#> CN25 NA   NA NA   NA   NA NA    NA     NA NA    NA 1.667 0.1667   5 1.667
```

In this case, the output has more detailed information than in the previous model, as the class performance measures are also included.

Queueing Calculators

The calculators included in **queueing** are frequently used for sizing call centres, telecomunications systems, etc.

1. **Erlang-B**. This function is derived from a *M/M/c/c* model when the *c+1* customer arrives when there are *c* already in the system.
2. **Erlang-C**. This function is frequently used in call centres to correctly set the number of agents. It denotes the probability of a customer having to queue because all the servers are busy in a *M/M/c*.
3. **Engset**. When the population is finite, that is, when each new arrival changes the probability of the next arrival, the *Engset* function gives the probability of an arrival having to return to the source merely because there is no room availabl. This situation is modelled by *M/M/c/c/N* with $c < N$.

The graphics normally shown in queueing theory books such as (Donald Gross and Carl M. Harris, 1974), page 111, can be easily obtained and improved with the calculators included:

```
servers    <- 1:50
numServers <- length(servers)
rho        <- c(1, 5, 10, 15, 20, 24, 30, 40, 50)
rho_size   <- length(rho)
pRes       <- array(data=0, dim=c(numServers, rho_size))

for (i in 1:numServers)
  for (j in 1:rho_size)
    pRes[i, j] <- B_erlang(i, rho[j])

colrs  <- rainbow(n=rho_size)

xlim  <- c(1, numServers)
ylim  <- c(0, 1)
xlab  <- "Number of servers"
ylab  <- "Probability"
gt    <- "B-Erlang prob. for different loads"
y     <- pRes[, 1]
x     <- servers
col   <- colrs[1]

plot(x=x, y=y, xlim=xlim, ylim=ylim, type="l", col=col, xlab=xlab, ylab=ylab, main=gt)

for (j in 2:rho_size)
  lines(x=1:numServers, y=pRes[, j], col=colrs[j])

leg <- as.character(rho)
tr  <- "topright"
lty <- rep(1, rho_size)
lwd <- rep(0.01, rho_size)

legend(x=tr, legend=leg, lty=lty, lwd=lwd, col=colrs)
```

Conclusions

queueing is a useful tool for both academic and professional use. In education, the main models learned in class are presented in detail, providing numerous performance measures within a versatile

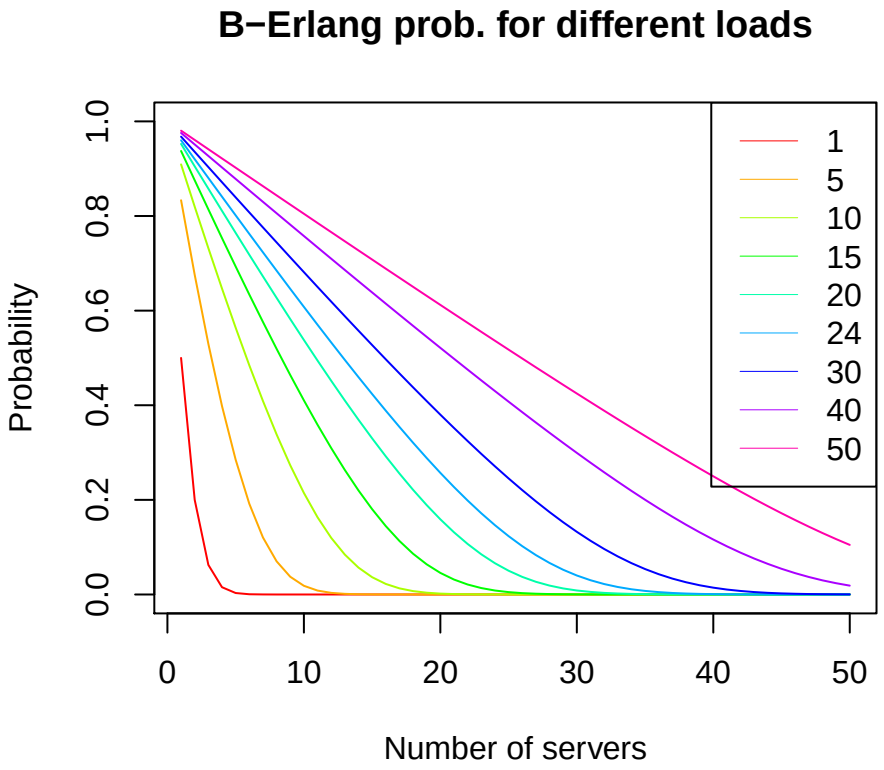


Figure 5: Probability of not finding an available server as the number of servers increase

tool, helpful to students and teachers alike. For professionals, the package can be used both to design and size systems or workloads according to perfomance requirements, and to explain deviations in systems as they are evolving.

queueing has been developed to make it easy to learn how the *NewInput*, *CheckInputs* and *QueueingModel* models are built. Both basic Markovian models and queueing networks can be built using the same process.

More queue disciplines, together with intermediate and advanced models, will be included progressively in the package to make it even more versatile.

Acknowledgments

This work would not have been possible without the invaluable help of Yolanda Román Montoya.

queueing has its roots in the peerless stochastic processes classes given by Sixto Ríos Insúa, Alfonso Mateos Caballero, Mª Concepción Bielza Lozoya and David Ríos Insúa. Although the package is dedicated especially to the memory of Sixto Ríos Insúa, I am sincerely grateful to and appreciative of all these outstanding educators.

Finally, I thank everyone who has made contributions and suggestions or made use of the package.

Bibliography

Allan L. Scherr. *An Analysis of Time-Shared Computer Systems*. The M.I.T. Press, Cambridge, Massachusetts, 1967. [p122]

Anthony Ebert. *queuecomputer: Computationally Efficient Queue Simulation*, 2016. URL https://github.com/AnthonyEbert/queuecomputer. [p116]

D. G. Kendall. Stochastic Processes Occurring in the Theory of Queues and Their Analysis by the Method of the Imbedded Markov Chain. *The Annals of Mathematical Statistics*, 24(3):338–354, 1953. URL https://doi.org/10.1214/aoms/1177728975. [p117]

Donald Gross and Carl M. Harris. *Fundamentals of Queueing Theory*. Wiley, 1974. [p116, 117, 118, 124]

E. Gelembe and I. Mitrani. *Analysis and Synthesis of Computer Systems*, 1980. [p116]

Edward D. Lazowska, John Zahorjan, G. Scott Graham, and Kenneth C. Sevcik. *Quantitative System Performance. Computer System Analysis Using Queueing Network Models*. Prentice-Hall, Inc., 1984. [p116, 117]

Erlang, Agner K. The Theory of Probabilities and Telephone Conversations. *Nyt Tidsskrift for Matematik*, 20(B):33–39, 1909. URL http://www.jstor.org/stable/24528622. [p116]

Hisashi Kobayashi. *Modeling and Analysis. An Introduction to System Performance Evaluation Methodology*. Addison Weasly, 1978. [p122]

I. Mitrani. *Modelling of Computer and Communication Systems*. Cambridge University Press, 1987. [p116]

I. Mitrani. *Probabilistic Modelling*. Cambridge University Press, 1998. [p116]

Iñaki Ucar and Bart Smeets. *simmer: Discrete-Event Simulation for R*, 2015. URL http://r-simmer.org, https://github.com/r-simmer/simmer. [p116]

J. Medhi. *Stochastic Models in Queueing Theory, 2nd Edition*. Academic Press, 2003. [p122]

Leonard Kleinrock. *Queueing Systems. Volume 1: Theory*. John Wiley and Sons, Inc, 1975. [p117]

Leonard Kleinrock. *Queueing Systems. Volume 2: Computer Applications*. John Wiley and Sons, Inc, 1976. [p116]

Mor Harchol-Balter. *Performance Modeling and Design of Computer Systems. Queueing Theory in Action*. Cambridge University Press, 2013. [p116]

J. J. Pazos Arias, A. Suárez González, and R. P. Díaz Redondo. *Teoría de Colas y Simulación de Eventos Discretos*. Pearson Educación, 2003. [p117]

Peter J. Denning and Jeffrey P. Buzen. The Operational Analysis of Queueing Network Models. *ACM Computing Surveys*, (10):225–261, 1978. URL https://doi.org/10.1145/356733.356735. [p122, 123]

Ramón Puigjaner, Juan José Serrano, and Alicia Rubio. *Evaluación y explotación de sistemas informáticos*. Editorial Síntesis, 1995. [p116]

Sixto Ríos Insúa, Alfonso Mateos Caballero, Mª Concepción Bielza Lozoya, and Antonio Jiménez Martín. *Investigación Operativa. Modelos Determinísticos y Estocásticos*. Editorial Centro de Estudios Ramón Areces, 2004. [p117]

Terry Green. *Cashier Number 3, Please!: Creating Fairer, Faster Service*. Marshall Cavendish, 2012. [p116]

Tom Vanderbilt. *Tráfico*. DEBATE, 2009. [p116]

Pedro Cañadilla Jiménez
Statistics Operations Research Department
Universidad de Granada
Spain
pcanadilla@correo.ugr.es

Yolanda Román Montoya
Statistics Operations Research Department
Universidad de Granada
Spain
yroman@ugr.es

ctmcd: An R Package for Estimating the Parameters of a Continuous-Time Markov Chain from Discrete-Time Data

by Marius Pfeuffer

Abstract This article introduces the R package **ctmcd**, which provides an implementation of methods for the estimation of the parameters of a continuous-time Markov chain given that data are only available on a discrete-time basis. This data consists of partial observations of the state of the chain, which are made without error at discrete times, an issue also known as the embedding problem for Markov chains. The functions provided comprise matrix logarithm based approximations as described in Israel et al. (2001), as well as Kreinin and Sidelnikova (2001), an expectation-maximization algorithm and a Gibbs sampling approach, both introduced by Bladt and Sørensen (2005). For the expectation-maximization algorithm Wald confidence intervals based on the Fisher information estimation method of Oakes (1999) are provided. For the Gibbs sampling approach, equal-tailed credibility intervals can be obtained. In order to visualize the parameter estimates, a matrix plot function is provided. The methods described are illustrated by Standard and Poor's discrete-time corporate credit rating transition data.

Introduction

The estimation of the parameters of a continuous-time Markov chain (see, e.g., Norris (1998) or Ethier and Kurtz (2005); also referred to as Markov process) when only discrete time observations are available is a widespread problem in the statistical literature. Dating back to Elfving (1937), this issue is also known as the embedding problem for discrete-time Markov chains. The problem occurs in the modeling of dynamical systems when due to various reasons such as a difficult measurement procedure only discrete-time observations are available. This is the case in a wide range of applications, e.g., in the analysis of gene sequence data (see, e.g., Hobolth and Stone (2009), Verbyla et al. (2013) or Chen et al. (2014)), for causal inference in epidemiology (see, e.g., Zhang and Small (2012)), for describing the dynamics of open quantum systems (see, e.g., Cubitt et al. (2012)), or in rating based credit risk modeling (see, e.g., Dorfleitner and Priberny (2013), Yavin et al. (2014) or Hughes and Werner (2016)) to name only a few.

In the following, an explicit statement of the missing data setting shall be given and the notation used in this manuscript shall be introduced: Consider that realizations of a continuous-time Markov chain, i.e., paths of states $s \in \{1, \ldots, S\}$, which change at times $\tau_1, \ldots, \tau_K$ are given. For a single path, this is exemplarily illustrated in figure 1.

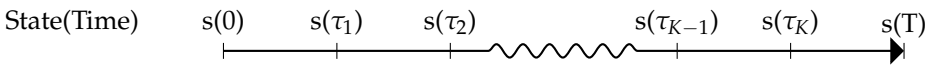


Figure 1: Discrete-Time / Continuous-Time Setting

In the missing data situation described in this paper, these paths are however not completely observed, but only at points in time 0 and T. The available observations are thus the states $s(0)$ and $s(T)$ and these states are assumed to be observed without error. The cumulative discrete-time data over all paths can be summarized into conditional transition matrices with absolute transition frequencies $\mathbf{N}_{T|0}$ or relative transition frequencies $\mathbf{P}_{T|0}$ (in the following, the abbreviate notations $\mathbf{N}_T$ and $\mathbf{P}_T$ will be used). The continuous-time state changes $s(\tau_k), k \in \{1, \ldots, K\}$ are latent variables.

A continuous-time Markov chain has the parameter set

$$\mathbf{Q} = \{q_{ij}\}_{1 \leq i \leq I, 1 \leq j \leq J, I=J=S} : q_{ii} \leq 0, q_{ij,i \neq j} \geq 0, \sum_{j=1}^{J} q_{ij} = 0,$$

which is called generator matrix, transition rate matrix or intensity matrix. The problem is now to estimate the parameters $\mathbf{Q}$ from the partial observations at times 0 and T. This allows to derive a matrix of conditional discrete-time state change predictions $\mathbf{P}_{T_a}$ for arbitrary time intervals $[0, T_a]$ of length T_a by employing the matrix exponential function

$$\mathbf{P}_{T_a} = \exp(\mathbf{Q} T_a).$$

Besides the R package **msm**, see Jackson et al. (2011), which only provides functions for direct likelihood optimization, there is no other publicly accessible implementation available which allows for estimating the parameters of a continuous-time Markov chain given that data have been only observed on a discrete-time basis. Against this background, this paper introduces the R package **ctmcd**, a continuously extended, improved and documented implementation based of what started as supplementary R code to Pfeuffer (2016). The functions of the package are explained and illustrated by Standard and Poor's corporate rating transition data. The outline of the paper is as follows: first, three matrix logarithm adjustment approaches are explained. Second, likelihood inference is illustrated for an instance of the expectation-maximization algorithm. Third, the implementation of a Gibbs sampler is presented to facilitate Bayesian inference. Numerical properties of the different approaches are evaluated and examples for more complex applications of the methods are shown. Finally, the results of the paper are summarized.

Matrix logarithm adjustment approaches

A basic approach to estimate generator matrices from discrete-time observations is to inversely use the matrix exponential relationship between conditional discrete time transition matrices $\mathbf{P}_T$ (the cumulative discrete-time state change data) and the parameters $\mathbf{Q}$, i.e., to employ a matrix logarithm function, which leads to the estimate

$$\mathbf{Q}_c = \log(\mathbf{P}_T) = \sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{k} (\mathbf{P}_T - \mathbf{I})^k.$$

Besides finite truncation of this Taylor series, the matrix logarithm can, e.g., also be calculated by an eigendecomposition, which is the default setting in **ctmcd**.

However, the matrix logarithm approach has two shortcomings. First, the matrix logarithm is not a bijective function. As, e.g., shown by Speakman (1967), a transition matrix can have more than one valid generator. However, for a certain subset of discrete-time transition matrices, it can be shown that there exists only a single unique generator, for details on criteria (for discrete-time transition matrices) under which this is the case, see, e.g., Cuthbert (1972), Cuthbert (1973), Singer and Spilerman (1976) or Israel et al. (2001). Second, the method requires that the derived matrix $\mathbf{Q}_c$ actually meets the above outlined parameter constraints for Markov generator matrices, concretely that off diagonal elements are non-negative, which is not necessarily the case. Therefore, in the following we shall discuss techniques for adjusting logarithms of the discrete time data matrices $\mathbf{P}_T$, so that proper generator matrices can be derived.

Diagonal and weighted adjustment

In this context, Israel et al. (2001) introduce two approaches. On the one hand, **diagonal adjustment (DA)** works by forcing negative off-diagonal elements of $\mathbf{Q}_c$ to zero

$$q_{ij,i\neq j} = 0 | q_{ij,c} < 0$$

and adjusting the diagonal elements

$$q_{ii} = -\sum_{j=1}^{J} q_{ij}$$

to ensure that $\sum_{j=1}^{J} q_{ij} = 0$. In **ctmcd** such an estimate can be performed by passing `method="DA"` to `gm()`, the generic generator matrix estimation function of the package. The method requires the specification of a discrete time transition matrix `tm`, which in the case of matrix logarithm adjustment approaches is a matrix of **relative** transition frequencies, which refers to the matrix $\mathbf{P}_T$ introduced above, as input data.

In order to illustrate the methods, we employ Standard and Poor's (2000) global corporate credit ratings data. The rating categories in this data set have the commonly known symbols AAA, AA, A, BBB, BB, B, C and D. These abbreviations represent states of decreasing credit quality whereas category D stands for the event of credit default, which means that if rated D the obligor cannot or does not have the willingness to meet its financial obligations any more. The data is provided as `tm_abs` a matrix of discrete-time absolute transition frequencies from the first to the last day of the fiscal year 2000. We have to take into account that the default category D has to be considered as an absorbing state, because once an obligor has defaulted it can not escape this state any more. Following the intuitive ordering of decreasing credit quality described above, in this example the default state will refer to row 8. Thus, in order to convert the data into the required format, we have to create

a matrix of relative transition frequencies `tm_rel` by standardizing the row entries to sum to 1 and adding a unit row vector as row 8 with the last entry of this row being 1. Subsequently, we can apply the diagonal adjustment approach by specifying `tm=tm_rel` and the time horizon of this discrete-time matrix as `te=1`, because `tm_abs` and `tm_rel` refer to credit rating changes for a single year time interval (fiscal year 2000).

```
data(tm_abs)
tm_rel <- rbind((tm_abs / rowSums(tm_abs))[1:7,], c(rep(0, 7), 1))
gmda <- gm(tm=tm_rel, te=1, method="DA")
```

On the other hand, Israel et al. (2001) also describe the **weighted adjustment (WA)** of the non-negative off-diagonal entries as an alternative, i.e., off diagonal elements are adjusted by

$$q_{ij,i\neq j} = q_{ij,c} + \frac{|q_{ij,c}|}{\sum_{j\neq i}|q_{ij,c}|}\sum_{j\neq i} q_{ij,c}\mathbb{1}(q_{ij,c}<0)|q_{ij,c}\geq 0,$$

where the cut off jump probability mass is redistributed among the remaining positive off diagonal elements according to their absolute values. In analogy to diagonal adjustment, a weighted adjustment estimate can be derived by using `method="WA"` as follows:

```
gmwa <- gm(tm=tm_rel, te=1, method="WA")
```

Quasi-optimization

The third matrix logarithm adjustment approach is the **quasi-optimization (QO)** procedure of Kreinin and Sidelnikova (2001). This method finds a generator $\mathbf{Q}$ from the set of all possible generator matrices $\mathbf{Q}'$ by solving the minimization problem

$$\mathbf{Q} = \arg\min_{\mathbf{Q}'}\sum_{i=1}^{I}\sum_{j=1}^{J}(q'_{ij} - q_{ij,c})^2,$$

which means that the algorithm chooses a generator matrix which is closest to the matrix logarithm in terms of sum of squared deviations.

```
gmqo <- gm(tm=tm_rel, te=1, method="QO")
```

By specifying `method="QO"`, we can then get a quasi-optimization approach result for our data. Despite the possibility to just show the parameter estimates for the different methods in the console using, e.g., `print(gmDA)` or simply calling `gmDA()`, **ctmcd** also provides a matrix plot function `plotM()` that especially allows the visualization of generator matrix estimates and can be easily accessed by the generic `plot()` function:

```
plot(gmda)
plot(gmwa)
plot(gmqo)
```

The results can be seen in figure 2.

Likelihood inference

Given that complete continuous-time data is available, the likelihood function for a generator matrix is given by

$$L(\mathbf{Q}) = \prod_{i=1}^{I}\prod_{j\neq i} q_{ij}^{N_{ij}(T)}\exp(-q_{ij}R_i(T)),$$

where $N_{ij}(T)$ denotes the number of transitions from i to j within time T and $R_i(T)$ for the cumulative sojourn times in state i before a state change occurs. A maximum likelihood estimate for a single off-diagonal element of $\mathbf{Q}$ can then be derived by

$$q_{ij,ML} = \frac{N_{ij}(T)}{R_i(T)},$$

for more information see, e.g., Inamura (2006).

Figure 2: Matrix Logarithm Adjustment Approaches

Expectation-maximization algorithm

The difficulty is now that when data is only observed at times 0 and T, the expressions $N_{ij}(T)$ and $R_i(T)$ are not known. In order to derive a maximum likelihood estimate given this partially accessible observations setting, Bladt and Sørensen (2005) derive an instance of the **expectation-maximization (EM) algorithm**. The missing data is then iteratively imputed by conditional expectations given the current parameter set. This requires a complicated computation of the integrals in

$$E(R_i(T)|\mathbf{Q}_l, s(0), s(T)) = \frac{n_{s(0)s(T)}\mathbf{u}_{s(0)}^T \left(\int_0^T \exp(\mathbf{Q}_l t)\mathbf{u}_i\mathbf{u}_j^T \exp(\mathbf{Q}_l(T-t))dt\right)\mathbf{u}_{s(T)}}{\mathbf{u}_{s(0)}^T \exp(\mathbf{Q}_l T)\mathbf{u}_{s(T)}}$$

$$\text{and} \quad E(N_{ij}(T)|\mathbf{Q}_l, s(0), s(T)) = \frac{n_{s(0)s(T)}q_{ij,l}\mathbf{u}_{s(0)}^T \left(\int_0^T \exp(\mathbf{Q}_l t)\mathbf{u}_i\mathbf{u}_j^T \exp(\mathbf{Q}_l(T-t))dt\right)\mathbf{u}_{s(T)}}{\mathbf{u}_{s(0)}^T \exp(\mathbf{Q}_l T)\mathbf{u}_{s(T)}},$$

where n refers to an element of the discrete-time absolute transition frequency matrix $\mathbf{N}_T$, $\mathbf{u}_k$ denotes a unit vector with entry 1 at position k and l points to the current iteration step of the EM algorithm. The computation of the integrals is carried out following the matrix exponential approach described in van Loan (1978) and Inamura (2006). In order to perform an estimate based on the EM algorithm an initial generator matrix guess has to be chosen, which has to be a proper generator matrix. In the following example, this will be the matrix `gm0`, which is an arbitrarily chosen generator matrix where all off diagonal entries are 1 and state 8 is determined as an absorbing state.

```
gm0 <- matrix(1, 8, 8)
diag(gm0) <- 0
diag(gm0) <- -rowSums(gm0)
gm0[8,] <- 0
```

The maximum likelihood estimate can then be obtained by using the `gm()` function, providing a matrix of absolute numbers of state changes (in the credit rating example i.e., `tm=tm_abs`), specifying the method argument by `method="EM"` and setting an initial guess (here: `gmguess=gm0`).

```
gmem <- gm(tm=tm_abs, te=1, method="EM", gmguess=gm0)
plot(gmem)
plot(gmem$ll, main="Expectation Maximization Algorithm\nLog Likelihood Path",
     xlab="Iteration", ylab="Log-Likelihood")
```

The result of this estimate can be seen in figure 3 together with a plot of the log-likelihood path of the single EM algorithm iteration steps.

Direct likelihood optimization

The function being actually optimized by the EM algorithm is the marginal likelihood

$$L(\mathbf{Q}|\mathbf{N}_T) = \prod_{i=1}^{I}\prod_{j=1}^{J}(\exp(\mathbf{Q}\cdot T))_{ij}^{n_T;ij}.$$

Besides the EM algorithm, also other numerical optimization methods can be employed to perform the maximization of this function. The R-package **msm**, which is actually built for estimating Markov models with covariates, so called multi-state models, contains simplex optimization (`opt.method="optim"`), Newton optimization (`opt.method="nlm"`), a bounded optimization by a quadratic approximation approach (`opt.method="bobyqa"`) introduced by Powell (2009) and a Fisher scoring technique by Kalbfleisch and Lawless (1985) (`opt.method="fisher"`). In order to benchmark the EM algorithm, the different techniques shall be compared using the derived maxima and the time needed to perform the estimation.

```
### Data transformation for msm function
mig <- NULL
id <- 0
for(i in 1:7) {
  for(j in 1:8) {
    if(tm_abs[i,j] > 0) {
      for(n in 1:tm_abs[i,j]) {
        id <- id + 1
        mig <- rbind(mig, c(id, 0, i), c(id, 1, j))
      }
    }
  }
}
mig_df <- data.frame(id=mig[,1], time=mig[,2], state=mig[,3])

### Comparing estimates
gmem <- gm(tm_abs, te=1, method="EM", eps=1e-7, gmguess=gm0)
ctmcdlogLik(gmem$par, tm_abs, 1)

q0 <- rbind(matrix(1, 7, 8), 0)
msm_est1 <- msm(state ~ time, id, data=mig_df, qmat=q0,
                opt.method="optim", gen.inits=TRUE)
ctmcdlogLik(qmatrix.msm(msm_est)[[1]], tm_abs, 1)
msm_est2 <- msm(state ~ time, id, data=mig_df, qmat=q0,
                opt.method="nlm", gen.inits=TRUE)
ctmcdlogLik(qmatrix.msm(msm_est)[[1]], tm_abs, 1)
msm_est3 <- msm(state ~ time, id, data=mig_df, qmat=q0,
                opt.method="nlm", gen.inits=TRUE)
ctmcdlogLik(qmatrix.msm(msm_est)[[1]], tm_abs, 1)

msm_est4 <- msm(state ~ time, id, data=mig_df, qmat=q0,
                opt.method="fisher", gen.inits=TRUE)
```

The marginal likelihood function for a given generator matrix, discrete-time interval T and corresponding discrete-time transition frequencies $\mathbf{N}_T$ can be computed by the function `ctmcdlogLik()`. Optimization with the previously employed remote initial value `gm0` fails for all **msm** optimization methods, with the closer built-in parameter initialization `gen.inits=TRUE`, we obtain the results presented in table 1. Optimization also fails for the method of Kalbfleisch and Lawless (1985). However, it is already mentioned in the helpfiles for the `msm()` function, that optimization using this approach lacks stability. Thus, the advantages of the EM algorithm in this specific data setting where no covariates are included in the calculation are its numerical performance and its stability.

Method	EM	optim	nlm	bobyqa
Log-Likelihood	-3194.255	-3194.255	-3194.255	-3194.259
Time Elapsed [s]	0.46	4.65	28.33	167.52

Table 1: Likelihood Optimization - Numerical Comparison

Confidence intervals

The package **ctmcd** also provides a function for deriving a confidence interval based on the asymptotic normality of the maximum likelihood estimate. As however in a partially observed data setting the maximum likelihood estimate is based on the likelihood function of the complete observations $\{N_{ij}(T), R_i(T)\}_{1\leq i\leq I, 1\leq j\leq J}$ given that only part of them, $\mathbf{N}_T$ are actually available, the Fisher information matrix has to be adjusted for the missing information in order to derive proper standard error estimates. Following Oakes (1999), a Fisher information matrix estimate for the observed data $\mathbf{I}_{\mathbf{N}_T}$ can be obtained by

$$\mathbf{I}_{\mathbf{N}_T} = \mathbf{I}_{\{N_{ij}(T),R_i(T)\}_{1\leq i\leq I,1\leq j\leq J}} - \mathbf{I}_{\{N_{ij}(T),R_i(T)\}_{1\leq i\leq I,1\leq j\leq J}|\mathbf{N}_T}.$$

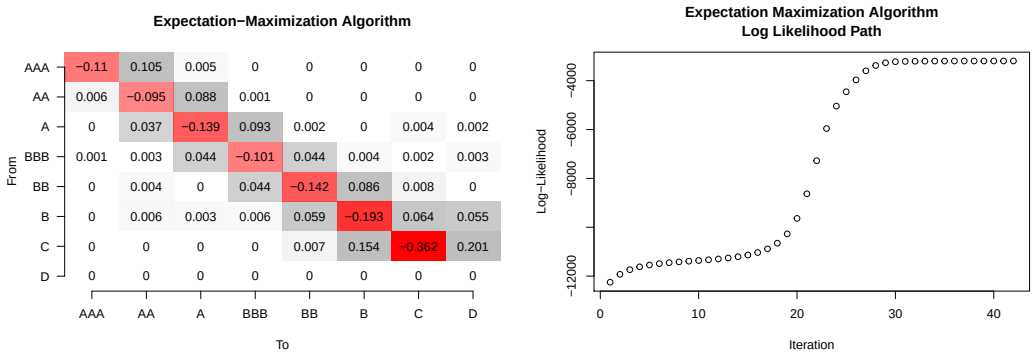


Figure 3: Maximum-Likelihood Estimation.

Bladt and Sørensen (2009) then concretize this expression for a generator matrix estimation framework and derive

$$\begin{aligned} I_{\mathbf{N}_T,(i-1)I+j,(i-1)I+j} =& \frac{1}{q_{ij}^2} E(N_{ij}(T)|\mathbf{Q}_I, s(0), s(T)) - \frac{1}{q_{ij}^2}\frac{\partial}{\partial q_{ij}} E(N_{ij}(T)|\mathbf{Q}_I, s(0), s(T)) \\ &+ \frac{\partial}{\partial q_{ij}} E(R_i(T)|\mathbf{Q}_I, s(0), s(T)) \end{aligned}$$

and $$I_{\mathbf{N}_T,(i-1)I+j,(i'-1)I+j'} = -\frac{1}{q_{ij}^2}\frac{\partial}{\partial q_{i'j'}} E(N_{ij}(T)|\mathbf{Q}_I, s(0), s(T)) + \frac{\partial}{\partial q_{i'j'}} E(R_i(T)|\mathbf{Q}_I, s(0), s(T))$$

as diagonal and off-diagonal $(i,j) \neq (i',j')$ elements of the observed Fisher information matrix $\mathbf{I}_{\mathbf{N}_T}$. The confidence interval then has the common form

$$q_{ij} \pm z_{1-\frac{\alpha}{2}} se(q_{ij}),$$

where $z_{1-\frac{\alpha}{2}}$ denotes the $1-\frac{\alpha}{2}$ quantile of the standard normal distribution. The method is implemented as a function `ciEM()`, which can be easily accessed using the generic `gmci()` command, which takes as arguments an EM algorithm estimate object and a significance level `alpha`.

```
ciem <- gmci(gmem, alpha=.05)
plot(ciem)
```

By default, the derivatives for the information matrix are calculated using the analytical expressions of Smith and dos Reis (2017) (`cimethod="SdR"`), numerical derivatives as suggested in Bladt and Sørensen (2009) can be accessed by `cimethod="BS"`.
The matrix plot function can also be applied to `"gmci"` interval estimate objects, see, e.g., figure 4. One can see in this example that interval estimates are not provided for all generator matrix entries. This is due to the fact that the numerical evaluation of the above described expressions for deriving the Fisher information matrix and inverting it becomes unstable when the parameter estimates are small.

Thus a lower limit eps can or has to be specified so that for generator matrix elements smaller than eps, no interval estimates are obtained. By default `eps=1e-04`.

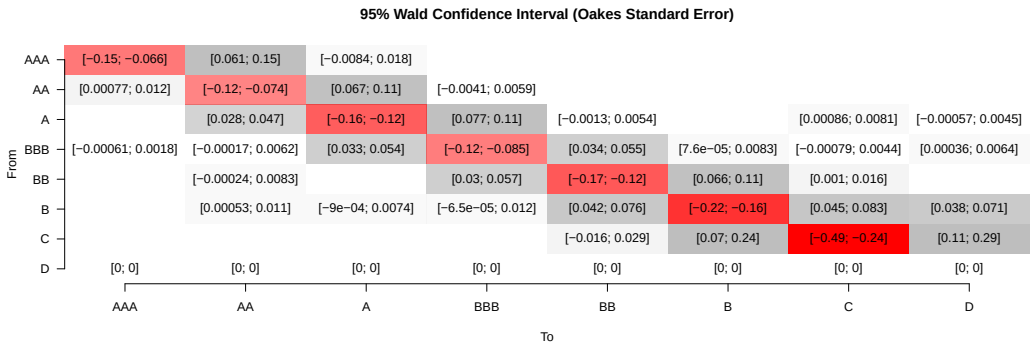


Figure 4: Confidence Interval

Bayesian inference

Gibbs sampler

Bladt and Sørensen (2005) show that the Gamma distribution $\Gamma(\phi, \psi)$ constitutes a conjugate prior for the off diagonal elements of the generator matrix under the continuous-time Markov chain likelihood function. The posterior distribution can then be derived as

$$f(\mathbf{Q}|\{s(0), s(T)\}) \propto L(\mathbf{Q}|\{s(0), s(T)\}) \prod_{i=1}^{I} \prod_{j \neq i} q_{ij}^{\phi_{ij}-1} \exp(-q_{ij}\psi_i)$$

$$\propto \prod_{i=1}^{I} \prod_{j \neq i} q_{ij}^{N_{ij}(T)+\phi_{ij}-1} \exp(-q_{ij}(R_i(T) + \psi_i)).$$

Based on these expressions, they describe a Gibbs sampling algorithm (GS) which in analogy to the EM algorithm iteratively simulates the missing data $N_{ij}(T)$ and $R_i(T)$ given the current parameter estimate and subsequently draws new parameter estimates given the imputed data. In order to use the method, the prior parameters have to be specified as a `"list"` object named `prior`. Thereby, the first element of the list has to be an $I \times J$ matrix of the Gamma parameters ϕ_{ij} and the second element a vector of length I with the parameters ψ_i. Consider, e.g.,

```
pr <- list()
pr[[1]] <- matrix(1, 8, 8)
pr[[1]][8,] <- 0
pr[[2]] <- c(rep(5, 7), Inf)
```

as a simple example, where $\phi_{ij} = 1$, $\psi_i = 5$ and there is an absorbing state 8, which can be specified by determining $\phi_{\cdot,8} = 0$ and $\psi_8 = \infty$. As for the EM algorithm we need to provide a matrix of *absolute*, rather than relative, transition frequencies as input data (in our example `tm=tm_abs`). Furthermore, the length of the burn-in period must be chosen (here: `burnin=1000`). Convergence of the algorithm is evaluated by the approach of Heidelberger and Welch (1981), which is implemented in the R package **coda**, see Plummer et al. (2006). The advantage of this method is that it can be applied to single chains; a shortcoming is that, as for similar methods, evaluation with multiple parameters is time consuming. Thus, besides specifying a p-value for the convergence test by the argument `conv_pvalue`, one can also set a frequency criterion `conv_freq` for how often with an equidistant number of trials convergence shall be checked. By default, `conv_pvalue=0.05` and `conv_freq=10`. One should notice that as the method of Heidelberger and Welch (1981) is a two sample location test for comparing the stability of the parameter estimates at the beginning and the end of the Markov chain, the hypotheses are set so that an increasing p-value implies a stricter convergence criterion. Another stopping rule is the maximum number of iterations `niter`, which by default is set as `niter=1e04`. If convergence according to the method of Heidelberger and Welch (1981) is not given before the maximum number of iterations is reached, a warning is displayed.

Setting the `method` argument to the value `"GS"` will then lead the generic generator matrix estimation method to provide a posterior mean estimate

```
gmgs <- gm(tm=tm_abs, te=1, method="GS", prior=pr, burnin=1000)
plot(gmgs)
```

The result can be seen in figure 5.

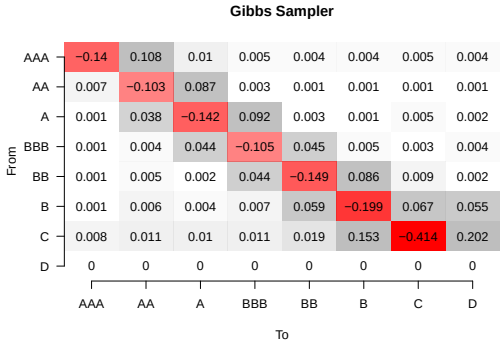


Figure 5: Posterior Mean Estimate

Endpoint-conditioned path sampling

Central to the Gibbs sampling algorithm is the sampling of realizations from the missing data full conditional distribution given the current parameters and the discrete time observations. This yields the sample paths from a continuous-time Markov chain with generator matrix estimate $\mathbf{Q}_l$ given initial and end states $s(0)$ and $s(T)$. In the package, two methods for deriving these sampling paths are provided, on the one hand the modified rejection sampling approach of Nielsen (2002) which can be accessed by `sampl_method="ModRej"`, on the other hand the uniformization sampling scheme of Fearnhead and Sherlock (2006), which is set as default method and can be manually employed by setting `sampl_method="Unif"`. As the simulation of trajectories of the process is in practice often very time consuming, the method is implemented in C++ based on the source code of Fintzi (2016) which itself is built upon the supplementary R code of Hobolth (2008). Figure 6 shows the time (in seconds) needed for sampling 10000 trajectories for each of the two methods and any combination of initial and endstates.

```
speedmat_modrej <- matrix(0, 8, 8)
speedmat_unif <- matrix(0, 8, 8)
tpm <- expm(gmgs$par)
for(i in 1:7){
  for(j in 1:8){
    elem <- matrix(0, 8, 8)
    elem[i,j] <- 1e5
    t0 <- proc.time()
    rNijTRiT_ModRej(elem, 1, gmgs$par)
    speedmat_modrej[i,j] <- (proc.time() - t0)[3]
    t0 <- proc.time()
    rNijTRiT_Unif(elem, 1, gmgs$par,tpm)
    speedmat_unif[i,j] <- (proc.time() - t0)[3]
  }
}

plotM(speedmat_modrej,
      main="Time for Simulation of 100,000 Paths\nModified Rejection Sampling",
      xnames=rownames(tm_abs), ynames=colnames(tm_abs))
plotM(speedmat_unif,
      main="Time for Simulation of 100,000 Paths\nUniformization Sampling",
      xnames=rownames(tm_abs), ynames=colnames(tm_abs))
```

Although in this example, the uniformization sampling approach clearly outperforms modified rejection sampling, it has to be mentioned that the computing time for deriving discrete time state transitions in the uniformization approach (which is carried out by using the matrix exponential) is not included, because it is only calculated once in each Gibbs sampling iteration step. Moreover, rejection rates of the approach of Nielsen (2002) might be lower if trajectories are sampled for a whole row

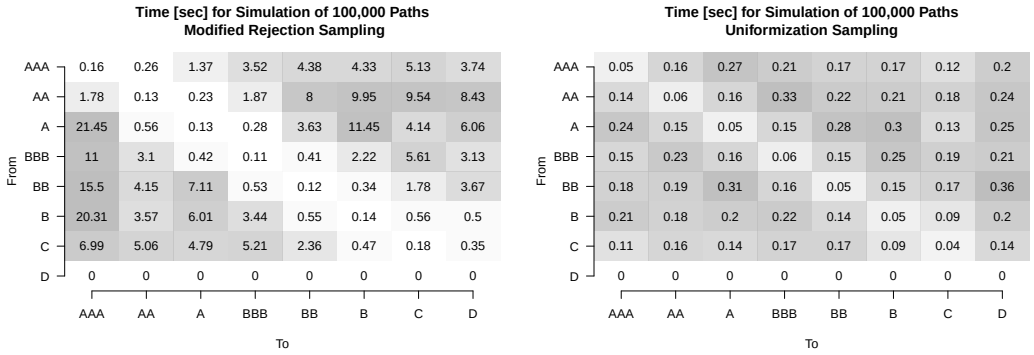


Figure 6: Endpoint-Conditioned Trajectory Sampling

of a transition matrix and not single initial and end state combinations because then more than one possible path ending can be accepted in a single draw. Therefore, to provide the option to combine the individual strengths of both approaches, it is possible to set `sampl_method="Comb"` and provide a matrix with entries of either `"M"` or `"U"` for the optional argument `combmat`, specifying which algorithm shall be used for simulating trajectories for the specific start and endpoint combination. Moreover, it is possible to include an own external sampling algorithm implementation by specifying `method="Ext"` and an argument `sampl_func` with a sampling function of the format

```
sf <- function(tmabs, te, gmest) {
  ### Derive Expected Holding Times (RiT) and Number of State Transitions (NijT)
  return(list(RiT=..., NijT=...))
}
```

Thereby, the matrix of absolute transition frequencies `tmabs`, the time interval for the discrete-time transitions `te` and the current parameter estimate of the Gibbs sampler `gmest` are input variables to this function and a vector of cumulative simulated holding times `RiT` and a matrix of continuous-time state changes `NijT` needs to be returned.

Parallelization

As the numerical performance of the Gibbs sampling algorithm is severely dependent on the performance of the endpoint conditioned path sampling algorithm, we would like to briefly point out that the whole method can also be run in parallel. This can be achieved by setting up a number of `nco` independent chains with `N/nco` iterations each, whereas `N` denotes the total number of iterations and `nco` the number of parallel threads. As long as the initial states of the chain are forgotten, the single generator matrix draws can be seen as independent realizations. Thus, a burnin period must be considered in every thread and `conv_pvalue` has to be set to 1 in order to ensure that the predetermined number of iterations is reached. Without loss of generality we use the packages **foreach** Microsoft and Weston (2017) and **doParallel** Corporation and Weston (2017) to provide with a simple example.

```
library(foreach)
library(doParallel)
N <- 1e5
nco <- detectCores()
cl <- makeCluster(nco)
registerDoParallel(cl)
gspar=foreach(i=1:nco, .packages=c("ctmcd", "expm")) %dopar%
  gm(tm=tm_abs, te=1, method="GS", burnin=1000, prior=pr,
     conv_pvalue=1, niter=N / nco)
stopCluster(cl)

### Derive Estimate
parlist <- lapply(gspar, function(x) x$par)
parest <- Reduce('+', parlist) / nco
```

In this multiple chain setting, convergence can be analyzed by the potential scale reduction factor diagnostic of Gelman and Rubin (1992). The potential scale reduction factor describes by what fraction the variance of the draws in the chain can be reduced if the chain is extended to an infinite length.

Convergence is assumed when the factor is close to 1. Rules of thumb on how close its value should be are, e.g., 1.2, see Bolker (2008) or 1.1, see Gelman et al. (2011). With the absorbing default state and diagonal elements being only a linear combination of the parameters in the single rows, chains for 49 parameters have to be evaluated. This can be conducted by employing the multivariate extension of the originally univariate factor, which has been introduced by Brooks and Gelman (1998).

```
### Check Convergence
library(coda)
chainlist <- as.mcmc.list(lapply(gspar, function(x) {
    as.mcmc(do.call(rbind, lapply(x$draws, as.vector)))
}))
parchainlist <- lapply(chainlist,
                       function(x) x[,as.vector(parest) > 0])
gelman.diag(parchainlist)
```

Employing the implementation of this diagnostic in the **coda** package, we derive a multivariate potential scale reduction factor of 1.01 in this example. Thus, convergence may be assumed.

Credibility intervals

After having discussed various aspects of point estimation, an example for Bayesian interval estimation shall be presented as well. Bladt and Sørensen (2009) show that equal-tailed credibility intervals can be easily obtained from samples of the joint posterior distribution by empirical quantiles

$$[q_{ij,\lceil L\frac{\alpha}{2}\rceil}, q_{ij,\lceil L\frac{1-\alpha}{2}\rceil}].$$

Calling `gmci()`, the generic function for generator matrix interval estimates, and specifying that an interval based on a Gibbs sampling object (here: `gmgs`) shall be derived will automatically call the `ciGS()` subroutine and yield an equal-tailed credibility interval as outlined above. Setting a confidence level `alpha=0.05` will then yield the estimate which can be seen in figure 7.

```
cigs <- gmci(gm=gmgs, alpha=0.05)
plot(cigs)
```

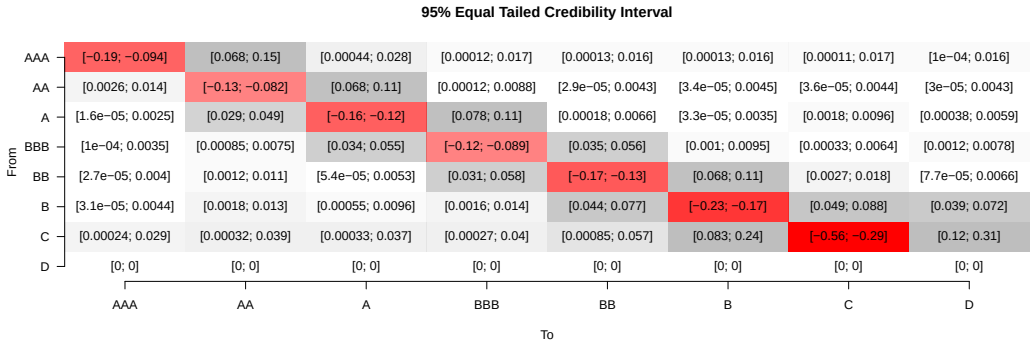


Figure 7: Credibility Interval

Comparison of Approaches

The approaches `"DA"`, `"WA"`, `"EM"` and `"GS"` are suitable for state spaces of size 2 or greater. However, with a state space of size 2, the approaches diagonal adjustment and weighted adjustment will always yield the same result as there is only one possible rearrangement of off-diagonal elements in this case. Method `"QO"` requires a state space of at least size 3 as two off diagonal elements for each row of a generator matrix are required to perform the implemented sorting scheme.

Figure 8 shows how the methods scale concerning numerical performance. The simulation study shows examples for the average time in seconds to perform an estimate with each of the methods outlined in this manuscript. Thereby, the EM algorithm and the Gibbs sampler are run for 100 iterations each and the data on which the estimates are performed is generated by distributing 1000 and respectively, 10000 discrete-time transitions over a single unit time horizon uniformly over

matrices of dimension 2×2 to 10×10. The estimation procedures are repeated 1000 times and the mean execution times are summarized in the graphics below. One can recognize that the matrix logarithm adjustment approaches require by far less computation time than the EM algorithm and the Gibbs sampler and that with increasing dimension of the state space, computing time is increasing as well. Moreover, one can also identify that the matrix logarithm adjustment approaches and the EM algorithm are - in contrast to the Gibbs sampler - almost only dependent on the size of the state space and not the number of discrete-time transitions in the sample.

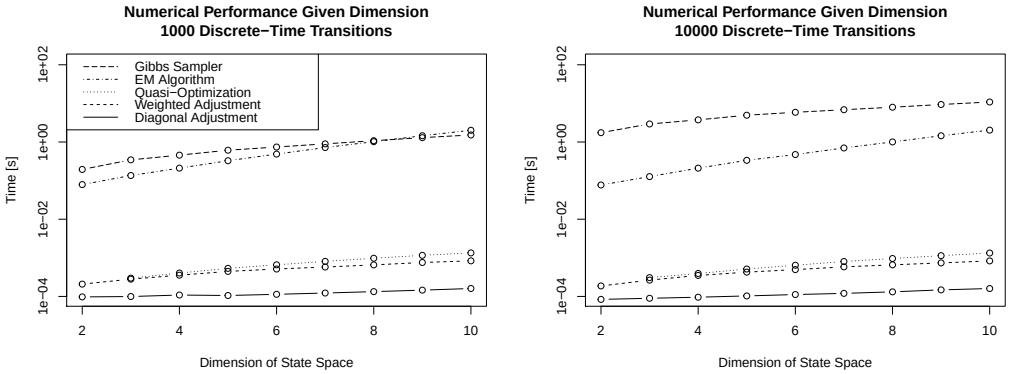


Figure 8: Speed Comparison

Profiles of Discrete-Time Transitions into Absorbing States

Based on the derived parameter estimates, predictions about discrete-time transitions into absorbing states of the Markov chain can be made and moreover, systemized as a function of the length of the discrete time interval. In the credit rating example shown in this paper, such functions are discrete time probabilities of default of a given initial rating category depending on the time horizon (in years) and shall be called probability of default profiles in the following. Thereby, an advantage of the Bayesian approach is that in contrast to the other methods outlined, interval estimates for such profiles can be easily derived. This involves the computation of empirical quantiles $p_{t_a,ij,\frac{\alpha}{2}}$ and $p_{t_a,ij,1-\frac{\alpha}{2}}$ of all elements of the discrete time transition matrix estimates

$$(\mathbf{P}_{t_a,1}, \ldots, \mathbf{P}_{t_a,L}) = (\exp(\mathbf{Q}_1 t_a), \ldots, \exp(\mathbf{Q}_L t_a))$$

given the single Gibbs sampling generator matrix draws $\mathbf{Q}_1, \ldots, \mathbf{Q}_L$, see Bladt and Sørensen (2009).

For the seven possible initial states in the credit rating example, probability-of-default profiles can then be obtained using the following code, the results are shown in figure 9.

```
tmax <- 20
for(cat in 1:7){
  absStvec <- sapply(1:tmax, function(t) expm(gmgs$par * t)[cat,8])
  quantMat <- matrix(0, 4, tmax + 1)
  for(t in 1:tmax){
    dtdraws <- lapply(gmgs$draws, function(x) expm(t * x))
    drawvec <- sapply(1:length(gmgs$draws), function(x) dtdraws[[x]][cat,8])
    quantMat[,t + 1] <- quantile(drawvec, c(.025, .05, .95, .975))
  }
  plot(0:tmax, c(0, absStvec), t="l", lwd=3, ylim=c(0, max(quantMat)),
       main=paste0("Absorbing State Profiles\nInitial Rating Category ",
                   rownames(tm_abs)[cat]),
       xlab="Time [Years]", ylab="Probability of Default")
  for(i in 1:4)
    lines(0:tmax, quantMat[i,], lty=c(3, 2, 2, 3)[i])
  legend("topleft", lty=c(3, 2, 1), c("95%", "90%", "Median"))
}
```

Complementary to the solid line, which represents the posterior mean estimate based discrete-time probability of default predictions, pointwise credibility intervals for $\alpha = 0.05$ and $\alpha = 0.1$ are computed here. As expected, one can recognize that with increasing time horizon, the width of the

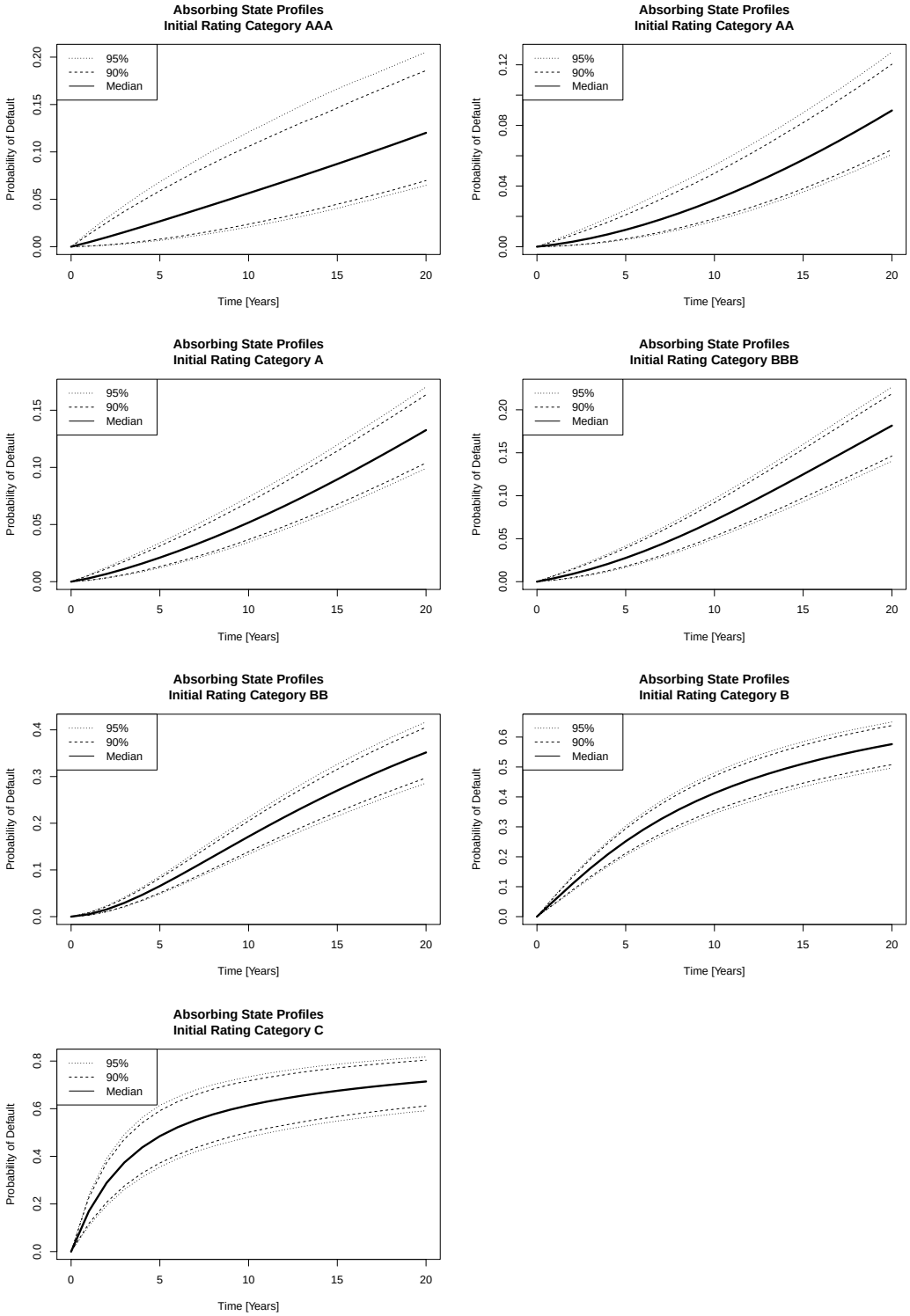


Figure 9: Probability-of-Default Profiles

intervals is increasing as well. Furthermore, the width of intervals increases with decreasing number of observations in the specific category. This can be seen, e.g., for the profile with pointwise intervals of initial rating category "AAA".

Summary

The problem of estimating the parameters of a continuous-time Markov chain from discrete-time data occurs in a wide range of applications and especially plays an important role in gene sequence data analysis and rating based credit risk modeling. This paper introduces and illustrates the **ctmcd** package, which provides an implementation of different approaches to derive such estimates. It supports matrix logarithm-based methods with diagonal and weighted adjustment as well as a quasi-optimization procedure. Moreover, maximum likelihood estimation is implemented by an instance of the EM algorithm and Bayesian estimates can be derived by a Gibbs sampling procedure. For the latter two approaches also interval estimates can be obtained. The Bayesian approach can be used to derive pointwise credibility intervals of discrete-time transition probabilities and systematic profiles of discrete-time transition probabilities into absorbing states given the corresponding time horizon. Above all, a matrix plot function is provided and can be used to visualize both point and interval estimates.

Acknowledgements

The author thanks two anonymous referees for their thorough review and highly appreciates the comments and suggestions which significantly contributed to increasing the quality of the R package **ctmcd** and this manuscript.

Bibliography

M. Bladt and M. Sørensen. Statistical inference for discretely observed Markov jump processes. *Journal of the Royal Statistical Society B*, 67(3):395–410, 2005. URL https://doi.org/10.1111/j.1467-9868.2005.00508.x. [p127, 130, 133]

M. Bladt and M. Sørensen. Efficient estimation of transition rates between credit ratings from observations at discrete time points. *Quantitative Finance*, 9(2):147–160, 2009. URL https://doi.org/10.1080/14697680802624948. [p132, 136, 137]

B. Bolker. *Ecological Models and Data in R*. Princeton University Press, 2008. [p136]

S. P. Brooks and A. Gelman. General methods for monitoring convergence of iterative simulations. *Journal of Computational and Graphical Statistics*, 7(4):434–455, 1998. [p136]

M.-H. Chen et al. *Bayesian Phylogenetics: Methods, Algorithms, and Applications*. CRC Press, 2014. URL https://doi.org/10.1093/sysbio/syv063. [p127]

M. Corporation and S. Weston. *doParallel: Foreach Parallel Adaptor for the 'parallel' Package*, 2017. URL https://CRAN.R-project.org/package=doParallel. R package version 1.0.11. [p135]

T. Cubitt et al. The complexity of relating quantum channels to master equations. *Communications in Mathematical Physics*, 310(2):383–418, 2012. URL https://doi.org/10.1007/s00220-011-1402-y. [p127]

J. R. Cuthbert. On uniqueness of the logarithm for Markov semi-groups. *Journal of the London Mathematical Society*, s2-4(4):623–630, 1972. [p128]

J. R. Cuthbert. The logarithm function for finite-state Markov semi-groups. *Journal of the London Mathematical Society*, s2-6(3):524–532, 1973. [p128]

G. Dorfleitner and C. Priberny. A quantitative model for structured microfinance. *The Quarterly Review of Economics and Finance*, 53(1):12–22, 2013. URL https://doi.org/10.1016/j.qref.2012.10.005. [p127]

G. Elfving. Zur Theorie der Markoffschen Ketten. *Acta Societatis Scientiarum Fennicae*, A.2(8):1–17, 1937. [p127]

S. N. Ethier and T. G. Kurtz. *Markov Processes: Characterization and Convergence*. Wiley, 2005. URL https://doi.org/10.1002/9780470316658. [p127]

P. Fearnhead and C. Sherlock. An exact Gibbs sampler for the Markov-modulated Poisson process. *Journal of the Royal Statistical Society B*, 68:767–784, 2006. URL https://doi.org/10.1111/j.1467-9868.2006.00566.x. [p134]

J. Fintzi. ECctmc: Simulation from endpoint-conditioned continuous time Markov chains. CRAN, 2016. [p134]

A. Gelman and D. B. Rubin. Inference from iterative simulation using multiple sequences. *Statistical Science*, pages 457–472, 1992. URL https://doi.org/10.1214/ss/1177011136. [p135]

A. Gelman et al. Inference from simulations and monitoring convergence. *Handbook of Markov Chain Monte Carlo*, pages 163–174, 2011. URL https://doi.org/10.1201/b10905-7. [p136]

P. Heidelberger and P. D. Welch. A spectral method for confidence interval generation and run length control in simulations. *Communications of the ACM*, 24(4):233–245, 1981. URL https://doi.org/10.1145/358598.358630. [p133]

A. Hobolth. A Markov chain Monte Carlo expectation maximization algorithm for statistical analysis of DNA sequence evolution with neighbor-dependent substitution rates. *Journal of Computational and Graphical Statistics*, 17(1):1–25, 2008. URL https://doi.org/10.1198/106186008X289010. [p134]

A. Hobolth and E. A. Stone. Simulation from endpoint-conditioned, continuous-time Markov chains on a finite state space, with applications to molecular evolution. *Annals of Applied Statistics*, 3(3): 1204–1231, 2009. URL https://doi.org/10.1214/09-AOAS247. [p127]

M. Hughes and R. Werner. Choosing Markovian credit migration matrices by nonlinar optimization. *Risks*, 4(3):31, 2016. URL https://doi.org/10.3390/risks4030031. [p127]

Y. Inamura. Estimating continuous time transition matrices from discretely observed data. *Bank of Japan Working Paper Series*, 2006. URL https://www.boj.or.jp/en/research/wps_rev/wps_2006/data/wp06e07.pdf. [p129, 130]

R. B. Israel et al. Finding generators for Markov chains via empirical transition matrices, with applications to credit ratings. *Mathematical Finance*, 11(2):245–265, 2001. URL https://doi.org/10.1111/1467-9965.00114. [p127, 128, 129]

C. H. Jackson et al. Multi-state models for panel data: the msm package for R. *Journal of Statistical Software*, 38(8):1–29, 2011. URL https://doi.org/10.18637/jss.v038.i08. [p128]

J. Kalbfleisch and J. F. Lawless. The analysis of panel data under a Markov assumption. *Journal of the American Statistical Association*, 80(392):863–871, 1985. URL https://doi.org/10.1080/01621459.1985.10478195. [p131]

E. Kreinin and M. Sidelnikova. Regularization algorithms for transition matrices. *Algo Research Quarterly*, 4(1):23–40, 2001. [p127, 129]

Microsoft and S. Weston. *foreach: Provides Foreach Looping Construct for R*, 2017. URL https://CRAN.R-project.org/package=foreach. R package version 1.4.4. [p135]

R. Nielsen. Mapping mutations on phylogenies. *Systematic Biology*, 51(5):729–739, 2002. URL https://doi.org/10.1080/10635150290102393. [p134]

J. R. Norris. *Markov Chains*. Cambridge University Press, 1998. URL https://doi.org/10.1017/CBO9780511810633. [p127]

D. Oakes. Direct calculation of the information matrix via the EM algorithm. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 61(2):479–482, 1999. URL https://doi.org/10.1111/1467-9868.00188. [p127, 132]

M. Pfeuffer. Generator matrix approximation based on discrete-time rating migration data. Master Thesis, Ludwig Maximilian University of Munich, 2016. [p128]

M. Plummer, N. Best, K. Cowles, and K. Vines. CODA: convergence diagnosis and output analysis for MCMC. *R news*, 6(1):7–11, 2006. [p133]

M. J. Powell. The BOBYQA algorithm for bound constrained optimization without derivatives. *Cambridge NA Report NA2009/06, University of Cambridge, Cambridge*, 2009. [p131]

B. Singer and S. Spilerman. The representation of social processes by Markov models. *American Journal of Sociology*, 82(1):1–54, 1976. [p128]

G. Smith and G. dos Reis. Robust and consistent estimation of generators in credit risk. *Quantitative Finance (To Appear)*, 2017. URL https://doi.org/10.1080/14697688.2017.1383627. [p132]

J. M. Speakman. Two markov chains with a common skeleton. *Zeitschrift für Wahrscheinlichkeitstheorie und Verwandte Gebiete*, 7(3):224–224, 1967. [p128]

Standard and Poor's. Global corporate rating transitions, 2000. URL https://cerep.esma.europa.eu/cerep-web/statistics/transitionMatrice.xhtml. [p128]

C. van Loan. Computing integrals involving the matrix exponential. *IEEE Transactions on Automatic Controls*, AC 23(3):395–404, 1978. [p130]

K. L. Verbyla et al. The embedding problem for Markov models of nucleotide substitution. *PloS one*, 8 (7):e69187, 2013. URL https://doi.org/10.1371/journal.pone.0069187. [p127]

T. Yavin et al. Transition probability matrix methodology for incremental risk charge. *Journal of Financial Engineering*, 1(1):1450010, 2014. URL https://doi.org/10.1142/S234576861450010X. [p127]

M. Zhang and D. S. Small. Effect of vitamin A deficiency on respiratory infection: Causal inference for a discretely observed continuous-time non-stationary Markov process. *Canadian Journal of Statistics*, 40(4):646–662, 2012. URL https://doi.org/10.1002/cjs.11161. [p127]

Marius Pfeuffer
Department of Statistics and Econometrics
University of Erlangen-Nuremberg
Lange Gasse 20, 90403 Nuremberg
Germany
marius.pfeuffer@fau.de

Furniture for Quantitative Scientists

by Tyson S. Barrett and Emily Brignone

Abstract A basic understanding of the distributions of study variables and the relationships among them is essential to inform statistical modeling. This understanding is achieved through the computation of summary statistics and exploratory data analysis. Unfortunately, this step tends to be under-emphasized in the research process, in part because of the often tedious nature of thorough exploratory data analysis. The `table1()` function in the **furniture** package streamlines much of the exploratory data analysis process, making the computation and communication of summary statistics simple and beautiful while offering significant time-savings to the researcher.

Introduction

A major, but often overlooked, aspect of the research process is the computation of summary statistics and exploratory data analysis. Through exploratory data analysis, researchers can begin to understand patterns and relationships in their data that are relevant for more complex modeling schemes. In addition, this exploration can inform future directions in their research area and can inspire better research questions. This is a valuable benefit of assessing summary statistics and exploring the data, for, as John Tukey (1980) stated, "Finding the question is often more important than finding the answer."

Generally, an important aspect of data exploration is the assessment of summary statistics. Yet, the computation and reporting of summary statistics can be tedious and messy, with many researchers computing and entering values into tables one variable at a time. This can be quite time-consuming and leaves the researcher susceptible to data-entry errors. Additionally, in the event of changes to the number of observations in the sample or to the nature of the stratifying variable, the whole table must be manually updated. These issues may lead researchers to conduct only minimal exploratory data analysis, potentially missing important relationships among study variables. Researchers may also opt to delegate the computation of summary statistics to research assistants. This is not always ideal, as assistants may lack the substantive expertise required to recognize important or theoretically interesting patterns in the data.

The presentation of sample summary statistics in conjunction with higher-level data analysis is also essential to the production of reproducible research.[1] In many fields there is a struggle to produce reproducible research (Open Science Collaboration, 2015; Chang and Li, 2015; Begley and Ioannidis, 2015). Thorough reporting of sample characteristics, including the distribution of sample characteristics stratified by key study variables, allows other researchers the opportunity to more carefully evaluate findings, replicate results, discover further research avenues, and critically compare findings across multiple studies. In the **furniture** package, as will be demonstrated, obtaining descriptive statistics for all study variables, either with or without a stratifying variable, is simple and can be done using just a few lines of code.

It is worth noting that there are other well-designed packages that help produce descriptive statistics. The package **tableone** is thoroughly developed and performs similar analyses to `table1()`. However, the syntax is more complex and lacks some of the flexibility that `table1()` offers. The **stargazer** and **psych** packages also offer informative summaries of the data for each variable, but are limited in their use for publication without manual entry, and offer few options for more involved data manipulation.

Data for example analyses

In order to demonstrate the utility of `table1()`, we descriptively explored and analyzed data from the National Health and Nutrition Examination Survey (NHANES) provided by the Centers for Disease Control and Prevention (National Center for Health Statistics, 2016). The data are attached in the package as "nhanes_2010" after having been cleaned using the **furniture**, **tidyverse** and **foreign** R packages (Barrett and Brignone, 2016; Wickham, 2016; R Core Team, 2016). The data contain information on general health, activity level, age, gender, drug use, and chronic conditions for 1,417 young adults (51.4% female) aged 18-30 (M = 23.3, SD = 3.96). Under the direction of the Centers for Disease Control and Prevention, the data were collected via a complex sampling strategy across the United States in 2013 and 2014.

[1] We use the definitions from Goodman et al. (2016): "methods reproducibility" refers to the ability to obtain the same results with the same methods and data and "results reproducibility" means being able to obtain similar results using the same methods with independent data.

Table 1

The general form of `table1()`[2] is below, with several of the most common arguments. Note that the structure is similar to other "tidy" packages, with the `data.frame` as the first argument, unquoted variables, and the ability to pipe.

```
library(furniture)
table1(df,
       var1, var2, var3, etc.,
       splitby = ~stratifying_variable,
       test = TRUE,
       type = "pvalues",
       second = NULL,
       output = "text",
       FUN = NULL,
       FUN2 = NULL)
```

where

- `df` is the `data.frame` or a `tbl_df` from the tidyverse of functions,
- `var1`, etc. are unquoted variable names (any number of variables),
- `splitby` is a one-sided formula with a variable name,
- `test` is logical (i.e. `TRUE` or `FALSE`) indicating whether bivariate tests of association should be run,
- `type` allows the p-values and/or test statistics to be displayed, along with allowing a simplification of the output of the table,
- `second` is an optional vector of quoted variable names that, instead of means and SDs, are summarized by medians and the inter-quartile range (or other user-defined statistics),
- `output` allows several outputs for the table, including Latex and Markdown,
- `FUN` provides the user with the ability to apply user-defined functions to summarize numeric variables (any function that works with `tapply()`), and
- `FUN2`, similarly to `FUN`, allows the user to define a function to apply to all variables listed in the `second` argument above.

Notably, if no variables are listed, all variables in the `data.frame` will be summarized. Other options exist that can be used to better format and adjust the table, some of which we highlight in subsequent sections (e.g. `format_number` and `var_names`). However, in its simplest form, `table1()` only requires the data frame.

```
table1(df)
```

Yet, the function is designed for much more.

The function is built on fast base functions and is appropriate for even very large data sets ($n > 1,000,000$). It uses simple non-standard evaluation which allows the user to create and modify variables from within the function itself. For example, we could summarize levels of a dichotomous version of `var1` as seen below:

```
table1(df,
       ifelse(var1 > median(var1), 1, 0),
       splitby = ~stratifying_variable,
       test = TRUE)
```

Other simple modifications are also possible (e.g., `factor(var1)`, `var1*100`).

Exploratory data analysis with Table 1

The `table1()` function below is used on the NHANES data to explore differences in demographic and psychosocial factors between individuals who have and have not been informed by a doctor that he or she is overweight. The following code uses the `data.frame` named `d1`, selects 12 variables, stratifies by whether the individual was designated by a doctor as overweight (`splitby = ~overweight`), calls for tests of bivariate associations (`test = TRUE`), and is printed in the "text2" format (`output = "text2"`).

[2]The analyses for this paper used **furniture** 1.5.4 and R version 3.4.1.

```
table1(d1,
       gender, age, active, marijuana, illicit,
       down, sleeping, low_energy, appetite, feel_bad,
       dead, difficulty,
       splitby = ~overweight,
       test = TRUE,
       output = "text2")
```

The preceding code produces the following table. Note that, for space, several rows were excluded at the bottom of the table.

```
|=====================================================|
                          overweight
                Yes            No              P-Value
 -              ---            --              -------
 Observations   335            1082
 gender                                        0.004
    Male        139 (41.5%)    549 (50.7%)
    Female      196 (58.5%)    533 (49.3%)
 age                                           0.100
                23.65 (3.99)   23.24 (3.95)
 active                                        0.251
                151.36 (94.21) 166.83 (106.12)
 marijuana                                     0.333
    Yes         181 (59.9%)    535 (56.6%)
    No          121 (40.1%)    411 (43.4%)
 ...            ...            ...             ...
|=====================================================|
```

The table provides the n for each group, the means and standard deviations (SD) by group for each numeric variable, and the counts and percentages by group for each factor variable. The table can also provide bivariate statistical comparisons using the supplied stratifying variable as the independent variable. For categorical variables, a χ^2 test is performed. For numeric variables, either a t-test or an analysis of variance is performed (potentially adjusting for heterogeneity of variance), depending on the number of levels of the stratifying variable. Only the tests' p-values are shown by default. The user may also request the specific test statistics (`type = "full"`) or simply stars representing significance (`type = "stars"`).

If means and SDs are insufficient for a numeric variable, for example in the case of a highly skewed numeric variable, medians and the interquartile range (IQR) can be produced through the use of the "second" argument. This argument accepts quoted names of numeric variables, and for those variables, returns median and IQR in place of means and SDs by default.

```
table1(d1,
       gender, age, active, marijuana,
       splitby = ~overweight,
       test = TRUE,
       output = "text2",
       second = c("age", "active"))
```

```
|=================================================|
                         overweight
                Yes          No            P-Value
 -              ---          --            -------
 Observations   335          1082
 gender                                     0.004
    Male        139 (41.5%)  549 (50.7%)
    Female      196 (58.5%)  533 (49.3%)
 age                                        0.1
                23.00 [7.0]  23.00 [7.0]
 active                                     0.251
                120.00 [90.0] 135.00 [142.5]
 marijuana                                  0.333
    Yes         181 (59.9%)  535 (56.6%)
    No          121 (40.1%)  411 (43.4%)
|=================================================|
```

The medians and IQRs are differentiated from the means and SDs by the square brackets that are applied to the IQRs. This keeps the table clean but provides information on the types of statistics being presented.

Beneficially, the user can define a function to use in place of the default means/SDs and medians/IQRs. for numeric variables. This is through the use of the `FUN` and `FUN2` arguments. `FUN` applies to any variable not listed in the `second` argument. It can be as simple as `FUN = min`—asking for the minimum of each variable–or as complicated of a function as tapply will accept. For an example of a multi-statistic function, see below.

```
table1(d1,
       gender, age, active,
       splitby = ~overweight,
       test = TRUE,
       output = "text2",
       FUN = function(x) paste0(min(x, na.rm=TRUE), ", ", max(x, na.rm=TRUE)))
```

The preceding code allows the anonymous function [i.e. `function(x) paste0(min(x,...))`], to produce the minimum and maximum, be applied to each of the numeric variables (i.e. in this case, age and active). This code produces the following table.

```
|================================================|
                           overweight
                 Yes         No          P-Value
 -               ---         --          -------
 Observations    335         1082
 gender                                  0.004
    Male         139 (41.5%) 549 (50.7%)
    Female       196 (58.5%) 533 (49.3%)
 age                                     0.1
                 18, 30      18, 30
 active                                  0.251
                 30, 505     20, 840
|================================================|
```

In the same way, `FUN2` allows the user to specify another function to be applied to the indicated numeric variables within the same table. It also allows for simple formatting changes to these statistics as well (e.g. can insert brackets, commas, or change the rounding of the numbers). In essence, this makes `table1()` useful for a wide variety of situations.

We can also simplify and condense the table. As demonstrated below, `type = "simple"` produces only percentages for the categorical variables (instead of counts and percentages) and `type = "condense"` displays the summary only for the reference category of dichotomous variables and removes much of the white space. Together, they provide a much more succinct table.

```
table1(d1,
       gender, age, active, marijuana, illicit, down,
       splitby = ~overweight,
       test = TRUE,
       output = "text2",
       type = c("simple", "condense"))
```

```
|=====================================================|
                            overweight
                 Yes            No              P-Value
 -               ---            --              -------
 Observations    335            1082
 gender: Female  58.5%          49.3%           0.004
 age             23.65 (3.99)   23.24 (3.95)    0.1
 active          151.36 (94.21) 166.83 (106.12) 0.251
 marijuana: No   40.1%          43.4%           0.333
 illicit: No     89.7%          88.4%           0.584
 down                                           <.001
    No           69.7%          81.1%
    Several Days 21.4%          14.8%
    Majority     5.6%           2.8%
    Everyday     3.3%           1.3%
|=====================================================|
```

Finally, with the rise of "piping" (Wickham, 2016), we have integrated functionality that allows the table to be part of a bigger pipeline. When in a pipeline, the function auto-detects the pipe, prints the table and invisibly returns the original data so that it can continue to be used in subsequent functions. For example,

```
d1 %>%
  filter(age > 20) %>%
  table1(gender, age, active, marijuana, illicit, down,
         splitby = ~overweight,
         test = TRUE,
         output = "text2",
         type = c("simple", "condense")) %>%
  lm(overweight ~ age + gender, data = .)
```

In the end, these simple tables provide several pieces of important information. Without needing any advanced modeling, we already have an idea of several relationships among the study variables. For example, several demographic and psychosocial factors are related to the designation by a doctor as being overweight, including trouble sleeping and feeling down. These insights can inform model building and follow-up visualizations. `table1()`, in conjunction with other summary techniques (e.g. the base function `summary` in R), provides a quick and broad understanding of the patterns and relationships in the data.

Easy communication of descriptive statistics

In addition to the power of exploratory data analysis in informing statistical modeling, `table1()` is an important tool in scientific communication. For this reason, it was equipped with several output formatting features.[3] Five that are particularly useful are discussed below.

1. `output` allows for two regular console outputs (i.e., "text" and "text2") and all `knitr::kable` options, including "latex", "markdown" and "html."
2. `var_names` allows the user to provide a list of names that will replace the variable names in the table.
3. `format_number` provides formatting of numbers with commas (e.g., 22,000 instead of 22000) when set to `TRUE`.
4. `type`, in addition to the "simple" and "condense" options discussed previously, provides three options for the presentation of statistical tests: 1) "pvalues", which is default, and displays the p-values for the tests of association, 2) "stars" which provides the common star notation for p-values, and 3) "full" which provides the test statistics and the p-values.[4]

These, among other options, allow for the production of quality tables that can be easily published via various mediums including peer-reviewed journals and online webpages.

The example below illustrates the simplicity of communicating the basic statistics of the NHANES sample in Table 1.

```
  table1(d1,
         gender, age, active, marijuana, illicit,
         down, sleeping, dead,
         splitby = ~overweight,
         var_names = c("Gender", "Age", "Activity (minutes)", "Marijuana",
                       "Other Illicit Drug", "Feeling Down",
                       "Trouble Sleeping", "Wish Were Dead"),
         format_number = TRUE,
         type = "simple",
         test = TRUE,
         output = "latex")
```

With only a few lines of code, simple yet important information about the sample, study variables, and their bivariate relationships to a key variable are elegantly produced for easy dissemination. With minor touching up, this table can be ready for professional and peer-reviewed publications.[5]

[3] For a list of all argument options, see Barrett and Brignone (2016).

[4] Only the "pvalues" option works when `"simple"` or `"condense"` are also included.

[5] The splitby variable name at the top of Table 1—"Overweight"—was added afterwards. This information does not print automatically in the `kable` output.

Table 1: Latex table produced from `table1()` with a number of formatting options.

	Overweight		
	Yes	No	P-Value
Observations	335	1,082	
Gender			0.004
– Male –	41.5%	50.7%	
– Female –	58.5%	49.3%	
Age			0.1
	23.65 (3.99)	23.24 (3.95)	
Activity (minutes)			0.251
	151.36 (94.21)	166.83 (106.12)	
Marijuana			0.333
– Yes –	59.9%	56.6%	
– No –	40.1%	43.4%	
Other Illicit Drug			0.584
– Yes –	10.3%	11.6%	
– No –	89.7%	88.4%	
Feeling Down			<.001
– No –	69.7%	81.1%	
– Several Days –	21.4%	14.8%	
– Majority –	5.6%	2.8%	
– Everyday –	3.3%	1.3%	
Trouble Sleeping			<.001
– No –	53%	64.7%	
– Several Days –	29.6%	23.1%	
– Majority –	7.6%	6.8%	
– Everyday –	9.9%	5.4%	
Wish Were Dead			0.02
– No –	94.4%	97.9%	
– Several Days –	3.3%	1.3%	
– Majority –	1.3%	0.5%	
– Everyday –	1%	0.3%	

For those using processors other than latex or markdown, the table can exported to a spreadsheet program, such as Microsoft's Excel, via the `export` argument. Here, all that needs to be provided is a string that will be the outputted file name (e.g., "myfile"). This will export the table as a formatted CSV to a new folder in the working directory called "Table1." From there, the table can be imported into a word processor, such as Microsoft's Word. Another, more manual option is simply copying-and-pasting from the console output into a spreadsheet. Using the "text to columns" feature common in spreadsheet programs, this table can become ready to import into a word processor. Regardless of the approach, the common errors of manually inputting values into a table are greatly reduced.

Additional features

In addition to `table1()`, the **furniture** package contains a data cleaning function (i.e. `washer()`) and an operator (i.e. `%xt%`). `washer()` is used to replace values in a vector with other values in a way that avoids more complex `ifelse` statements. The `%xt%` operator can be used for simple cross-tabulations among two categorical variables. These, although not discussed much here, are helpful in getting the data ready for more in depth analysis in `table1()`. In fact, both were used to get the data in the format found in the package. More information can be found in the package documentation and at `tysonstanley.github.io`.

Summary

Ultimately, each function in the **furniture** package is designed to simplify important data exploration with the long-term goal of increasing both methods and results reproducibility. With the `table1()` function, the computation and communication of descriptive statistics is made simple and beautiful, streamlining the exploratory data analysis process. We hope that the user-friendly syntax and polished output enables researchers to efficiently perform more thorough exploration of their data and to more

easily communicate their findings.

Bibliography

T. Barrett and E. Brignone. *furniture: Furniture for Applied Quantitative Researchers*, 2016. URL https://CRAN.R-project.org/package=furniture. R package version 1.5.0. [p142, 146]

C. G. Begley and J. P. A. Ioannidis. Reproducibility in science: Improving the standard for basic and preclinical research. *Circulation Research*, 116(1):116–126, 2015. ISSN 15244571. doi: 10.1161/CIRCRESAHA.114.303819. [p142]

A. C. Chang and P. Li. Is Economics Research Replicable? Sixty Published Papers from Thirteen Journals Say "Usually Not". *Finance and Economics Discussion Series*, 083:1–26, 2015. ISSN 19362854. doi: 10.17016/FEDS.2015.083. [p142]

S. N. Goodman, D. Fanelli, and J. P. A. Ioannidis. What does research reproducibility mean? *Science Translational Medicine*, 8(341):1–6, 2016. ISSN 1946-6234. doi: 10.1126/scitranslmed.aaf5027. [p142]

National Center for Health Statistics. National Health and Nutrition Examination Survey Data. Technical report, U.S. Department of Health and Human Services, Centers for Disease Control and Prevention, Hyattsville, MD, 2016. URL http://www.cdc.gov/nchs/nhanes/. [p142]

Open Science Collaboration. Estimating the reproducibility of psychological science. *Science*, 349 (6251):aac4716–aac4716, 2015. ISSN 0036-8075. doi: 10.1126/science.aac4716. URL http://science.sciencemag.org/content/349/6251/aac4716. [p142]

R Core Team. *foreign: Read Data Stored by Minitab, S, SAS, SPSS, Stata, Systat, Weka, dBase, ...*, 2016. URL https://CRAN.R-project.org/package=foreign. R package version 0.8-67. [p142]

J. Tukey. We Need Both Exploratory and Confirmatory. *The American Statistician*, 34(1):79–88, 1980. [p142]

H. Wickham. *tidyverse: Easily Install and Load 'Tidyverse' Packages*, 2016. URL https://CRAN.R-project.org/package=tidyverse. R package version 1.0.0. [p142, 146]

Tyson S. Barrett
Utah State University
2810 Old Main Hill, Logan, UT, 84322
U.S.A
tyson.barrett@usu.edu

Emily Brignone
VA Pittsburgh Healthcare System
Pittsburgh, PA 15240
U.S.A
emilybrignone@gmail.com

BayesBD: An R Package for Bayesian Inference on Image Boundaries

by Nicholas Syring and Meng Li

Abstract

We present the **BayesBD** package providing Bayesian inference for boundaries of noisy images. The **BayesBD** package implements flexible Gaussian process priors indexed by the circle to recover the boundary in a binary or Gaussian noised image. The boundary recovered by **BayesBD** has the practical advantages of guaranteed geometric restrictions and convenient joint inferences under certain assumptions, in addition to its desirable theoretical property of achieving (nearly) minimax optimal rate in a way that is adaptive to the unknown smoothness. The core sampling tasks for our model have linear complexity, and are implemented in C++ for computational efficiency using packages **Rcpp** and **RcppArmadillo**. Users can access the full functionality of the package in both the command line and the corresponding **shiny** application. Additionally, the package includes numerous utility functions to aid users in data preparation and analysis of results. We compare **BayesBD** with selected existing packages using both simulations and real data applications, demonstrating the excellent performance and flexibility of **BayesBD** even when the observation contains complicated structural information that may violate its assumptions.

Introduction

Boundary estimation is an important problem in image analysis with wide-ranging applications from identifying tumors in medical images (Li et al., 2010), classifying the process of machine wear by analyzing the boundary between normal and worn materials (Yuan et al., 2016), to identifying regions of interest in satellite images, such as the boundary of Scotland's Lake Menteith (Cucala and Marin, 2014; Marin and Robert, 2014). Furthermore, boundaries present in epidemiological or ecological data may reflect the progression of a disease or an invasive species; see Waller and Gotway (2004), Lu and Carlin (2005), and Fitzpatrick et al. (2010).

There is a rich literature on image segmentation for both noise-free and noisy observations; see the surveys in Ziou and Tabbone (1998); Basu (2002); Maini and Aggarwal (2009); Bhardwaj and Mittal (2012), and particularly the Bayesian approaches in Hurn et al. (2003) and Grenander and Miller (2007). Recently, Li and Ghosal (2015) developed a flexible nonparametric Bayesian model to detect image boundaries, which achieved four aims of guaranteed geometric restriction, (nearly) minimax optimal rate adaptive to the smoothness level, convenience for joint inference, and computational efficiency. However, despite the theoretical soundness, the practical implementation of Li and Ghosal's method is far from trivial, mostly in the approachability of the proposed nonparametric Bayesian framework and further improvement in the speed of posterior sampling algorithms, which becomes critical in attempts to popularize this approach in statistics and the broader scientific community. In this paper, we present the R package **BayesBD** (Syring and Li, 2017) which aims to fill this gap. The developed **BayesBD** package provides support for analyzing binary images and Gaussian-noised images, which commonly arise in many applications. We implement various options for posterior calculation including the Metropolis-Hastings sampler (Hastings, 1970) and slice sampler (Neal, 2003). To further speed up the Markov Chain Monte Carlo (MCMC), we take advantage of the integration via **RcppArmadillo** (Eddelbuettel, 2013; Eddelbuettel and Sanderson, 2014) of R and the compiled C++ language. We further integrate the **BayesBD** package with **shiny** (RStudio, Inc, 2016) to facilitate the usage of implemented boundary detection methods in real applications.

A far as we know, there are no other R packages for image boundary detection problems achieving the four goals mentioned above. An earlier version of the **BayesBD** package (Li, 2015) provided first-of-its-kind tools for analyzing images, but support for Gaussian-noised images, C++ implementations, more choices of posterior samplers, and **shiny** integrations were not available until the current version. For example, the nested loops required for MCMC sampling were inefficient in R programming. The combination of new programming and faster sampling algorithms means that a typical simulation example consisting of 5000 posterior samples from $10,000$ data points can now be completed in about one minute.

The rest of the paper is organized as follows. We first introduce the problem of statistical inference on boundaries of noisy images, the nonparametric Bayesian models in use, and posterior sampling algorithms. We then demonstrate how to use the main functions of the package for data analysis working with both the command line and **shiny**. We next conduct a comprehensive experiment on the comparison of sampling methods and coding platforms, scalability of **BayesBD**, and comparisons

with existing packages including **mritc**, **bayesImageS**, and **bayess**. We illustrate a pair of real data analyses of medical and satellite images. The paper is concluded by a Summary section.

Statistical analysis of image boundaries

Image data

An image observed with noise may be represented by a set of data points or pixels $(Y_i, X_i)_{i=1}^n$, where the intensities Y_i are observed at locations $X_i \in \mathcal{X} = [0,1]^2$. Following Li and Ghosal (2015), we assume that there is a closed region $\Gamma \in \mathcal{X}$ such that the intensities Y_i are distributed as follows conditionally on whether its location is inside Γ or in the background Γ^c:

$$Y_i \sim \begin{cases} f(\cdot;\xi) & X_i \in \Gamma; \\ f(\cdot;\rho) & X_i \in \Gamma^c, \end{cases} \tag{1}$$

where f is a given probability mass function or probability density function of a parametric family up to unknown parameters (ξ, ρ). For example, Figure 1 shows two simulated images where the parametric family is Bernoulli and Gaussian, respectively. These images can be reproduced using the functions `par2obs`, `parnormobs`, and `plotBD` which will be demonstrated in detail later on.

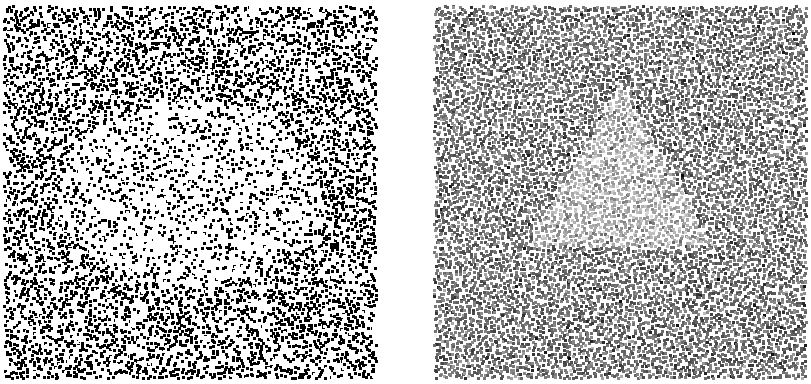


Figure 1: Left: a binary image generated using an elliptical boundary and parameters $\pi_1 = 0.65$ and $\pi_2 = 0.35$. Right: a Gaussian-noised image generated using a triangular boundary and parameters $\mu_1 = 1$, $\mu_2 = -1$, and $\sigma_1 = \sigma_2 = 1$. Both images have the size 100×100.

The parameter of interest is the boundary of the closed region $\gamma := \partial\Gamma$, which is assumed to be closed and smooth, while (ξ, ρ) are nuisance parameters. We make the following assumptions about the noisy image:

1. The pixel locations X_i are sampled either completely randomly, i.e., $X_i \overset{i.i.d.}{\sim} \mathsf{Unif}(\mathcal{X})$ or jitteredly randomly, i.e., $\mathcal{X}$ is first partitioned into blocks $\mathcal{X}_i$ using an equally-spaced grid and then locations are sampled $X_i \sim \mathsf{Unif}(\mathcal{X}_i)$.
2. The closed region Γ is star-shaped with a known reference point $O \in \Gamma$, i.e., the line segment joining O to any point in Γ is also in Γ.
3. The boundary γ is an $\alpha-$Hölder smooth function, i.e., $\gamma \in \mathbb{C}^\alpha(\mathbb{S})$ where

$$\mathbb{C}^\alpha(\mathbb{S}) := \{f : \mathbb{S} \to \mathbb{R}^+, |f^{(\alpha_0)}(x) - f^{(\alpha_0)}(y)| \leq L_f \|x - y\|^{\alpha - \alpha_0}, \forall x, y \in \mathbb{S}\}$$

, where $\mathbb{S}$ is the unit circle, α_0 is the largest integer strictly smaller than α, L_f is some positive constant, and $\|\cdot\|$ is the Euclidean distance.

Assumptions 2 and 3 imply that the region of interest is star-shaped with a smooth boundary. While these assumptions are crucial to guarantee desirable asymptotic properties of the estimator implemented in **BayesBD** and are reasonable in many applications, it is certainly of great interest to investigate the performance of **BayesBD** when these assumptions are violated. In what follows, we study numerous examples that are not uncommon in practice but violate these assumptions to some extent, to demonstrate the flexibility of **BayesBD** and its capacity to handle practical images that may be much more complicated than the two-region setting with assumptions above. These examples include the triangular boundary in Figure 1 which has a piecewise smooth boundary. and

thus violates Assumption 3, and three real data examples including the image of Lake Menteith 8 and two neuroimaging examples 6 where multiple regions are present and the region of interest has non-smooth or even discontinuous boundary.

Letting Θ be the parameter space of the parametric family f, conditions to separate the inside and outside parameters are needed. Examples of the parameter space Θ^* for (ξ, ρ) include but are not limited to:

4A. One-parameter family such as Bernoulli, Poisson, exponential distributions, and $\Theta^* = \Theta^2 \cap \{(\xi, \rho) : \rho < \xi\}$, or $\Theta^* = \Theta^2 \cap \{(\xi, \rho) : \rho > \xi\}$.

4B. Two-parameter family such as Gaussian distributions, and $\Theta^* = \Theta^2 \cap \{((\mu_1, \sigma_1), (\mu_2, \sigma_2)) : \mu_1 > \mu_2, \sigma_1 = \sigma_2\}$, or $\Theta^* = \Theta^2 \cap \{((\mu_1, \sigma_1), (\mu_2, \sigma_2)) : \mu_1 > \mu_2, \sigma_1 > \sigma_2\}$, or $\Theta^* = \Theta^2 \cap \{((\mu_1, \sigma_1), (\mu_2, \sigma_2)) : \mu_1 = \mu_2, \sigma_1 > \sigma_2\}$.

In practice, the order restriction in 4A or 4B is often naturally obtained depending on the concrete problem. For instance, in brain oncology, a tumor often has higher intensity values than its surroundings in a positron emission tomography scan, while for astronomical applications objects of interest emit light and will be brighter. A more general condition for any parametric family can be referred to Condition (C) in Li and Ghosal (2015).

It is worth noticing that although model 1 follows a two-region framework, the method in Li and Ghosal (2015) and our developed **BayesBD** have the flexibility to handle data with multiple regions by running the two-region method iteratively, which is demonstrated in the neuroimaging application below.

A nonparametric Bayesian model for image boundaries

Let $Y = \{Y_i\}_{i=1}^n$ and $X = \{X_i\}_{i=1}^n$, then the likelihood of the image data described in (1) is

$$L(Y|X, \theta) = \prod_{i \in I_1} f(Y_i; \xi) \prod_{i \in I_2} f(Y_i; \rho),$$

where $I_1 = \{i : X_i \in \Gamma\}$, $I_2 = \{i : X_i \in \Gamma^c\}$, and θ denotes the full parameter (ξ, ρ, γ).

We view γ as a curve mapping $[0, 2\pi] \to \mathbb{R}^+$ and model it using a randomly rescaled Gaussian process prior on the circle $\mathbb{S}$: $\gamma(\omega) \sim \mathsf{GP}(\mu(\omega), G_a(\cdot, \cdot)/\tau)$ where the covariance kernel

$$\begin{aligned} G_a(t_1, t_2) &= \exp(-a^2\{(\cos 2\pi t_1 - \cos 2\pi t_2)^2 + (\sin 2\pi t_1 - \sin 2\pi t_2)^2\}) \\ &= \exp\{-4a^2 \sin^2(\pi t_1 - \pi t_2)\} \end{aligned}$$

is the so-called *squared exponential periodic kernel* obtained by mapping the squared exponential kernel on unit interval $[0, 1]$ to the circle through $Q : [0, 1] \to \mathbb{S}$, $\omega \to (\cos 2\pi\omega, \sin 2\pi\omega)$ as in MacKay (1998). The parameters a and τ control the smoothness and scale of the kernel, respectively. As shown in Li and Ghosal (2015), the covariance kernel has the following closed form decomposition: $G_a(t, t') = \sum_{k=1}^{\infty} v_k(a)\psi_k(t)\psi_k(t')$ where

$$v_1(a) = e^{-2a^2} I_0(2a^2); \quad v_{2j}(a) = v_{2j+1}(a) = e^{-2a^2} I_j(2a^2),\ j \geq 1;$$

and $I_n(x)$ denotes the modified Bessel function of the first kind of order n and $\psi_j(t)$ is the jth Fourier basis function in $\{1, \cos 2\pi t, \sin 2\pi t, ...\}$. The above expansion allows us to write the boundary as a sum of basis functions:

$$\gamma(\omega) = \mu(\omega) + \sum_{k=1}^{\infty} z_k \psi_k(\omega), \tag{2}$$

where $z_k \sim \mathsf{N}(0, v_k(a)/\tau)$. In practice, we truncate this basis function expansion using the first L functions, i.e., $\gamma(\omega) = \mu(\omega) + \sum_{k=1}^{L} z_k \psi_k(\omega)$. In the **BayesBD** package, we use $L = 2J + 1$ with the default $J = 10$, which seems adequate for accurate approximation of $\gamma(w)$ as shown in Li and Ghosal (2015), but users may specify a different value depending on the application.

We use a Gamma prior distribution Gamma(α_a, β_a) for the rescaling factor a. This random rescaling scheme is critical to obtain rate adaptive estimates without assuming the smoothness level α in Assumption 3 is known; see, for example, van der Vaart and van Zanten (2009) and Li and Ghosal (2015). The default values of hyperparameters are $\alpha_a = 2$ and $\beta_a = 1$.

We use a constant function as the prior mean function $\mu(\cdot)$, with value determined by user input or by an initial maximum likelihood estimation. The other hyperparameter and parameters follow standard conjugate priors. Specifically, we use a Gamma distribution Gamma$(\alpha_\tau, \beta_\tau)$ prior for τ with default values $\alpha_\tau = 500$ and $\beta_\tau = 1$. Priors for the nuisance parameters ξ and ρ depend on the parametric family f, which are:

- For binary images: the parameters are the probabilities $(\pi_1, \pi_2) \sim \text{OIB}(\alpha_1, \beta_1, \alpha_1, \beta_1)$, where OIB stands for ordered independent Beta distributions.
- For Gaussian noise: the parameters are the mean and standard deviation $(\mu_1, \sigma_1, \mu_2, \sigma_2)$ with prior distributions $(\mu_1, \mu_2) \sim \text{OIN}(\mu_0, \sigma_0^2, \mu_0, \sigma_0^2)$ and $(\sigma_1^{-2}, \sigma_2^{-2}) \sim \text{OIG}(\alpha_2, \beta_2, \alpha_2, \beta_2)$, where OIN and OIG are ordered independent normal and Gamma distributions, respectively.

The orders in OIB, OIN and OIG are provided by users if such information is available; otherwise, the ordered independent distributions revert to independent distributions. Our default specifications are chosen to make the corresponding prior distributions spread out. For example, in the **BayesBD** package, the default values are $\alpha_1 = \beta_1 = 0$ for binary images, and $\mu_0 = \bar{Y}, \sigma_0 = 10^3$ and $\alpha_2 = \beta_2 = 10^{-2}$ for Gaussian noise, where $\bar{Y}$ is the same mean of all intensities. Under Assumptions 1–4, Li and Ghosal (2015) proved that the nonparametric Bayes approach is (nearly) rate-optimal in the minimax sense, adaptive to unknown smoothness level α.

Posterior sampling and estimation of the boundary

Let $z = \{z_i\}_{i=1}^L$ and $\Sigma_a = \text{diag}(v_1(a), \ldots, v_L(a))$. We use Metropolis-Hastings (MH) with the Gibbs sampler (Geman and Geman, 1984) to sample the joint distribution of the parameters (z, ξ, ρ, τ, a), where the MH step is for the vector parameter z. We also allow a slice sampling with the Gibbs sampler where slice sampling is used for z as in Li and Ghosal (2015). We give the detailed sampling algorithms for binary image in Algorithm 1 and Gaussian-noised images in Algorithm 2. Comparisons between MH and slice sampling, along with other numerical performances are referred to in the section on **Performance tests**.

Initialize the parameters: $z = 0$, $\tau = 500$, $a = 1$, and $\mu(\cdot)$ is taken to be constant, i.e. a circle. Then, initialize $(\xi, \rho) = (\pi_1, \pi_2)$ by the maximum likelihood estimates given $\mu(\cdot)$.

1. At iteration $t+1$, sample $z^{(t+1)} | (\pi_1^{(t)}, \pi_2^{(t)}, \tau^{(t)}, a^{(t)}, Y, X)$ one entry at a time, using either MH sampling and slice sampling, using the following logarithm of the conditional posterior density

$$N_1 \log \frac{\pi_1^{(t)}(1-\pi_2^{(t)})}{\pi_2^{(t)}(1-\pi_1^{(t)})} + n_1 \log \frac{1-\pi_1^{(t)}}{1-\pi_2^{(t)}} - \frac{\tau}{2}(z^{(t)})^\top \Sigma_{a^{(t)}}^{-1} z^{(t)},$$

   where $n_1 = \sum_{i=1}^n 1(r_i < \gamma_i^{(t)})$ and $N_1 = \sum_{i=1}^n 1(r_i < \gamma_i^{(t)}) Y_i$; here (r_i, ω_i) are the polar coordinates of pixel location X_i and $\gamma_i^{(t)} = \gamma^{(t)}(\omega_i)$ is the radius of the image boundary at iteration t and the ith pixel.

2. Sample $\tau^{(t+1)} | z^{(t+1)}, a^{(t)} \sim \text{Gamma}(\alpha^\star, \beta^\star)$ where $\alpha^\star = \alpha_\tau + L/2$ and $\beta^\star = \beta_\tau + (z^{(t+1)})^\top \Sigma_{a^{(t)}}^{-1} z^{(t+1)}/2$.

3. Sample $(\pi_1, \pi_2) | (z, Y) \sim \text{OIB}(\alpha_1 + N_1, \beta_1 + n_1 - N_1, \alpha_1 + N_2, \beta_1 + n_2 - N_2)$, where $n_2 = \sum_{i=1}^n 1(r_i \geq \gamma_i^{(t)})$ and $N_2 = \sum_{i=1}^n 1(r_i \geq \gamma_i^{(t)}) Y_i$.

4. Sample $a^{(t+1)} | (z^{(t+1)}, \tau^{(t+1)})$ by slice sampling using the logarithm of the conditional posterior density

$$-\sum_{k=1}^L \frac{\log v_k(a^{(t)})}{2} - \sum_{k=1}^L \frac{\tau^{(t+1)} z_k^2}{2 v_k(a^{(t)})} + (\alpha_a - 1) \log a^{(t)} - \beta_a.$$

Algorithm 1: – Binary images.

Let $\{\gamma_t(\omega)\}_{t=1}^T$ be the posterior samples after burn-in where T is the number of posterior samples. We use the posterior mean as the estimate and construct a variable-width uniform credible band. Specifically, let $(\widehat{\gamma}(\omega), \widehat{s}(\omega))$ be the posterior mean and standard deviation functions derived from $\{\gamma_t(\omega)\}$. For the tth MCMC run, we calculate the distance $u_t = \|(\gamma_t - \widehat{\gamma})/s\|_\infty = \sup_\omega \{|\gamma_t(\omega) - \widehat{\gamma}(\omega)|/\widehat{s}(\omega)\}$ and obtain the 95th percentile of all the u_t's, denoted as L_0. Then a 95% uniform credible band is given by $[\widehat{\gamma}(\omega) - L_0 \widehat{s}(\omega), \widehat{\gamma}(\omega) + L_0 \widehat{s}(\omega)]$.

Initialize the parameters: $z = 0$, $\tau = 500$, $a = 1$, and $\mu(\cdot)$ is taken to be constant, i.e. a circle. Then, initialize $(\xi, \rho) = (\pi_1, \pi_2)$ by the maximum likelihood estimates given $\mu(\cdot)$.

1. At iteration $t+1$, sample $z^{(t+1)}|(\mu_1^{(t)}, \sigma_1^{(t)}, \mu_2^{(t)}, \sigma_2^{(t)}, \tau^{(t)}, a^{(t)}, Y, X)$ one entry at a time, using either slice sampling or MH sampling, using the following logarithm of the conditional posterior density

$$-n_1(\log\sigma_1^{(t)} - \log\sigma_2^{(t)}) - \sum_{i\in I_1}\frac{(Y_i-\mu_1^{(t)})^2}{2(\sigma_1^2)^{(t)}} - \sum_{i\in I_2}\frac{(Y_i-\mu_2^{(t)})^2}{2(\sigma_2^2)^{(t)}} - \frac{\tau(z^{(t)})^\top\Sigma_{a^{(t)}}^{-1}z^{(t)}}{2}.$$

2. Sample $\tau^{(t+1)}|z^{(t+1)}, a^{(t)}$ as in Algorithm 1.

3. Sample $(\mu_1, \sigma_1, \mu_2, \sigma_2)|(z, Y)$ conjugately.

   - Sample $(\sigma_1^{-2})^{(t+1)}$ from a Gamma distribution with parameters

$$\alpha = \alpha_2 + \frac{n_1}{2}, \qquad \beta = \beta_2 + \sum_{i\in I_1}\frac{(Y_i-\bar{Y}^{(1)})^2}{2} + \frac{\sigma_0^{-2}n_1}{n_1+\sigma_0^{-2}}\frac{(\bar{Y}^{(1)}-\mu_0)^2}{2},$$

     where $\bar{Y}^{(1)}$ is the sample mean of intensities in I_1.

   - sample $\mu_1^{(t+1)}$ from a normal distribution with mean and standard deviation

$$\frac{\sigma_0^{-2}\mu_0}{n_1+\sigma_0^{-2}} + \frac{n_1\bar{y}_1}{n_1+\sigma_0^{-2}}, \qquad (n_1+\sigma_0^{-2})^{-1/2}.$$

   - Sample $(\sigma_2^{-2})^{(t+1)}$ and $\mu_2^{(t+1)}$ analogously.
   - If `ordering` information is available, sort $(\mu_1^{(t+1)}, \mu_2^{(t+1)})$ and $(\sigma_1^{(t+1)}, \sigma_2^{(t+1)})$ accordingly.

4. Sample $a^{(t+1)}|(z^{(t+1)}, \tau^{(t+1)})$ as in Algorithm 1.

Algorithm 2: – Gaussian-noised images.

Analysis of image boundaries using BayesBD from the command line

There are three steps to our Bayesian image boundary analysis: load the image data into R in the appropriate format, use the functions provided to sample from the joint posterior of the full parameter $\theta = (\xi, \rho, \gamma)$, and summarize the posterior samples both numerically and graphically.

Generating image data

Two functions are included in **BayesBD** to facilitate data simulation for numerical experiments: `par2obs` for binary images and `parnormobs` for Gaussian-noised images. Table 1 describes the function arguments to `par2obs`, which returns sampled intensities and pixel locations in both Euclidean and polar coordinates. The function `parnormobs` is similar, with the replacement of arguments `pi.in` and `pi.out` by `mu.in`, `mu.out`, `sd.in`, and `sd.out` corresponding to parameters μ_1, μ_2, σ_1, and σ_2, respectively.

As a demonstration, the following code generates a 100×100 binary image of an ellipse using a jitteredly-random design with a reference point of (0.5, 0.5):

```
> gamma.fun = ellipse(a = 0.35, b = 0.25)
> bin.obs = par2obs(m = 100, pi.in = 0.6, pi.out = 0.4, design = 'J',
+ gamma.fun, center = c(0.5, 0.5))
```

Similarly, the following code generates a 100×100 Gaussian-noised image with a triangle boundary using a random uniform design with a reference point of (0.5, 0.5):

```
> gamma.fun = triangle2(0.5)
> norm.obs = parnormobs(m = 100, mu.in = 1, mu.out = -1, sd.in = 1,
+ sd.out = 1, design = 'U', gamma.fun, center = c(0.5, 0.5))
```

The output of either `par2obs` or `parnormobs` is a list containing the intensities Y in a vector named `intensity`, the vectors `theta.obs` and `r.obs` containing the polar coordinates of the pixel locations X, and the reference point contained in a vector named `center`. Image data to be analyzed using **BayesBD** can be in a list of this form, or may be a `.png` or `.jpg` image file.

Argument	Description
`m`	Generate $m \times m$ observations over the unit square.
`pi.in`	The success probability, $P(Y_i = 1)$, if $X_i \in I_1$.
`pi.out`	The success probability, $P(Y_i = 1)$, if $X_i \in I_2$.
`design`	Determines how locations X_i are determined: 'D' for deterministic (equally-spaced grid) design, 'U' for completely uniformly random, or 'J' for jitteredly random design.
`center`	Two-dimensional vector of Euclidean coordinates (x,y) of the reference point in I_1.
`gamma.fun`	A function to generate boundaries, e.g. ellipse or triangle2.

Table 1: Arguments of the `par2obs` function.

Analysis and visualization

There are two functions to draw posterior samples following Algorithm 1 and 2 based on images either simulated or provided by users: `fitBinImage` for binary images, and `fitContImage` for Gaussian-noised images. These sampling functions take the same arguments, with the exception of the `ordering` input which is duplicated in `fitContImage` to allow ordering of the two parameters, i.e., the mean and standard deviation. The inputs for `fitBinImage` are summarized in Table 2. We have included a function `rectToPolar` to facilitate formatting the image data for `fitBinImage` and `fitContImage` by converting the rectangular coordinates of the pixels to polar coordinates. The initial boundary is a circle with radius `inimean` and center `center`. The radius `inimean` may be specified by the user or left blank, in which case it will be estimated using maximum likelihood.

`image`	The noisy observation, either a list with elements: `intensity`, a vector of intensities; `theta.obs` a vector of pixel radian measure from `center`; `r.obs` a vector of pixel radius measure from `center`; or a string giving the path to a `.png` or `.jpg` file.
`gamma.fun`	the true boundary, if known, used for plotting.
`center`	the reference point in Euclidean coordinates.
`inimean`	A constant specifying the initial boundary μ. Defaults to NULL, in which case μ is estimated automatically using maximum likelihood.
`nrun`	The number of MCMC runs to keep for estimation.
`nburn`	The number of initial MCMC runs to discard.
`J`	The number of eigenfunctions to use in estimation is $2J + 1$.
`ordering`	Indicates which Bernoulli distribution has higher success probability: `"I"`, the Bernoulli distribution inside the boundary; `"O"`, ther Bernoulli distribution outside the boundary; `"N"`, no ordering information is available.
`mask`	Logical vector (same length as `obs$intensity`) to indicate region of interest. Should this data point be included in the analysis? Defaults to NULL and uses all data points.
`slice`	Should slice sampling be used to sample Fourier basis function coefficients?
`outputAll`	Should all posterior samples be returned?

Table 2: Arguments of the `fitBinImage` function.

If argument `outputAll` is `FALSE`, the functions `fitBinImage` and `fitContImage` produce output vectors `theta`, `estimate`, `upper`, and `lower` giving a grid of 200 values on $[0, 2\pi]$ on which the boundary

γ is estimated, along with 95% uniform credible bands. If argument `outputAll` is `TRUE`, the functions also return posterior samples of (ξ, ρ) and Fourier basis function coefficients z.

Following the examples of data simulation for binary and Gaussian-noised images in the previous section, we can obtain posterior samples via

```
> bin.samples= fitBinImage(bin.obs, c(.5, .5), NULL, 4000, 1000, 10, "I", NULL, TRUE,
+ FALSE)
```

for a binary image and

```
> norm.samples = fitContImage(norm.obs, c(.5, .5), NULL, 4000, 1000, 10, "I", "N", NULL,
TRUE, FALSE)
```

for a Gaussian-noised image. For each sampling function, we have set the center of the image as $(0.5, 0.5)$, instructed the sampler to use a mean function of $\mu(\cdot) = 0.4$, keep 4000 samples, burn 1000 samples, use $L = 2 \times 10 + 1 = 21$ basis functions to model γ, use slice sampling for the basis function coefficients, and return only the plotting results.

Using the function `plotBD` we can easily construct plots of our model results. There are two arguments to `plotBD`: `fitted.image` is a list of results from `fitBinImage` or `fitContImage`, and `plot.type` which indicates whether to plot the image data only, the posterior mean and credible bands, or the posterior mean overlaid on the image data. Using the posterior samples we obtained from the Gaussian-noised image above, we can plot our results with the following code

```
> par(mfrow = c(1, 2))
> plotBD(norm.samples, 1)
> plotBD(norm.samples, 2)
```

to produce Figure 2.

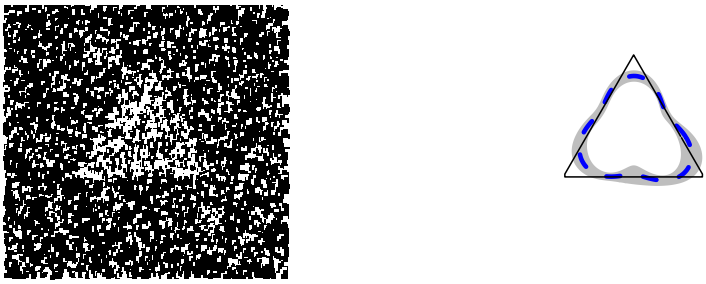


Figure 2: Output from `fitBinImage` as plotted using `plotBD`. The plot on the left is the simulated binary observation, and the plot on the right is the estimated boundary and 95% uniform credible bands.

BayesBD shiny app

In order to reach a broad audience of users, including those who may not be familiar with R, we have created a **shiny** app version of **BayesBD** to implement the boundary detection method. The app has the full functionality to reproduce the simulations in Section 24.5 and conduct real data applications using images uploaded by users. The app is accessible both from the package by running the code `BayesBDshiny()` in an R session or externally by visiting `https://syring.shinyapps.io/shiny/`. With the app users can analyze real data or produce a variety of simulations.

Once the app is open, users are presented with an array of inputs to set as illustrated in Figure 3. In order to analyze real data, the user should select "user continuous image" or "user binary image" from the second drop-down menu. Next, the user inputs the system path to the `.png` or `.jpg` file. A plot of the image will appear and the user will be prompted to identify the image center with a mouse click. The user has some control over the posterior sample size, but we recommend to first limit the number of available samples in order to display results quickly. Finally, the user may enter ordering information for the mean and variance of pixel intensities at the bottom of the display.

For simulations, we first select either an elliptical or triangular boundary, or upload an R script with a custom boundary function. Next, the user instructs the app to either simulate a binary or Gaussian-noised image, or to use binary or Gaussian data the user has uploaded. In addition to the

boundary function, the user specifies the reference point. Sample sizes for simulations are kept at 100×100 pixels. Finally, the last several inputs allow the user to customize the intensity parameters (ξ, ρ) for binary and Gaussian simulations. Once the user has selected all settings, clicking the "Update" button at the bottom of the window will run the posterior sampling algorithm, which should take less than a minute on a typical computer for image data of 100×100 pixels. The "Download" button provides the user with a file indicating which pixels were contained inside the estimated boundary as determined by the outer edge of the 95% uniform credible bands.

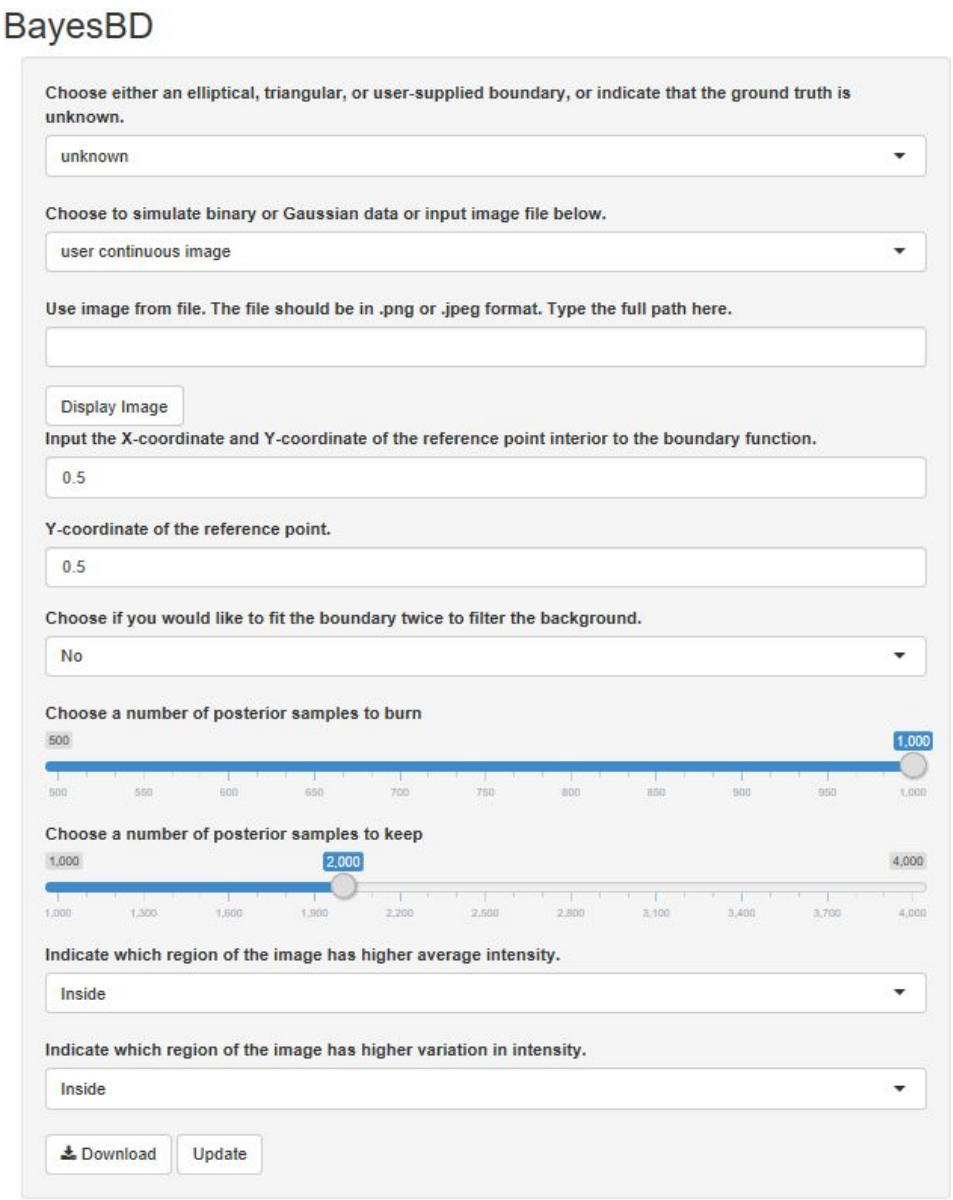


Figure 3: Screenshot of shiny app implementing **BayesBD**.

Performance tests

Comparison of sampling methods

The main aim of this section is to highlight the speed improvements we have made in the latest version of **BayesBD**. Our flexibility in choosing between slice and Metropolis-Hastings (MH) sampling algorithms gives users the potential to unlock efficiency gains. We highlight these gains below for both binary and Gaussian-noised simulations, and note that the faster MH method suffers little in accuracy.

In our performance tests, we consider the following examples, which correspond to examples B2, B3, and G1 from Li and Ghosal (2015).

S1. Image is an ellipse centered at $(0.1, 0.1)$ and rotated 60° counterclockwise. Intensities are binary with $\pi_1 = 0.5$ and $\pi_2 = 0.2$.

S2. Image is an equilateral triangle centered at $(0, 0)$ with height 0.5. Intensities are binary with $\pi_1 = 0.5$ and $\pi_2 = 0.2$.

S3. Image is an ellipse centered at $(0.1, 0.1)$ and rotated 60° counterclockwise. Intensities are Gaussian with $\mu_1 = 4$, $\sigma_1 = 1.5$, $\mu_2 = 1$, and $\sigma_2 = 1$.

We simulated each case 100 times using $n = 100 \times 100$ observations per simulation. In each run, we sampled 4000 times from the posterior after a 1000 sample burn in. The results of our performance tests comparing slice with MH sampling are summarized in Table 3.

If Metropolis-Hastings sampling is used instead of slice sampling, we observe a speed up by about a factor of two for binary images, seven for the Gaussian-noised image. Slice sampling is guaranteed to produce unique posterior samples, and may give better results than Metropolis-Hastings samplers, especially when Metropolis-Hastings mixes poorly and produces many repeated samples. However, slice sampling may involve a very large number of proposed samples for each accepted sample, requiring many likelihood evaluations. On the other hand, each Metropolis Hastings sample requires only two evaluations of the likelihood.

To measure the accuracy of **BayesBD** we use three metrics: the Lebesgue error, which is simply the area of the symmetric difference between the posterior mean boundary and the true boundary; the Dice Similarity Coefficient(DSC), see Feng and Tierney (2015); and Hausdorff distance, see Thompson (2014) and Thompson (2017). We have included the utility functions `lebesgueError`, `dsmError`, and `hausdorffError`, which take as input the output of either `fitBinImage` or `fitContImage` and output the corresponding error. In the binary image examples considered, the different sampling algorithms did not affect the accuracy of the posterior mean boundary estimates when measured by Lebesgue error, i.e., the area of the symmetric difference between the estimated boundary and the true boundary. For the Gaussian-noised image, the slice sampling method produced Lebesgue errors approximately an order of magnitude smaller than when using Metropolis-Hastings sampling, but the overall size of the errors was still small in both cases and practically indistinguishable when plotting results. With our built-in functions, it is easy to reproduce Example S2 in Table 3 with the following code:

```
> gamma.fun = triangle2(0.5)
> for(i in 1:100){
+         norm.obs = par2obs(m = 100, pi.in = 0.5, pi.out = 0.2, design = 'J',
+                                          center = c(0.5, 0.5), gamma.fun)
+         norm.samp.MH = fitBinImage(norm.obs, gamma.fun, NULL, NULL, 4000, 1000,
+                                          10, "I", rep(1, 10000), FALSE, FALSE)
+         norm.samp.slice = fitBinImage(norm.obs, gamma.fun, NULL, NULL, 4000,
+                                           1000, 10, "I", rep(1, 10000), TRUE, FALSE)
+         print(c(dsmError(norm.samp.MH), hausdorffError(norm.samp.MH),
+                 lebesgueError(norm.samp.MH), dsmError(norm.samp.slice),
+                 hausdorffError(norm.samp.slice), lebesgueError(norm.samp.slice)))
+ }
```

Example	Sampling Method	Runtime (s)	Lebesgue Error	DSC	Hausdorff
S1	MH	58	0.01(0.01)	0.02(0.00)	0.02(0.00)
	Slice	100	0.01(0.00)	0.02(0.00)	0.02(0.00)
S2	MH	45	0.02(0.00)	0.09(0.02)	0.09(0.01)
	Slice	82	0.02(0.00)	0.09(0.01)	0.09(0.01)
S3	MH	66	0.01(0.01)	0.01(0.01)	0.01(0.01)
	Slice	488	0.00(0.00)	0.01(0.01)	0.01(0.01)

Table 3: Average Runtimes (in seconds) and Lebesgue errors (with standard deviations) of posterior mean boundary estimates using C++.

Comparison of coding platforms

Our use of C++ for posterior sampling has led to very significant efficiency gains over using R alone. Implementation of C++ with R is streamlined using **Rcpp** and **RcppArmadillo**.

The first implementation of **BayesBD** (version 0.1 in Li (2015)) was entirely written in R. The results in Table 4 labeled R code reflect this first version of the package, while those labeled C++ code

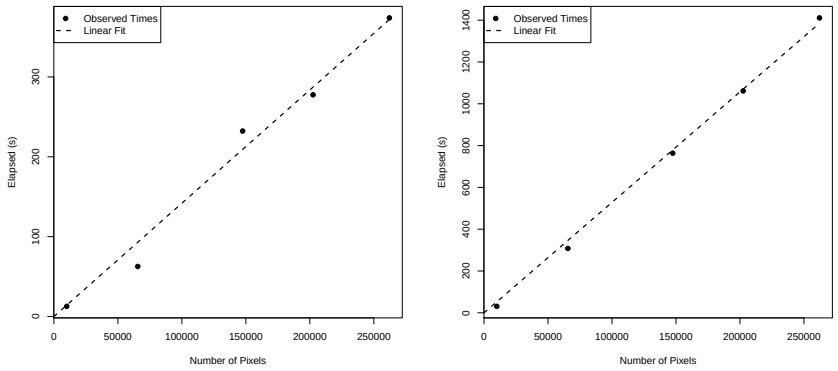


Figure 4: Left: Runtimes of `fitBinImage`. Right: Runtimes of `fitContImage`

are from the new version. The main takeaway is that the new code developed using C++ is at least three times faster in the binary image examples and over six times faster for the Gaussian-noised image example, both measured while using slice sampling.

Example	Coding Method	Runtime (s)
S1	C++	100
	R	375
S2	C++	82
	R	246
S3	C++	488
	R	3327

Table 4: Average run times (in seconds) using slice sampling.

Scalability of BayesBD

The Gaussian process is notorious for scaling poorly as n increases because it is usually necessary to invert of a large covariance matrix. By utilizing the analytical decomposition of a GP kernel in (2), we eliminates the step of inverting a n by n covariance matrix and the **BayesBD** package appears to achieve a linear complexity. We investigate the scalability of **BayesBD** by plotting the system time against sample size for `fitBinImage` and `fitContImage` in Figure 4 using a triangular boundary curve and 5000 MCMC iterations. Both algorithms appear to scale approximately linearly in number of pixels, which makes sense as the costliest computations in Steps 1 and 3 in Algorithms 1 and 2 only involve sums over the n pixels.

Comparison with existing packages

Although no packages besides **BayesBD** provide boundary estimation, there are several existing packages that can provide image segmentation or filtering. Below we make some qualitative comparisons between **BayesBD** and **mritc**, **bayesImageS**, and **bayess**; see Feng and Tierney (2015), Moores et al. (2017), and Robert and Marin (2015) in a later section. Figure 5 compares these packages using two simulated images with Bernoulli and Gaussian noise, respectively. **BayesBD** gives very reasonable estimates for the true boundaries; **mritc** package fails to deliver a recognizable smoothed image in either example; and **bayesImageS** was able to produce a very clear segmentation for the Gaussian-noised ellipse example, but not for the triangle image with Bernoulli noise.

Real data application

Medical imaging

Chen et al. (2006) studied the performance of two different tracers used in conjunction with positron emission tomography (PET) scans in identifying brain tumors. Figure 6, reproduced from (Chen

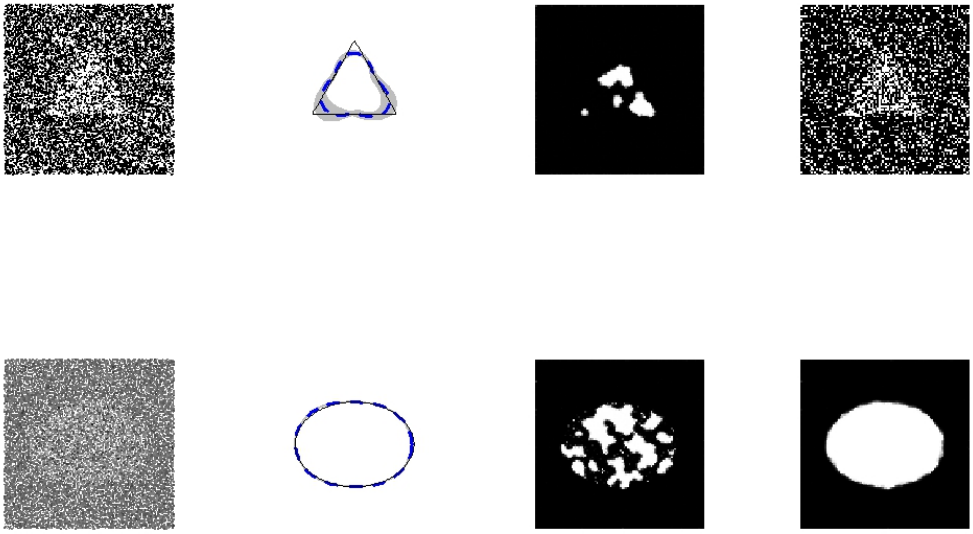


Figure 5: Left to right: the image, the **BayesBD** boundary estimate, the **mritc** filtered image, and the **bayesImageS** filtered image.

et al., 2006), gives an example of the image data used in diagnosing tumors, and demonstrates their conclusion that the F-FDOPA tracer provides a more informative PET scan than the F-FDA tracer. We use the **BayesBD** package to analyze the F-FDOPA PET scan images in Figure 6. The tumor imaging data along with sample code for reproducing the following analysis can be found in the documentation to the **BayesBD** package.

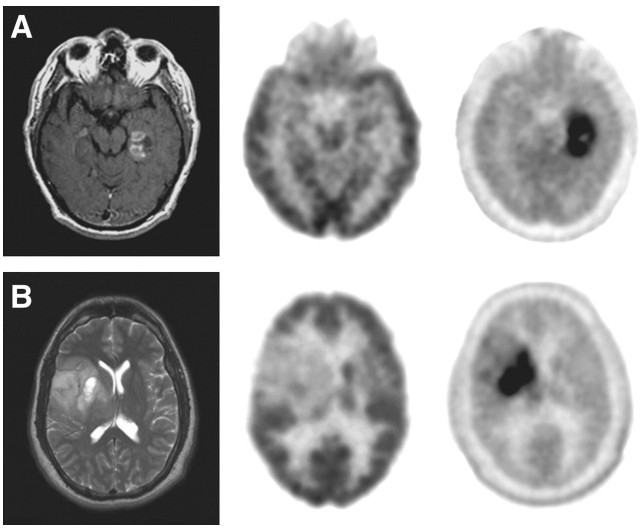
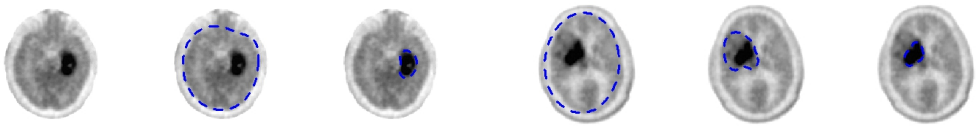


Figure 6: MRI (left), F-FDG PET (middle), and F-FDOPA PET (right) of glioblastoma (A) and grade II oligodendroglioma (B). Image taken from Chen et al. (2006).

We convert the two F-FDOPA PET images in Figure 6 into 111×111-pixel images and normalize the intensities to the interval [0,10]. The pixel coordinates are a grid on $[0,1] \times [0,1]$ and we choose reference points $(0.7, 0.5)$ and $(0.4, 0.55)$ for each image, roughly corresponding to the center of the darkest part of each image. We use the default mean function, choose $J = 10$ for 21 basis functions, and sample 4000 times after a 1000 burn-in using MH sampling.

Figure 7 displays posterior mean boundary estimates for the F-FDOPA images in Figure 6. From the analysis on glioblastoma (A) in the first two plots, it seems that that we accurately capture the regions



A. Glioblastoma B. Grade II oligodendroglioma

Figure 7: F-FDOPA PET images from Chen et al. (2006) (left) fit twice, and (right) fit three times to filter the background and find features at increasing granularity.

of interest in the F-FDOPA PET images. Furthermore, it is expected that the Gaussian assumption on the real data may fail, and this shows that the method implemented in **BayesBD** is robust to model misspecifications, thus practically useful.

Tumor heterogeneity, which is not unusual in many applications, may make the boundary detection problem more challenging (Heppner, 1984; Ananda and Thomas, 2012). The **BayesBD** package allows us to address tumor heterogeneity by a repeated implementation. We first apply `fitContImage` to the entire image, which includes a white background not of interest, and produce the estimated boundary. This step succeeds in separating the brain scan from the white background. A second run is performed on the subset of the image inside the outer 95% uniform credible band, producing a nested boundary. In general, this technique can be used in a multiple region setting where the data displays more heterogeneity than the simple "image and background" setup in (1).

Satellite imaging

We compared the performance of **BayesBD** with the R packages **mritc** and **bayess** using an image of Lake Menteith available in the **bayess** package. **BayesBD** gives a very reasonable estimate for the boundary of the lake even though it is not smooth. The **mritc** package again does not provide useful output in this example, but **bayess** produces a nicely-segmented image; see Figure 8.

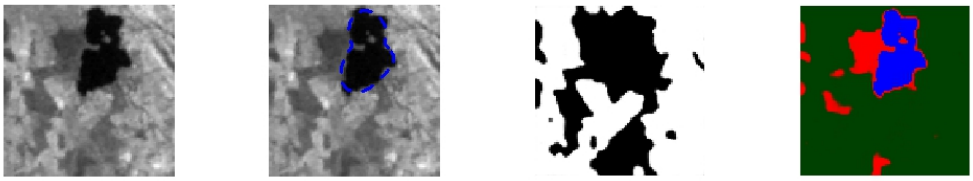


Figure 8: Left to right: the image, the **BayesBD** boundary estimate, the **mritc** filtered image, and the **bayess** segmented image.

Summary

BayesBD is a new computational platform for image boundary analysis with many advantages over existing software. The underlying methods in functions `fitBinImage` and `fitContImage` are based on theoretical results ensuring their dependability over a range of problems. Our use of **Rcpp** and **RcppArmadillo** help make **BayesBD** much faster than base R code and further speed can be gained by our flexible sampling algorithms. Finally, our integration with **shiny** provides users with an easy way to utilize our package without having to write R code.

For the latest updates to **BayesBD** and requests, readers are recommended to check out the package page at CRAN or refer to the Github page at `https://github.com/nasyring/GSOC-BayesBD`.

Acknowledgments

This work was partially supported by the Google Summer of Code program. We thank the Associate Editor and one anonymous referee for comprehensive and constructive comments that helped to improve the paper.

Bibliography

R. S. Ananda and T. Thomas. Automatic segmentation framework for primary tumors from brain MRIs using morphological filtering techniques. In *Biomedical Engineering and Informatics (BMEI), 2012 5th International Conference on*, pages 238–242. IEEE, 2012. [p160]

M. Basu. Gaussian-based edge-detection methods—a survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, 32(3):252–260, 2002. [p149]

S. Bhardwaj and A. Mittal. A survey on various edge detector techniques. *Procedia Technology*, 4: 220–226, 2012. [p149]

W. Chen, D. H. S. Silverman, D. Sibylle, J. Czernin, N. Kamdar, W. Pope, N. Satyamurthy, C. Schiepers, and T. Cloughesy. F-FDOPA PET imaging of brain tumors: Comparison study with F-FDG PET evaluation of diagnostic accuracy. *Journal of Nuclear Medicine*, 47(6):904–911, 2006. [p158, 159, 160]

L. Cucala and J.-M. Marin. "bayesian inference on a mixture model with spatial dependence. *J. Comput. Graph. Stat.*, 22(3):584–597, 2014. [p149]

D. Eddelbuettel. *Seamless R and C++ Integration with Rcpp*. Springer-Verlag, New York, 2013. ISBN 978-1-4614-6867-7. [p149]

D. Eddelbuettel and C. Sanderson. Rcpparmadillo: Accelerating R with high-performance C++ linear algebra. *Computational Statistics & Data Analysis*, 71:1054–1063, 2014. URL http://dx.doi.org/10.1016/j.csda.2013.02.005. [p149]

D. Feng and L. Tierney. *Mritc: MRI Tissue Classification*, 2015. URL https://CRAN.R-project.org/package=mritc. [p157, 158]

M. C. Fitzpatrick, E. L. Preisser, A. Porter, J. Elkinton, L. A. Waller, B. P. Carlin, and A. M. Ellison. Ecological boundary detection using Bayesian areal wombling. *Ecology*, 91(12):3448–3455, 2010. ISSN 00129658. URL http://www.jstor.org/stable/29779525. [p149]

S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6(6):721–741, 1984. [p152]

U. Grenander and M. I. Miller. *Pattern Theory: From Representation to Inference.* Oxford University Press, New York, 2007. [p149]

W. K. Hastings. Monte carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1):97–109, 1970. [p149]

G. H. Heppner. Tumor heterogeneity. *Cancer research*, 44:2259–2265, 1984. [p160]

M. A. Hurn, O. K. Husby, and H. Rue. Advances in Bayesian image analysis. In P. J. Green, N. L. Hjort, and S. Richardson, editors, *Highly Structured Stochastic Systems*, pages 301–322. Oxford University Press, Oxford, 2003. [p149]

C. Li, C. Xu, C. Gui, and M. D. Fox. Distance regularized level set evolution and its application to image segmentation. *IEEE Trans. Image Processing*, 19(12):3243–3254, 2010. [p149]

M. Li. *BayesBD: Bayesian Boundary Detection in Images*, 2015. R package version 0.1. [p149, 157]

M. Li and S. Ghosal. Bayesian detection of image boundaries, 2015. URL https://arxiv.org/abs/1508.05847. [p149, 150, 151, 152, 156]

H. Lu and B. P. Carlin. Bayesian areal wombling for geographical boundary analysis. *Geographical Analysis*, 37(3):265–285, 2005. URL http://proxying.lib.ncsu.edu/index.php?url=/docview/289033244?accountid=12725. [p149]

D. J. MacKay. Introduction to Gaussian processes. *NATO ASI Series F Computer and Systems Sciences*, 168:133–166, 1998. [p151]

R. Maini and H. Aggarwal. Study and comparison of various image edge detection techniques. *International Journal of Image Processing*, 3(1):1–11, 2009. [p149]

J.-M. Marin and C. P. Robert. *Bayesian Essentials with R*. Springer-Verlag, 2014. [p149]

M. Moores, K. Mengersen, and D. Feng. *Bayesian Methods for Image Segmentation Using a Potts Model*, 2017. URL https://CRAN.R-project.org/package=bayesImageS. [p158]

R. Neal. Slice sampling. *The Annals of Statistics*, 31(3):705–767, 2003. [p149]

C. P. Robert and J.-M. Marin. *Bayesian Essentials with R*, 2015. URL https://CRAN.R-project.org/package=bayess. [p158]

RStudio, Inc. *Easy Web Applications in R.*, 2016. URL: http://www.rstudio.com/shiny/. [p149]

N. Syring and M. Li. *BayesBD: Bayesian Inference for Image Boundaries*, 2017. R package version 1.2. [p149]

R. F. Thompson. RadOnc: An R package for analysis of dose-volume histogram and three- dimensional structural data. *J. Rad. Onc. Info.*, 6(1):98–100, 2014. [p157]

R. F. Thompson. *RadOnc: Analytical Tools for Radiation Oncology*, 2017. URL https://CRAN.R-project.org/package=RadOnc. [p157]

A. W. van der Vaart and J. H. van Zanten. Adaptive Bayesian estimation using a Gaussian random field with inverse gamma bandwidth. *Ann. Statist.*, 37(5B):2655–2675, 2009. ISSN 0090-5364. URL https://doi.org/10.1214/08-aos678. [p151]

L. A. Waller and C. A. Gotway. *Applied Spatial Statistics for Public Health Data*. Wiley Series in Probability and Statistics. John Wiley & Sons, 2004. ISBN 0-471-38771-1. URL https://doi.org/10.1002/0471662682. [p149]

W. Yuan, K. S. Chin, M. Hua, G. Dong, and C. Wang. Shape classification of wear particles by image boundary analysis using machine learning algorithms. *Mechanical Systems and Signal Processing*, 72-73:346–358, 2016. [p149]

D. Ziou and S. Tabbone. Edge detection techniques – an overview. *Pattern Recognition and Image Analysis C/C of Raspoznavaniye Obrazov I Analiz Izobrazhenii*, 8:537–559, 1998. [p149]

Nicholas Syring
Department of Statistics
North Carolina State University, 5109 SAS Hall
2311 Stinson Dr.
Raleigh, NC 27695 USA
nasyring@ncsu.edu

Meng Li
Rice University
Department of Statistics
6100 Main St
Houston, TX 77251-1892 USA
meng@rice.edu

arulesViz: Interactive Visualization of Association Rules with R

by Michael Hahsler

Abstract Association rule mining is a popular data mining method to discover interesting relationships between variables in large databases. An extensive toolbox is available in the R-extension package **arules**. However, mining association rules often results in a vast number of found rules, leaving the analyst with the task to go through a large set of rules to identify interesting ones. Sifting manually through extensive sets of rules is time-consuming and strenuous. Visualization and especially interactive visualization has a long history of making large amounts of data better accessible. The R-extension package **arulesViz** provides most popular visualization techniques for association rules. In this paper, we discuss recently added interactive visualizations to explore association rules and demonstrate how easily they can be used in **arulesViz** via a unified interface. With examples, we help to guide the user in selecting appropriate visualizations and interpreting the results.

Introduction

Many organizations generate a significant amount of transaction data on a daily basis. For example, a department store like "Macy's" stores customer shopping information originating from point-of-sale systems and online shopping on a large scale. Association rule mining (Agrawal et al., 1993; Tan et al., 2006) is one of the major techniques to detect and extract useful information from large-scale transaction data. Rules found in the data are of the form 'if customers purchase in a transaction products A and B then they are more likely also to purchase product C in the same transaction.' This approach can be easily extended to non-retail settings by replacing products with web pages, movies, different answers to a questionnaire, etc. A well-known practical problem with association rule mining is that it tends to create a significant number of potentially interesting rules. Analysts are often overwhelmed by the sheer number of rules and need tools to support exploring large sets of rules efficiently.

Visualization has a long history of making large data sets better accessible and is successfully used to communicate both abstract and concrete ideas in many areas like education, engineering, and science (Prangsmal et al., 2009). According to Chen et al. (2008), the application of visualization falls into two phases. First, the exploration phase where the analysts will use graphics that are mostly incompatible for presentation purposes but make it easy to find interesting and important features of the data. The amount of interaction needed during exploration is very high and includes filtering, zooming, and rearranging data. After key findings are discovered in the data, these results must be presented in a way suitable for presentation for a larger audience. In this second phase, it is important that the analyst can manipulate the presentation to highlight the findings. Many researchers applied visualization techniques like scatter plots, matrix visualizations, graphs, mosaic plots and parallel coordinates plots to help analyze association rules (see Bruzzese and Davino (2008) for a recent overview paper).

This paper introduces the recently added implementations of interactive versions of several popular visualization techniques in the R-package **arulesViz** (Hahsler, 2017) and demonstrates how to use the package's simple unified interface. Choosing an appropriate visualization and interpreting the results needs some experience. To give the user some guidance, this paper discusses three major groups of interactive visualizations including scatter plots, matrix visualization and graph-based visualization. With examples, the paper shows how the results of different visualizations can be interpreted to gain more insight into the found set of association rules.

The rest of the paper is organized as follows. We start with definitions used in association rule mining and a discussion of different visualization methods for association rules. Then, we introduce the unified interface in package **arulesViz**. We demonstrate with small examples how to create and interpret different interactive visualizations. We conclude the paper with a short discussion of how the plots can be used to explore a set of association rules.

Association rules

Mining association rules was fist introduced by Agrawal et al. (1993) and, following the notation used by Agrawal et al. (1993), Hahsler et al. (2005) and Tan et al. (2006), can formally be defined as:

Let $\mathcal{D} = \{t_1, t_2, \ldots, t_m\}$ be a set of transactions called the *database*, and let $I = \{i_1, i_2, \ldots, i_n\}$ be

the set of all *items* considered in the database. Each transaction in $\mathcal{D}$ has a unique transaction ID and contains a subset of the items in I. A *rule* is defined as an expression $X \Rightarrow Y$ where $X, Y \subseteq I$ and $X \cap Y = \emptyset$. The sets of items (for short *itemsets*) X and Y are called *antecedent* (left-hand-side or LHS) and *consequent* (right-hand-side or RHS) of the rule. Often rules are restricted to only a single item in the consequent.

Association rules are rules which surpass a user-specified minimum support and minimum confidence threshold. The *support*, $\text{supp}(X)$, of an itemset X is a measure of importance defined as the proportion of transactions in the data set which contain the itemset. The *confidence* of a rule is defined as $\text{conf}(X \Rightarrow Y) = \text{supp}(X \cup Y)/\text{supp}(X)$, measuring how likely it is to see Y in a transaction containing X. An association rule $X \Rightarrow Y$ needs to satisfy

$$\text{supp}(X \cup Y) \geq \sigma \quad \text{and} \quad \text{conf}(X \Rightarrow Y) \geq \delta,$$

where σ and δ are the minimum support and minimum confidence thresholds, respectively.

Another popular measure for association rules used throughout this paper is *lift* (Brin et al., 1997). The lift of a rule is defined as

$$\text{lift}(X \Rightarrow Y) = \text{supp}(X \cup Y)/\left(\text{supp}(X)\,\text{supp}(Y)\right)$$

and can be interpreted as the deviation of the support of the whole rule from the support expected under independence given the supports of both sides of the rule. Greater lift values ($\gg 1$) indicate stronger associations. Measures like support, confidence, and lift are called interest measures because they help with focusing on potentially more interesting rules. For a more detailed treatment of association rules and interest measures, we refer the reader to the introduction paper (Hahsler et al., 2005) for package **arules** (Hahsler et al., 2017) and the literature referred to there.

Association rules are typically generated in a two-step process. First, minimum support is used to produce the set of all *frequent itemsets* for the data set. Frequent itemsets are itemsets which satisfy the minimum support constraint. Then, in a second step, each frequent itemset is used to generate all possible candidate rules from it, and all rules which do not satisfy the minimum confidence constraint are removed. Analyzing this process, we can see that in the worst case we will generate $2^n - n - 1$ frequent itemsets with more than two items from a database with n distinct items. Since each frequent itemset will in the worst case generate at least two rules, we will end up with a set of rules in the order of $O(2^n)$. Typically, increasing minimum support is used to keep the number of association rules found at a manageable size. However, this also removes potentially interesting rules with less support. Therefore, the need to deal with large sets of association rules is unavoidable when applying association rule mining in a real setting. Here we discuss interactive visualization as a potential means to analyze large sets of association rules.

Visualizing association rules

Many researchers applied existing visualization techniques to sets of association rules. Several popular techniques are discussed in the overview by Bruzzese and Davino (2008) and implemented in **arulesViz** (Hahsler, 2017). Here we focus on interactive visualizations falling into one of the three most important groups scatter plots, matrix visualization, and graph-based visualization. The main components of the three groups of visualization are shown in Figure 1. Scatter plots focus on interest measures, and rules with similar values for these measures are placed close to each other. Matrix visualizations focus on visualizing rules that have the same antecedent or consequent by placing them in the same column or row, repectively. The graph-based visualization shows how rules share individual items. The properties of interactive visualizations in these groups implemented in **arulesViz** are summarized in Table 1. The table includes information on the maximum size of the rule set that can be effectively visualized, the number of measures of interestingness that are visualized, the primary focus of the visualization, and the interactive features that are currently available. Additional static and interactive visualizations are available in package **arulesViz** and we refer the reader to the package's manual for these.

Visualization starts with a set of association rules formalized here as the set

$$\mathcal{R} = \{\langle X_1, Y_1, \boldsymbol{\theta}_1\rangle, \ldots, \langle X_i, Y_i, \boldsymbol{\theta}_i\rangle, \ldots, \langle X_n, Y_n, \boldsymbol{\theta}_n\rangle\},$$

where X_i is the rule antecedent, Y_i is the rule consequent and $\boldsymbol{\theta}_i$ is a vector with available measures of interestingness (e.g., support, confidence, lift) for the i-th rule, $i = 1, \ldots, n$.

A straightforward visualization of association rules is to produce a scatter plot with two interest measures on the axes (see Figure 1(a)). Such a presentation can be found already in an early paper by Bayardo, Jr. and Agrawal (1999) when they discuss *sc-optimal rules.* Scatter plots focus solely on

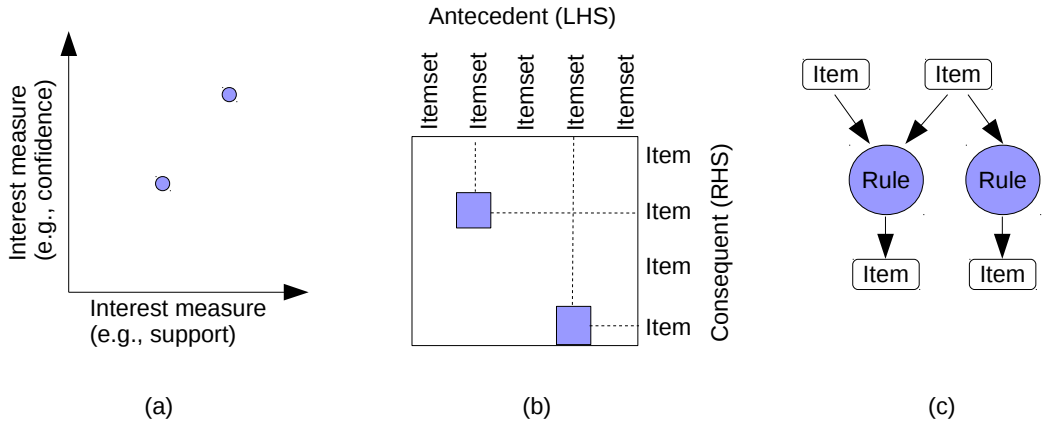


Figure 1: The main components of association rule visualization using (a) a scatter plot, (b) matrix visualization, and (c) graph-based visualization. Rules are shown in color. Color shading can be used to indicate the value of an additional interest measure of the rule (e.g., lift).

Technique	Method (arulesViz)	Set size	Measures	Focus	Interactive features
Scatter plot	`"scatterplot"`	1,000s	3	Interest measures	hover, zoom, pan
Two-Key plot	`"two-key plot"`	1,000s	2 + order	Rule length	hover, zoom, pan
Matrix-based	`"matrix"`	$< 1,000$	1	RHS & LHS	hover, zoom, pan
Grouped matrix	`"grouped matrix"`	100,000s	2	RHS & LHS	drill down, inspect
Graph-based	`"graph"`	100s	2	Items	hover, zoom, pan, brush
Graph-b. (external)	`"graph"`	1,000s	2	Items	tool dependent

Table 1: Interactive visualization methods based on scatter plots, matrix visualization and graphs available in **arulesViz**.

the measures of interestingness $\boldsymbol{\theta}_i$ by choosing two measures (often support and confidence) for the x and y-axis, respectively. A third measure (often lift) can be added to the plot using color. Unwin et al. (2001) introduced a special version of a scatter plot called the *Two-key plot*. Here support and confidence are used for the x and y-axis and the color of the points is used to indicate "order," which is defined as the number of items contained in the rule. Scatter plots can be used for large sets of association rules and give an impression of the distribution of rules concerning large and small values for the chosen interest measures. However, it completely ignores the items in rules and the fact that rules share items. This leads to the issue that two almost identical rules, differing only by a single item, can be located in very different areas on the plot. Standard interactive features for scatter plots can be used. This includes zooming into the plot, panning, and hovering over points to obtain information about the rule it represents. The number of rules that can be effectively visualized and interactively explored (with zooming in) is theoretically not limited, however, for practical purposes it depends mainly on the capability of the display system to render the needed amount of points in an acceptable amount of time. Also overplotting becomes a problem for large rule sets. This typically limits the rule set size to no more than several 1,000 rules.

While the scatter plot focuses on the similarity of rules regarding measures of interestingness like support and confidence, matrix-based visualization for association rules organizes rules in a matrix using distinct antecedent and consequent itemsets as the columns and rows, respectively. The matrix $\mathbf{M}$ is created by identifying the set of A unique antecedents and C unique consequents in $\mathcal{R}$. An $A \times C$ matrix $\mathbf{M} = (m_{ac})$, $a = 1, \ldots, A$ and $c = 1, \ldots, C$, is created with one column for each unique antecedent and one row for each unique consequent. The matrix is populated by setting $m_{ac} = \theta_{i,m}$ where $i = 1, \ldots, n$ is the rule index, m is a chosen interest measure (e.g., lift), and a and c correspond to the position of X_i and Y_i in the matrix. The matrix is displayed using matrix shading, i.e., a color-shaded square at the intersection of the antecedent column and the consequent row of a given rule (Ong et al., 2002). The basic layout is shown in Figure 1(b). If no rule is available for an antecedent/consequent combination, which can easily happen because of the minimum support constraint, then the value in the matrix is undefined and the intersection area in the plot is left blank. Note that association rules in **arules** and most other tools restrict the consequent to a single item, but the size of the antecedent itemset is not restricted. This means that the number of rows in $\mathbf{M}$ is typically much smaller than the number of columns. The order of the rows and columns of the visualized matrix can have a profound impact on the effectiveness of the visualization in guiding the analyst in exploring the rule set. Ong et al. (2002) suggest to reorder antecedents by increasing support and the

consequents by increasing confidence. Two other options are to organize the itemsets by similarity by placing antecedents with similar items close together, or by organizing them so more interesting rules can be easily identified (e.g., placing the rules with highest lift in the top-left corner of the matrix by simply ordering rows and columns by decreasing average lift). Matrix-based visualization is limited in the number of rules it can visualize effectively since large sets of rules typically also have large sets of unique antecedents resulting in a huge matrix which makes exploration more challenging using repeated zooming in and out. This is somewhat mitigated by reordering the matrix, but we still recommend to use less than 1,000 rules.

The grouped matrix-based visualization (Hahsler and Karpienko, 2016) enhances matrix-based visualization by organizing the large set of different antecedents (columns) into a small set of groups via clustering. For grouping, the set of antecedents is split into a set of k groups $\mathcal{S} = \{S_1, S_2, \ldots, S_k\}$ while minimizing the within-cluster sum of squares

$$\text{argmin}_{\mathcal{S}} \sum_{i=1}^{k} \sum_{\mathbf{m}_j \in S_i} ||\mathbf{m}_j - \boldsymbol{\mu}_i||^2,$$

where $\mathbf{m}_j$, $j = 1, \ldots, A$, is a column vector representing all rules with the same antecedent and $\boldsymbol{\mu}_i$ is the center (mean) of the vectors in S_i. We use the k-means algorithm by Hartigan and Wong (1979) and restart it ten times with randomly initialized centers to find a suitable solution. Before clustering, the missing values for antecedent/confidence combinations that do not pass the minimum support, or minimum confidence threshold are replaced by a neutral element (e.g., 1 for lift). The result is a smaller matrix with groups of antecedents as columns. Similar to the regular matrix-based visualization. the matrix is again sorted such that more interesting groups are moved to the top left corner and grouped rules are presented using a balloon plot. For interactive exploration, drilling-down into a group can be easily done by selecting only the rules in the group and applying the same clustering procedure again. Note that the standard matrix visualization is a special case of the grouped visualization with $k = A$.

Graph-based techniques (Klemettinen et al., 1994; Rainsford and Roddick, 2000; Buono and Costabile, 2005; Ertek and Demiriz, 2006) concentrate on the relationship between individual items reflecting their membership in different association rules. Association rules are visualized using two different types of vertices to represent the set of items I (or the subset that is used in the rule set) and the set of rules $\mathcal{R}$, respectively. The edges indicate the relationship in rules. An example is shown in Figure 1(c). Interest measures are typically added to the plot as labels, by color or width of the arrows displaying the edges, or by the size and color of the vertices. For visualization, standard graph drawing algorithms (e.g., force-directed layout algorithms) are used to create the layout. Standard interactive features available for graph visualization (e.g., zooming and panning) can be used. Graph-based visualization offers a very appealing representation of rules but they tend to become cluttered and thus are only viable for small rule sets (typically 100 or less). External tools for network visualization allow more advanced visualization, with tool-dependent interactive features like grouping nodes which may make this visualization useful for even larger rule sets.

In the following, we will present how these visualizations and interactive features can be created and used to analyze association rules with **arulesViz**.

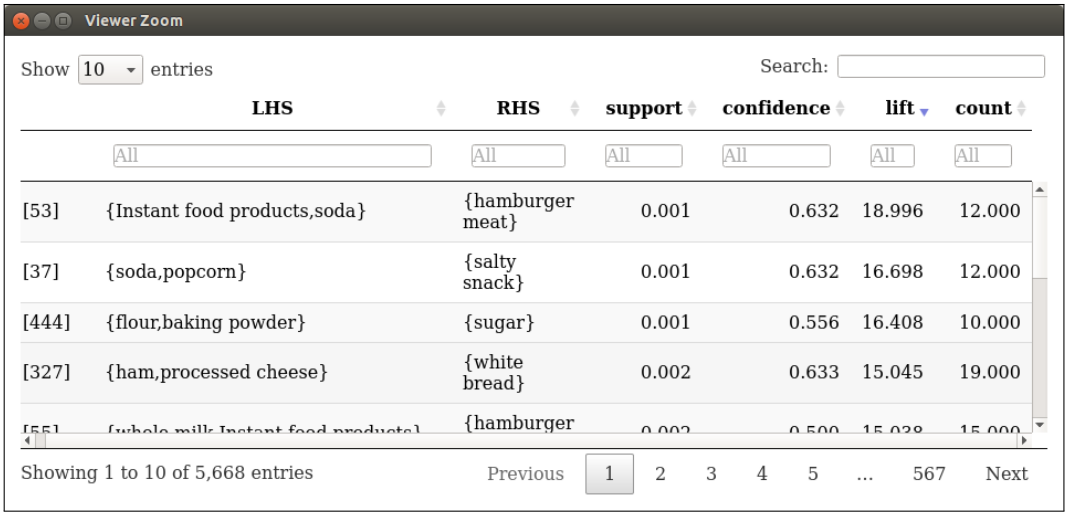
Data preparation and unified interface of arulesViz

The package **arulesViz** (Hahsler, 2017) is part of the **arules** package ecosystem for handling and mining association rules (Hahsler et al., 2011). Considerable effort has been put into providing a straightforward and consistent interface, which allows the user to explore different visualization options easily. Before we start with the visualization, we need to mine some association rules. Throughout this paper, we use a small demo data set called "Groceries" which is included in **arules**. We use this data set so the reader can easily reproduce the presented results. We first load the package and the data set.

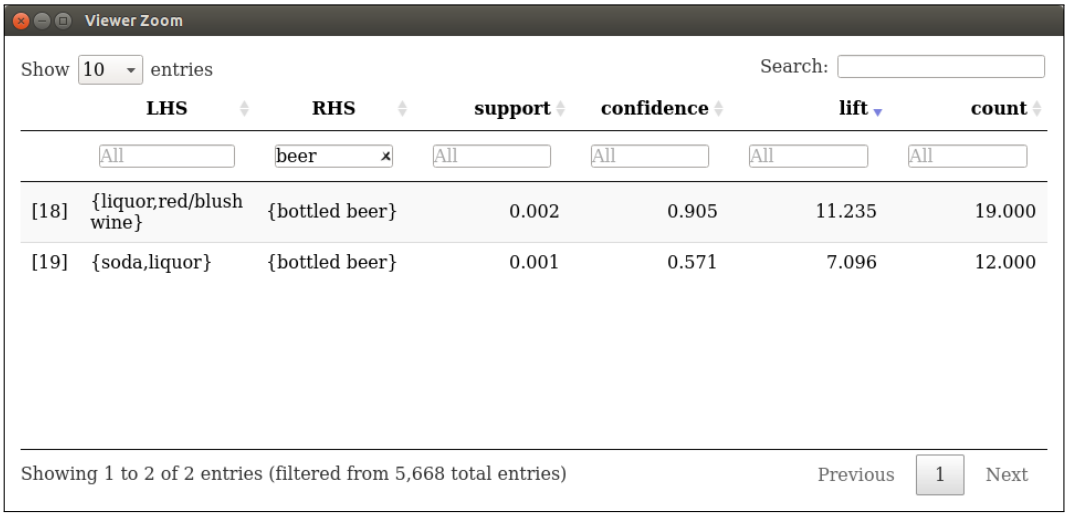
```
library("arulesViz")
data("Groceries")
```

Groceries contains sales data from a small grocery store with 9835 transactions and a moderate number of 169 items (product groups). It is easy to mine association rules using the Apriori algorithm implemented in **arules**. Since the data set is very sparse with each transaction only containing a small fraction of the 169 product groups, we use a very low minimum support threshold of 0.1% of the transactions. To create more rules, we also reduce the minimum confidence threshold from the default value of 80% to 50%.

```
rules <- apriori(Groceries, parameter = list(support = 0.001, confidence = 0.5))
```



(a)



(b)

Figure 2: Interactive data table for the mined rule set. Table (a) shows the rules sorted by lift. The top three rules represent typical barbecue needs including hamburger meat, movie snacks, and baking ingredients, respectively. Table (b) shows the rules filtered with only rules with beer in the consequent (RHS). We see that other alcoholic beverages in the LHS produce high-lift rules.

```
rules
```

```
set of 5668 rules
```

The result is a set of 5668 association rules. Many real applications may include thousands of items and rule sets with millions of rules, but even inspecting all 5668 rules in this example manually is cumbersome. Common practice is to examine only the top rules according to a measure of interestingness which can be done in **arules** using the functions `sort()` and `inspect()`. To make inspection easier, **arulesViz** provides a interactive inspect method which creates a data table using package **DT** (Xie, 2016).

```
inspectDT(rules)
```

Figure 2 shows the result of this interactive inspection method which allows the user to sort rules given different interest measures, specify ranges for measures, and provides filters for items. In the example in Figure 2(a) we sort the rules by lift. We can see that the result is a set of product groups we would expect to be highly correlated. The first three rules represent typical barbecue needs including hamburger meat, movie snacks, and baking ingredients, respectively. In Figure 2(b), we selected only

the rules with 'beer' in the rules consequent (RHS). The rules show very strong relationship (a high lift value) with liquor, soda, and wine, but a rather low support of only 0.2% and 0.1%, respectively.

While interactive inspection of rules using a table is very useful for experts who know what they are looking for, visualization and especially interactive visualization can help to understand the found rules better. Different visualization methods are quite distinct regarding presentation, but in **arulesViz** much work has been spent on creating a single, simple and consistent interface that allows the user to analyze a rule set quickly using different methods. The main interface is the `plot()` method is defined as

```
plot(x, method = NULL, measure = "support", shading = "lift", engine = "default",
    data = NULL, control = NULL, ...)
```

where

- `x` is the set of rules to be visualized,
- `method` is the visualization method (some are given in Table 1 and more can be found in the manual page included in **arulesViz**),
- `measure` and `shading` contain the interest measures used by the plot,
- `engine` was introduced in a recent release to let the user choose between different rendering engines (e.g., using R's grid static graphics, grid-based interactive features or HTML widgets),
- `data` can contain the transaction data set used to mine the rules (only necessary for some methods),
- `control` is a list with visualization-specific control arguments to customize the plot. Using the the control argument `verbose = TRUE` will show all available arguments and the default values for the chosen method and engine. For convenience additional arguments are appended to the control list.

In the following sections, we will introduce the main interactive visualization methods implemented in **arulesViz** with simple examples. Static and additional methods are described in the package documentation.

Scatter plot

A scatter plot for association rules uses two interest measures, one on each of the axes. The default plot for association rules in **arulesViz** is a scatter plot using support and confidence on the axes. The measure defined by shading (default: lift) is visualized by the color of the points. A color key is provided to the right of the plot.

```
plot(rules)
```

The resulting plot for the rules mined in the previous section is shown in Figure 3. We can see that rules with high lift have typically a relatively low support. Bayardo, Jr. and Agrawal (1999) argue that the most interesting rules (sc-optimal rules) reside on the support/confidence border. This can be clearly seen by high-lift rules residing close to the bottom-left corner of the plot which represents the minimum support cut-off. Since support and confidence are the results of counting, rules often share the same support and confidence value, leading to considerable overplotting. The plot function automatically adds some jitter in this case. Any measure stored in the quality slot of the set of rules can be used for the axes (vector of length 2 for parameter `measure`) or for color shading (`shading`). The following measures are available for our set of rules.

```
head(quality(rules))
```

```
  support confidence lift
1 0.00112      0.733 2.87
2 0.00122      0.522 2.84
3 0.00132      0.591 2.31
4 0.00132      0.565 2.21
5 0.00132      0.520 2.04
6 0.00366      0.643 2.52
```

These are the default measures generated by the Apriori implementation used in **arules**. To add other measures, we refer the reader to the function `interestMeasure()` included in **arules**.

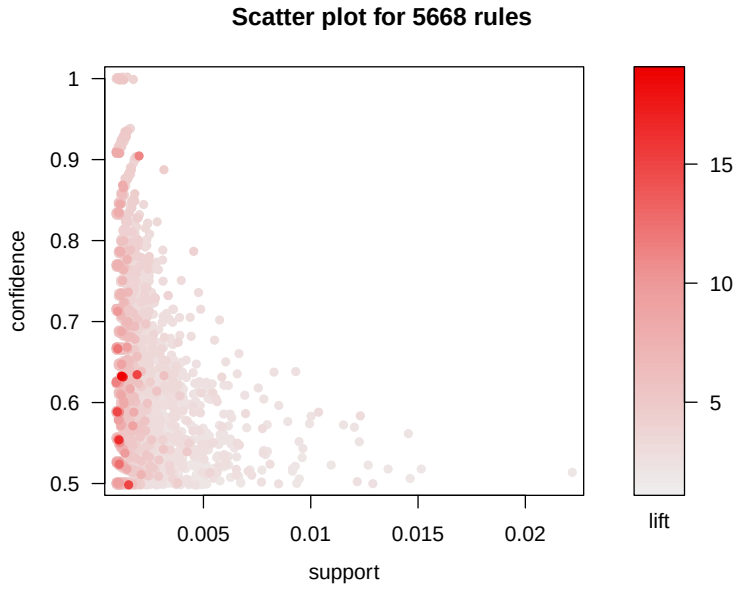


Figure 3: Default scatter plot showing support, confidence, and lift of rules. The plot mainly gives an overview of the distribution of support and confidence in the rule set. There are a few high-confidence rules in the top left corner, and high-lift rules are located close to the minimum support threshold (left corner of the plot).

The default plot is a static visualization providing limited utility for exploration since the individual rules cannot be identified directly in the plot. **arulesViz** offers a JavaScript-based scatter plot visualization using **plotly** (Sievert et al., 2017) which is capable of creating an interactive HTML widget. This visualization can be selected by setting the engine parameter to `"htmlwidget"`, and it supports identifying rules by hovering over a point, zooming in and out, and panning.

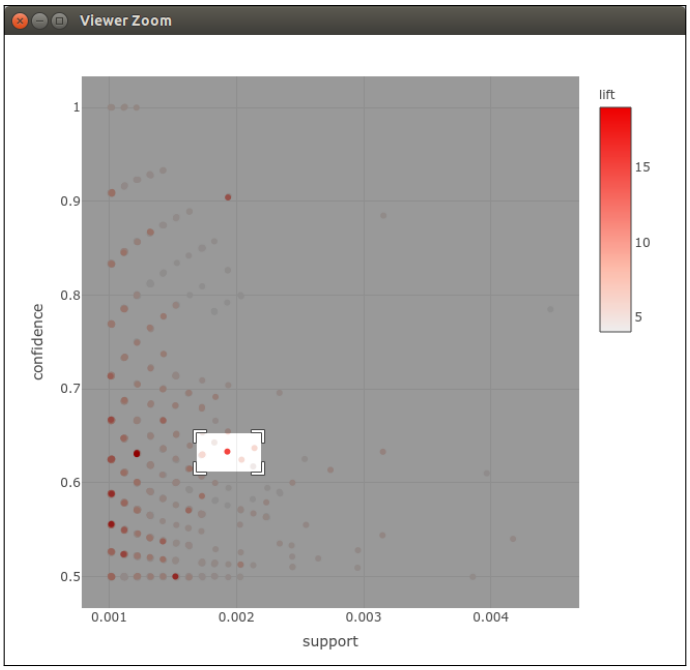
```
plot(rules, engine = "htmlwidget")

Warning message:
      plot: Too many rules supplied. Only plotting the best 1000 rules using measure
      lift (change parameter max if needed)
```

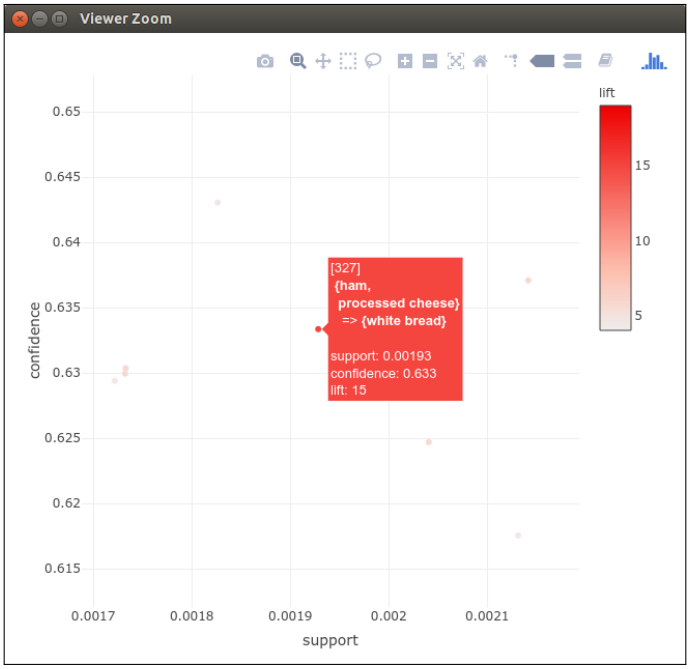
Note that HTML widgets are rendered by the client which gets very slow with many points. This is why the number of rules visualized is by default restricted to the 1,000 rules with the highest value for the measure of interestingness specified in `shading` (default is lift). The call above creates a warning message to notify the user that not all rules are included in the plot. The user can change the limit for the restriction using the control parameter `max`. For example, `max = Inf` visualizes all rules. Figure 4 shows an example of the interactive use of the plot. In Figure 4(a), we zoom into a region of average confidence that contains some high-lift rules (dark red). Figure 4(b) shows the zoomed-in view where rules can be inspected by hovering over the corresponding point. The selected rule has high support and contains the ingredients necessary to make ham and cheese sandwiches. The number in brackets ([327]) is the index in the rule set. Note that the tight groups of rules with low support and a confidence around 0.63 represent rules with the same confidence and support. They are just spread out slightly using jitter to reduce overplotting.

Matrix-based visualization

Matrix-based visualization creates a matrix with unique antecedent and consequent itemsets forming the columns and rows, respectively. The matrix contains the values for a interest measure selected by the analyst and is visualized using matrix shading. The order of rows and columns is arbitrary, however, to improve the ability to analyze the data, we suggest in **arulesViz** to reorder the matrix such that the row averages decrease from top to bottom and the column averages decrease from left to right. This pushes the rules with higher values of interestingness to the top-left position in the plot. Here we plot the rules using the method matrix and render it using an HTML widget.

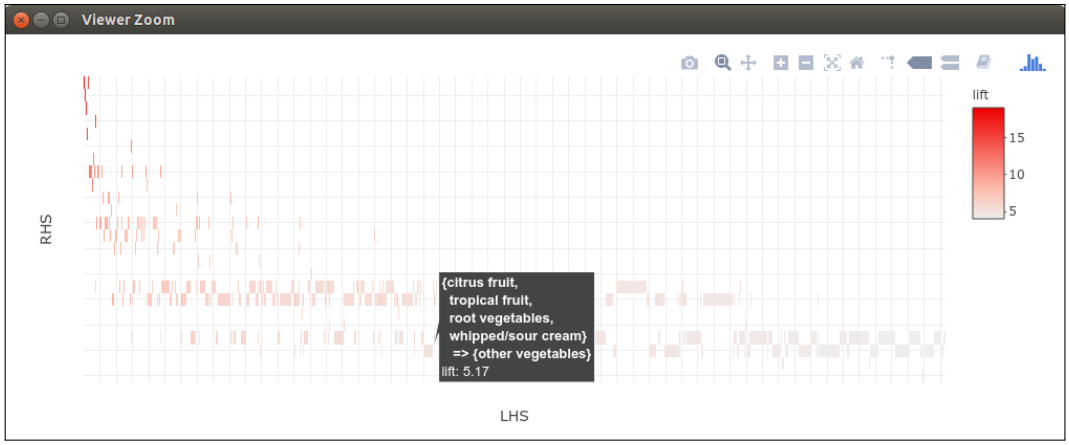


(a)

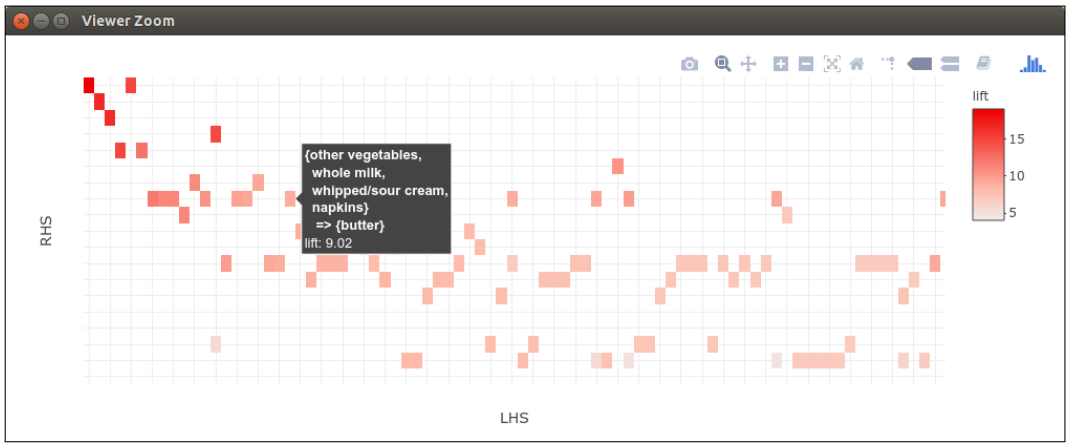


(b)

Figure 4: Interactive scatter plot producing a HTML widget. This plot only visualizes the 1000 rules with the highest lift. Plot (a) shows how the area around high-lift rules is selected for zooming in. Plot (b) showed the selected region and hovering over the high-lift rule reveals that it contains ingredients to make sandwiches.



(a)



(b)

Figure 5: Interactive matrix-based visualization where the matrix is ordered such that the rules with the largest lift values are located close to the top-left corner. Only the 1000 rules with the highest lift are visualized. Hovering over a rule close to the bottom in plot (a) shows that the row contains many rules with the consequent "root vegetables." Plot (b) is zoomed into the top-left corner. The highlighted rule has the consequent "butter". There are many entries in the row indicating that many strong rules are resulting in butter.

```
plot(rules, method = "matrix", engine = "htmlwidget")

Warning message:
      plot: Too many rules supplied. Only plotting the best 1000 rules using
      lift (change parameter max if needed)
```

The warning message shows again that only the top 1,000 rules are included in the visualization for performance reasons. The resulting plot is shown in Figure 5. Rows represent rules with the same consequent. A rule from the row with the consequent "root vegetables" is highlighted in Figure 5(a). There are many rules with this consequent and the lift is still relatively high, since only the 1,000 rules with the highest lift are visualized. The plot guides the analyst to the rules with the highest lift by placing them in the top-left corner. Figure 5(b) shows the plot zoomed into the top-left corner. A rule is inspected and contains "butter". The many other entries in the same row indicate that butter is part of many strong rules. Other rows can easily be inspected by hovering over a rule in that row.

Matrix-based visualization is limited in the number of rules it can visualize effectively since large sets of rules typically also have large sets of unique antecedents. This would require the analyst to repeatedly zoom in and out. Grouped matrix-based visualization (Hahsler and Karpienko, 2016) enhances matrix-based visualization by grouping antecedents of rules via clustering and sorting rules by "interestingness" to handle a larger number of rules. Grouped rules are presented as an aggregate in a matrix that is visualized as a balloon plot.

```
plot(rules, method = "grouped matrix", engine = "interactive")
```

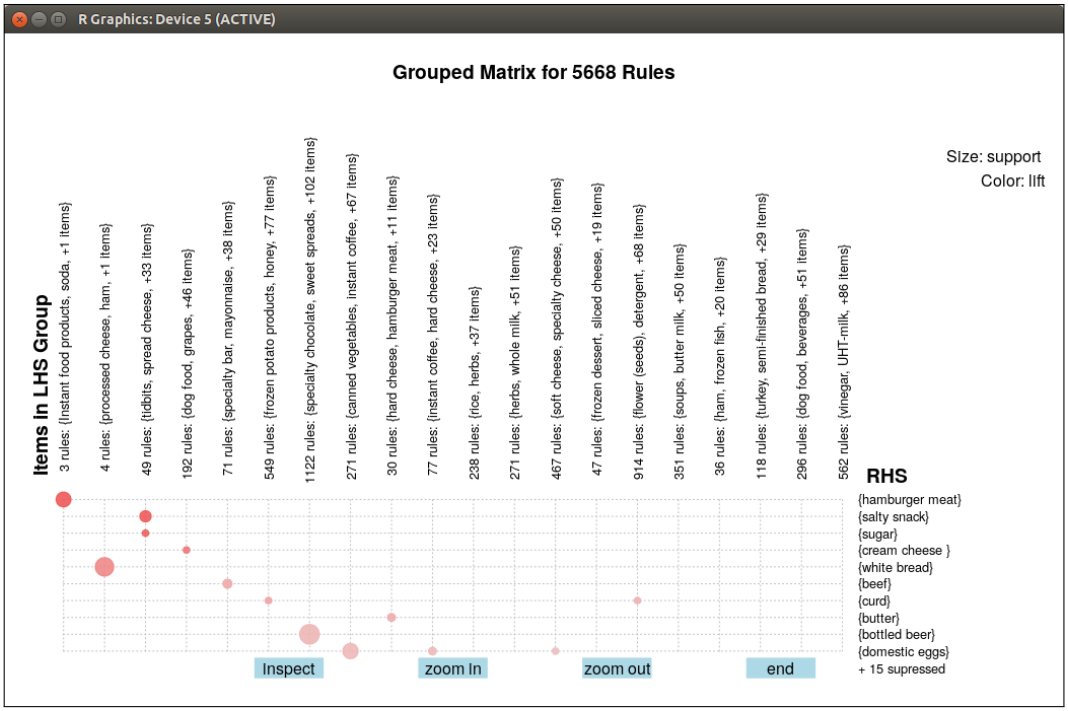


Figure 6: Grouped matrix-based visualization. The highest interest group (top-left hand corner) consists of 3 rules which contain"Instant food product", "soda" and an additional item in the antecedent and all rules have the consequent "hamburger meat."

This plot can easily visualize larger sets of rules. The resulting visualization of the set of 5668 rules mined earlier is shown in Figure 6. The visualization is very similar to the regular matrix visualization, but the columns represent groups of antecedents. The plot is again organized such that the most interesting rules according to lift (the default measure of interestingness) are shown in the top-left corner. The highest interest group consists of 3 rules which contain "Instant food products, " and "soda" and an additional item in the antecedent and the consequent is "hamburger meat." The balloon size represents support and the color indicates lift. We also see below all RHS items, that the plot suppresses 15 consequent items representing rules with low lift values to create a less convoluted plot. The number of items shown for the antecedent groups and how many consequent items, if any, are suppressed can be specified by the user.

The interactive features of the grouped matrix visualization allow the user to identify the rules making up groups and also to zoom into and out of groups. The visualization is currently only implemented using interactive features of the **grid** graphics system (`engine = "interactive"`), and a HTML widget version is planned for the future.

Graph-based visualization

Graph-based techniques concentrate on the relationship between individual items in the rule set. **arulesViz** offers an interactive visualization based on package **visNetwork** (Almende B.V. et al., 2017).

```
plot(rules, method = "graph", engine = "htmlwidget")

Warning message:
      plot: Too many rules supplied. Only plotting the best 100 rules using
      lift (change control parameter max if needed)
```

Graph-based visualization tend to become cluttered and thus are only viable for very small sets of rules. The warning message informs the user that only the top 100 rules are included in the visualization. Figure 7(a) shows the initial view with the graph centered. The force-directed layout used in the visualization moves items which are included in many rules and rules which share many items towards the center of the plot. Items which are in very few rules are pushed to the periphery of the plot. It is interesting to note that high-lift rules are also on the outside of the plot. This is due to the fact that rules with high lift levels typically appear at the minimum support/confidence boundary and low-support items are part of fewer rules and thus are pushed to the periphery of the graph. The

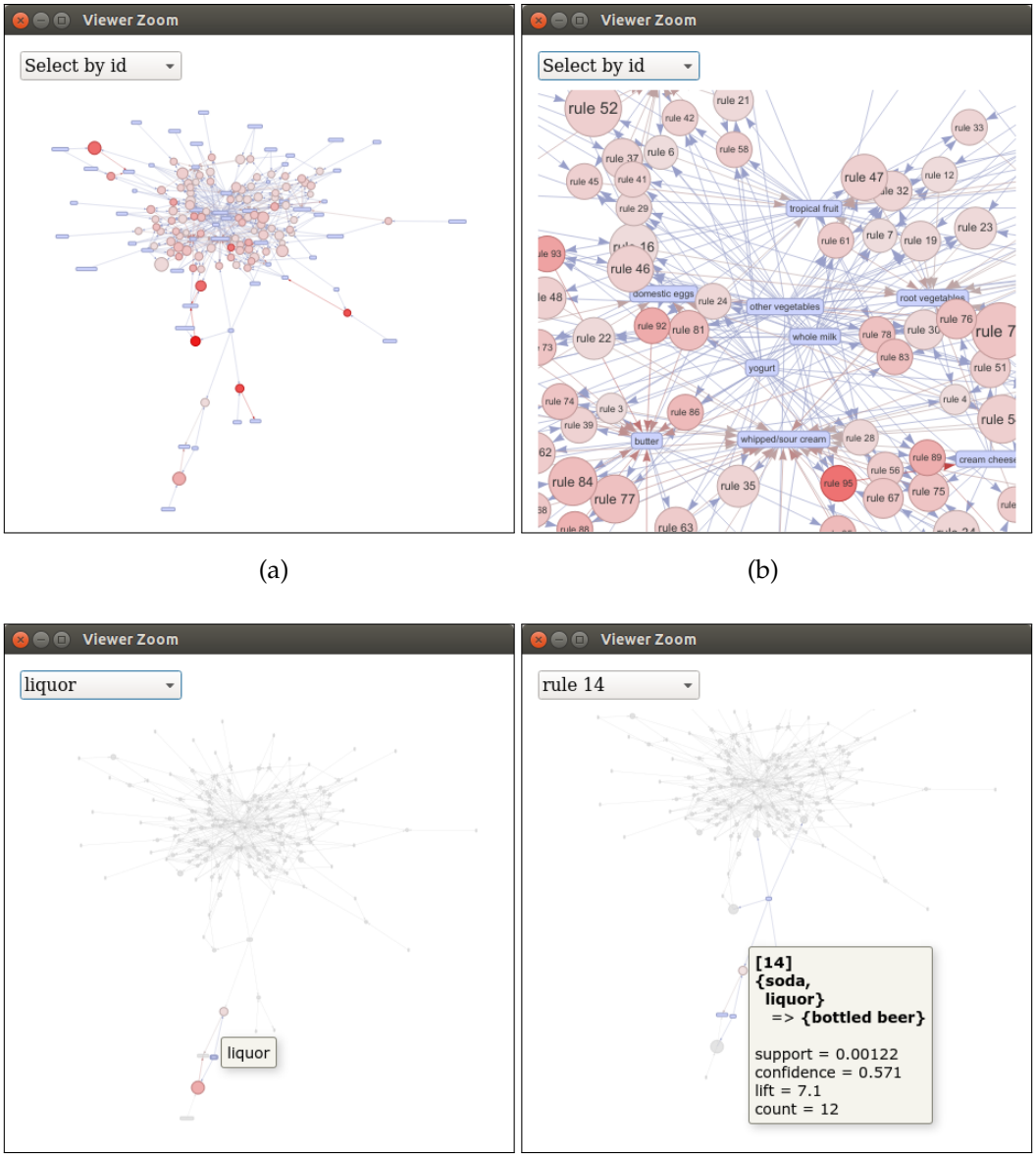


Figure 7: Graph-based visualization with items and rules as vertices. Plot (a) shows the initial view of the complete graph. Plot (b) uses zooming in to show the center of the graph, revealing the most frequent items (including "other vegetables" and "milk"). Plot (c) shows how to use the pull-down menu in the top-left corner to find the item "liquor" in the graph. Plot (d) inspects a single rule that is connected to "liquor."

visualization suppresses labels when they become too small to be readable. Figure 7(b) uses zooming in to show the center of the graph. The items in the center ("other vegetables", "whole milk" and "yogurt") are the most frequently bought items and are thus part of many rules. These items are often not very interesting because domain experts typically are aware of how the most common items relate to each other. Figure 7(c) shows how the pull-down menu in the top-left corner can be used to analyze how a given item relates in the rule set with other items. In the example, the item "liquor" is selected and the visualization highlights the two rules the item is part of. Figure 7(d) inspects one of the rules. A useful strategy is to inspect all high-lift rules and the outside area of the graph to find interesting rules. For some data sets, the rules also form two or more weakly or even unconnected components indicating that the items in the groups have only little affect on each other.

A big restriction of graph-based visualization for association rules is that they are only useful for a very small set of rules. To explore large sets of rules with graphs, the analyst needs to be supported by advanced interactive features like filtering, grouping, and coloring nodes. Such features are available in interactive visualization and exploration platforms for networks and graphs like *Gephi* (Bastian

et al., 2009). **arulesViz** can export graphs for sets of association rules in the GraphML format or as a Graphviz dot-file to be explored in tools like Gephi. For example, the complete rule set can be exported by

```
saveAsGraph(rules, file = "rules.graphml")
```

This file can be imported directly into most tools for analysis.

Conclusion

While each of the three basic groups of visualizations reveal similar information, they focus on different properties of the analyzed rule set and some can visualize larger rule sets than others effectively. A single visualization by itself is typically not sufficient to understand all aspects of a rule set, but repeated use of different methods can lead the analyst to deeper insight into the data. Analysis typically starts with creating a scatter plot to inspect the rules with extreme values for support, confidence and lift. The advantage is that this visualization can deal with relatively large rule sets. Alternatively, an interactive table can also be used for this task. A grouped matrix plot can then be used to inspect the rules and group of rules. Finally, graph-based visualization can be employed to get a deeper understanding of a smaller set of rules and items. This visualization is especially useful to present found results because it is easy to understand for non-analysts. **arulesViz** makes this process easier by providing a simple and unified `plot` method, where different visualizations can be explored by just changing the method argument. Many methods also can create JavaScript-based HTML widgets (Vaidyanathan et al., 2017) which can be saved as HTML files, included in web-based applications using shiny (Chang et al., 2017), or used in interactive documents created with R markdown (Allaire et al., 2017).

Bibliography

R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 207–216. ACM Press, 1993. URL https://doi.org/10.1145/170035.170072. [p163]

J. Allaire, J. Cheng, Y. Xie, J. McPherson, W. Chang, J. Allen, H. Wickham, A. Atkins, R. Hyndman, and R. Arslan. *Rmarkdown: Dynamic Documents for R*, 2017. URL https://CRAN.R-project.org/package=rmarkdown. R package version 1.6. [p174]

Almende B.V., B. Thieurmel, and T. Robert. *visNetwork: Network Visualization Using 'vis.js' Library*, 2017. URL https://CRAN.R-project.org/package=visNetwork. R package version 2.0.1. [p172]

M. Bastian, S. Heymann, and M. Jacomy. Gephi: An open source software for exploring and manipulating networks. In *International AAAI Conference on Weblogs and Social Media*, pages 361–362, 2009. [p173]

R. J. Bayardo, Jr. and R. Agrawal. Mining the most interesting rules. In *KDD '99: Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 145–154. ACM, 1999. URL https://doi.org/10.1145/312129.312219. [p164, 168]

S. Brin, R. Motwani, J. D. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket data. In *SIGMOD 1997, Proceedings ACM SIGMOD International Conference on Management of Data*, pages 255–264, Tucson, Arizona, USA, 1997. URL https://doi.org/10.1145/253262.253325. [p164]

D. Bruzzese and C. Davino. Visual mining of association rules. In *Visual Data Mining: Theory, Techniques and Tools for Visual Analytics*, pages 103–122. Springer-Verlag, 2008. URL https://doi.org/10.1007/978-3-540-71080-6_8. [p163, 164]

P. Buono and M. F. Costabile. Visualizing association rules in a framework for visual data mining. In *From Integrated Publication and Information Systems to Virtual Information and Knowledge Environments*, pages 221–231, 2005. URL https://doi.org/10.1007/978-3-540-31842-2_22. [p166]

W. Chang, J. Cheng, J. Allaire, Y. Xie, and J. McPherson. *Shiny: Web Application Framework for R*, 2017. URL https://CRAN.R-project.org/package=shiny. R package version 1.0.5. [p174]

C. H. Chen, A. Unwin, and W. Hardle, editors. *Handbook of Data Visualization*. Springer Handbooks of Computational Statistics. Springer-Verlag, 2008. URL https://doi.org/10.1007/978-3-540-33037-0. [p163]

G. Ertek and A. Demiriz. A framework for visualizing association mining results. In *ISCIS*, pages 593–602, 2006. URL https://doi.org/10.1007/11902140_63. [p166]

M. Hahsler. *arulesViz: Visualizing Association Rules and Frequent Itemsets*, 2017. URL http://CRAN.R-project.org/package=arulesViz. R package version 1.3-0. [p163, 164, 166]

M. Hahsler and R. Karpienko. Visualizing association rules in hierarchical groups. *Journal of Business Economics*, pages 1–19, 2016. URL https://doi.org/10.1007/s11573-016-0822-8. [p166, 171]

M. Hahsler, B. Grün, and K. Hornik. Arules – A computational environment for mining association rules and frequent item sets. *Journal of Statistical Software*, 14(15):1–25, 2005. URL https://doi.org/10.18637/jss.v014.i15. [p163, 164]

M. Hahsler, S. Chelluboina, K. Hornik, and C. Buchta. The arules R-package ecosystem: Analyzing interesting patterns from large transaction datasets. *Journal of Machine Learning Research*, 12:1977–1981, 2011. URL http://jmlr.csail.mit.edu/papers/v12/hahsler11a.html. [p166]

M. Hahsler, C. Buchta, B. Grün, and K. Hornik. *Arules: Mining Association Rules and Frequent Itemsets*, 2017. URL http://CRAN.R-project.org/. R package version 1.5-3. [p164]

J. A. Hartigan and M. A. Wong. A k-means clustering algorithm. *Applied Statistics*, 28(1):100–108, 1979. URL https://doi.org/10.2307/2346830. [p166]

M. Klemettinen, H. Mannila, P. Ronkainen, H. Toivonen, and A. I. Verkamo. Finding interesting rules from large sets of discovered association rules. In *CIKM*, pages 401–407, 1994. URL https://doi.org/10.1145/191246.191314. [p166]

K.-H. Ong, K.-L. Ong, W.-K. Ng, and E.-P. Lim. Crystalclear: Active visualization of association rules. In *In ICDM'02 International Workshop on Active Mining AM2002*, 2002. [p165]

M. E. Prangsmal, C. A. M. van Boxtel, G. Kanselaar, and P. A. Kirschner. Concrete and abstract visualizations in history learning tasks. *British Journal of Educational Psychology*, 79:371–387, 2009. URL https://doi.org/10.1348/000709908x379341. [p163]

C. P. Rainsford and J. F. Roddick. Visualisation of temporal interval association rules. In *IDEAL '00: Proceedings of the Second International Conference on Intelligent Data Engineering and Automated Learning, Data Mining, Financial Engineering, and Intelligent Agents*, pages 91–96. Springer-Verlag, 2000. [p166]

C. Sievert, C. Parmer, T. Hocking, S. Chamberlain, K. Ram, M. Corvellec, and P. Despouy. *Plotly: Create Interactive Web Graphics via 'plotly.js'*, 2017. URL https://CRAN.R-project.org/package=plotly. R package version 4.7.1. [p169]

P.-N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining, (First Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006. ISBN 0321321367. [p163]

A. Unwin, H. Hofmann, and K. Bernt. The twokey plot for multiple association rules control. In *PKDD '01: Proceedings of the 5th European Conference on Principles of Data Mining and Knowledge Discovery*, pages 472–483. Springer-Verlag, 2001. URL https://doi.org/10.1007/3-540-44794-6_39. [p165]

R. Vaidyanathan, Y. Xie, J. Allaire, J. Cheng, and K. Russell. *Htmlwidgets: HTML Widgets for R*, 2017. URL https://CRAN.R-project.org/package=htmlwidgets. R package version 0.9. [p174]

Y. Xie. *DT: A Wrapper of the JavaScript Library 'DataTables'*, 2016. URL https://CRAN.R-project.org/package=DT. R package version 0.2. [p167]

Michael Hahsler
Engineering Management, Information, and Systems
Lyle School of Engineering
Southern Methodist University
P.O. Box 750123
Dallas, TX 75275-0123 USA mhahsler@lyle.smu.edu

ManlyMix: An R Package for Manly Mixture Modeling

by Xuwen Zhu, Volodymyr Melnykov

Abstract Model-based clustering is a popular technique for grouping objects based on a finite mixture model. It has countless applications in different fields of study. The R package **ManlyMix** implements the Manly mixture model that allows modeling skewness within data groups and performs cluster analysis. **ManlyMix** is a powerful diagnostics tool that is capable of conducting investigation concerning the normality of variables upon fitting of a Manly forward or backward model. Theoretical foundations as well as description of functions are provided. All features of the package are illustrated with examples in great detail. The analysis of real-life datasets demonstrates the flexibility and usefulness of the package.

Introduction

Finite mixture models provide a powerful tool to model heterogeneous data. Their flexibility, close connection to cluster analysis, and interpretability make them increasingly appealing to researchers and practitioners these days. The applications of finite mixture modeling can be found in all fields, including medicine (Schlattmann, 2009), transportation (Park and Lord, 2009), dendrochronology (Michael and Melnykov, 2016), and environment science (Gillespie and Neale, 2006), just to name a few.

The Bayes decision rule, applied to posterior probabilities obtained in the course of fitting a mixture model, yields a clustering result. Such a procedure is called model-based clustering. It assumes the existence of a one-to-one correspondence between each distribution in the mixture model and underlying data group.

If all components in the model are Gaussian distributions, the mixture is called a Gaussian mixture model. Gaussian mixtures are very popular among practitioners due to their interpretability and simplicity. However, when there is severe skewness in data, Gaussian mixtures models do not provide a good fit to the data. As a result, model-based clustering might produce unsatisfactory results. In such cases, more flexible mixtures should be adopted. Some existing software packages that provide such functionality are listed in Table 1. Here, **mixsmsn**, **EMMIXskew**, and **EMMIXuskew** packages are based on skew-normal and skew-t distributions, which are popular choices for modeling skewed data. On the other hand, **flowClust** is the only package that implements a transformation-based mixture model. It relies on the celebrated Box-Cox transformation to near-normality applied to all dimensions within the same mixture component. This leads to extra K parameters λ_k in the resulting mixture. The package is shown to model flow cytometry data effectively. In many applications, however, it is reasonable to assume that transformation parameters can vary not only from component to component but also from variable to variable. In this paper, we introduce the R package **ManlyMix** (Zhu and Melnykov, 2016b), which provides readers with an alternative approach to modeling and clustering skewed data. Manly mixture models (Zhu and Melnykov, 2016a) are constructed based on the Manly back-transformation applied to each variable in multivariate Gaussian components.

The **ManlyMix** package implements several functions associated with Manly mixture models including the core function for running the EM algorithm, the forward and backward model selection procedure for eliminating unnecessary transformation parameters, and the Manly K-means algorithm, which serves as an extension of the traditional K-means. Other capabilities of the package include computing a Manly mixture overlap, simulating datasets from a Manly mixture, constructing density or contour plots for a fitted model, and assessing the variability of estimated parameters. The highlights of **ManlyMix** include:

Package	Mixture components
flowClust (Lo et al., 2009)	t mixture with Box-Cox transformation
mixsmsn (Prates et al., 2013)	scale skew-normal and skew-t
EMMIXskew (Wang et al., 2013)	restricted skew-normal and skew-t
EMMIXuskew (Lee and McLachlan, 2014)	unrestricted skew-t

Table 1: Existing R packages for mixture modeling of skewed data.

- providing an alternative approach to modeling heterogeneous skewed data;
- calling core functions from C for speed;
- providing excellent model interpretability through output of skewness parameters;
- preventing overfitting of the data by implementing model selection algorithms;
- offering effective assessment of mixture characteristics through the overlap calculation and variability assessment.

This paper is organized in the following way. A brief introduction to the Expectation-Maximization (EM) algorithm for Manly mixture models as well as the classification Expectation-Maximization (CEM) algorithm for Manly K-means is provided in the second section. In section "Package functionality and illustrative examples", a comprehensive description of all functions in **ManlyMix** is given along with the analysis of two real-life datasets. All features of the package are illustrated in great detail. Demo examples are constructed in section four for users to conduct further investigation of **ManlyMix**. In the last section, we provide a brief summary for the paper.

Methodological and algorithmic details

Manly mixture model

Consider a dataset $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_n$ of size n, where $\boldsymbol{X}_i$'s are p-variate independent observations that are identically distributed. The exponential (Manly) transformation to near normality is defined by

$$\mathcal{M}(\boldsymbol{X}; \boldsymbol{\lambda}) = \left(\frac{e^{\lambda_1 X_1} - 1}{\lambda_1}, \ldots, \frac{e^{\lambda_p X_p} - 1}{\lambda_p} \right)^T,$$

where the distribution of $\mathcal{M}(\boldsymbol{X}; \boldsymbol{\lambda}_k)$ can be effectively approximated by multivariate normal distribution for an appropriate choice of $\boldsymbol{\lambda}$ (Manly, 1976). $\mathcal{M}^{-1}$ represents the Manly back-transformation. This leads to a so-called Manly mixture model given by

$$g(\boldsymbol{x}; \boldsymbol{\Psi}) = \sum_{k=1}^{K} \tau_k \phi(\mathcal{M}(\boldsymbol{x}; \boldsymbol{\lambda}_k); \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \exp\{\boldsymbol{\lambda}_k^T \boldsymbol{x}\}, \tag{1}$$

where K is the number of components in the model, τ_k's are mixing proportions such that $\sum_{k=1}^{K} \tau_k = 1$, and $\boldsymbol{\lambda}_k = \left(\lambda_{k1}, \ldots, \lambda_{kp}\right)^T$ is a p-dimensional skewness vector which controls the transformation of the kth component. $\phi(\cdot; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ is the p-variate normal probability density function. $\boldsymbol{\mu}_k$ and $\boldsymbol{\Sigma}_k$ are the mean vector and variance-covariance matrix of the kth component after transformation. $\boldsymbol{\Psi}$, as the entire parameter vector, includes τ_k's, $\boldsymbol{\mu}_k$'s and $\boldsymbol{\Sigma}_k$'s.

To find the MLE of the parameter vector $\boldsymbol{\Psi}$, the Expectation-Maximization (EM) algorithm (Dempster et al., 1977; McLachlan and Krishnan, 2008) needs to be employed. Each iteration of the EM algorithm consists of two steps, the E-step and M-step. Let s denote the iteration number. The E-step computes the posterior probabilities

$$\pi_{ik}^{(s)} = \frac{\tau_k^{(s-1)} \phi(\mathcal{M}(\boldsymbol{x}_i; \boldsymbol{\lambda}_k^{(s-1)}); \boldsymbol{\mu}_k^{(s-1)}, \boldsymbol{\Sigma}_k^{(s-1)}) \exp\{(\boldsymbol{\lambda}_k^{(s-1)})^T \boldsymbol{x}_i\}}{\sum_{k'=1}^{K} \tau_{k'}^{(s-1)} \phi(\mathcal{M}(\boldsymbol{x}_i; \boldsymbol{\lambda}_{k'}^{(s-1)}); \boldsymbol{\mu}_{k'}^{(s-1)}, \boldsymbol{\Sigma}_{k'}^{(s-1)}) \exp\{(\boldsymbol{\lambda}_{k'}^{(s-1)})^T \boldsymbol{x}_i\}} \tag{2}$$

based on the the parameter vector from the previous step, $\boldsymbol{\Psi}^{(s-1)}$. The M-step updates the parameters in each iteration. The closed-form expressions are available for the parameters $\tau_k^{(s)}$, $\boldsymbol{\mu}_k^{(s)}$, $\boldsymbol{\Sigma}_k^{(s)}$ and are given by

$$\begin{aligned} \tau_k^{(s)} &= \frac{\sum_{i=1}^{n} \pi_{ik}^{(s)}}{n}, \qquad \boldsymbol{\mu}_k^{(s)} = \frac{\sum_{i=1}^{n} \pi_{ik}^{(s)} \mathcal{M}(\boldsymbol{x}_i; \boldsymbol{\lambda}_k^{(s)})}{\sum_{i=1}^{n} \pi_{ik}^{(s)}}, \quad \text{and} \\ \boldsymbol{\Sigma}_k^{(s)} &= \frac{\sum_{i=1}^{n} \pi_{ik}^{(s)} (\mathcal{M}(\boldsymbol{x}_i; \boldsymbol{\lambda}_k^{(s)}) - \boldsymbol{\mu}_k^{(s)})(\mathcal{M}(\boldsymbol{x}_i; \boldsymbol{\lambda}_k^{(s)}) - \boldsymbol{\mu}_k^{(s)})^T}{\sum_{i=1}^{n} \pi_{ik}^{(s)}}. \end{aligned} \tag{3}$$

For $\boldsymbol{\lambda}_k$, closed-form solution is not available and Nelder-Mead numerical optimization of the function

$$Q_k(\boldsymbol{\lambda}_k|\boldsymbol{\Psi}^{(s)})(\boldsymbol{\lambda}_k) = \sum_{i=1}^{n} \pi_{ik}^{(s)} \Bigg\{ \log \phi \Bigg(\mathcal{M}(\boldsymbol{x}_i;\boldsymbol{\lambda}_k); \sum_{i=1}^{n} \pi_{ik}^{(s)} \mathcal{M}(\boldsymbol{x}_i;\boldsymbol{\lambda}_k) / \sum_{i=1}^{n} \pi_{ik}^{(s)}, \\ \sum_{i=1}^{n} \pi_{ik}^{(s)} \Big(\mathcal{M}(\boldsymbol{x}_i;\boldsymbol{\lambda}_k) - \frac{\sum_{i=1}^{n} \pi_{ik}^{(s)} \mathcal{M}(\boldsymbol{x}_i;\boldsymbol{\lambda}_k)}{\sum_{i=1}^{n} \pi_{ik}^{(s)}} \Big) \Big(\mathcal{M}(\boldsymbol{x}_i;\boldsymbol{\lambda}_k) - \frac{\sum_{i=1}^{n} \pi_{ik}^{(s)} \mathcal{M}(\boldsymbol{x}_i;\boldsymbol{\lambda}_k)}{\sum_{i=1}^{n} \pi_{ik}^{(s)}} \Big)^T \Bigg) \\ + {\boldsymbol{\lambda}_k}^T \boldsymbol{x}_i \Bigg\} + const \tag{4}$$

gives us the estimated parameter vector.

The EM algorithm could be started with an initial partition of the data passed into the M-step. Or the E-step is run first with initial parameters $\tau_k^{(0)}, \boldsymbol{\mu}_k^{(0)}, \boldsymbol{\Sigma}_k^{(0)}, \boldsymbol{\lambda}_k^{(0)}$. The algorithm stops when the convergence criterion is met. In the R package **ManlyMix**, we monitor the relative difference between Q-function values from two consecutive steps. If it is smaller than a user specified tolerance level, $1e-5$ by default, the algorithm stops. This is a speedier choice due to the fact that Q-function values are immediately available after numeric optimization of Equation 4. Such criterion is similar to monitoring the relative difference between log-likelihood values. Upon convergence, the Bayes decision rule assigns each observation to its cluster according to the maximized posterior probabilities from the last E-step. The estimated label of the ith observation is given by

$$\hat{Z}_i = \text{argmax}_k \hat{\pi}_{ik}. \tag{5}$$

In **ManlyMix**, the function `Manly.EM()` runs the EM algorithm for a Manly mixture model and returns estimated model parameters, posterior probabilities, as well as a classification vector. This function is constructed in C for computational efficiency.

Pairwise overlap

Pairwise overlap, introduced by Maitra and Melnykov (2010), is a measure of the interaction between two mixture components. If we denote ω_{k_1,k_2} as the pairwise overlap of components k_1 and k_2, it is defined as the sum of two misclassification probabilities

$$\omega_{k_1,k_2} = \omega_{k_1|k_2} + \omega_{k_2|k_1}, \tag{6}$$

where $\omega_{k_1|k_2}$ represents the probability that a random variable $\boldsymbol{X}$ is mistakenly classified to group k_1 while it came from the component k_2. For a Manly mixture, $\omega_{k_1|k_2}$ can be written as

$$\begin{aligned} \omega_{k_1|k_2} &= \Pr\left[\frac{\phi(\mathcal{M}(\boldsymbol{X};\boldsymbol{\lambda}_{k_1});\boldsymbol{\mu}_{k_1},\boldsymbol{\Sigma}_{k_1}) \exp\{\boldsymbol{\lambda}_{k_1}^T \boldsymbol{X}\}}{\phi(\mathcal{M}(\boldsymbol{X};\boldsymbol{\lambda}_{k_2});\boldsymbol{\mu}_{k_2},\boldsymbol{\Sigma}_{k_2}) \exp\{\boldsymbol{\lambda}_{k_2}^T \boldsymbol{X}\}} > \frac{\tau_{k_2}}{\tau_{k_1}} \middle| \boldsymbol{X} \sim \phi(\mathcal{M}(\boldsymbol{x};\boldsymbol{\lambda}_{k_2});\boldsymbol{\mu}_{k_2},\boldsymbol{\Sigma}_{k_2}) \exp\{\boldsymbol{\lambda}_{k_2}^T \boldsymbol{x}\} \right] \\ &= \Pr\Bigg[-\frac{1}{2}(\mathcal{M}(\mathcal{M}^{-1}(\boldsymbol{Y};\boldsymbol{\lambda}_{k_2});\boldsymbol{\lambda}_{k_1}) - \boldsymbol{\mu}_{k_1})^T \boldsymbol{\Sigma}_{k_1}^{-1} (\mathcal{M}(\mathcal{M}^{-1}(\boldsymbol{Y};\boldsymbol{\lambda}_{k_2});\boldsymbol{\lambda}_{k_1}) - \boldsymbol{\mu}_{k_1}) + \boldsymbol{\lambda}_{k_1}^T \mathcal{M}^{-1}(\boldsymbol{Y};\boldsymbol{\lambda}_{k_2}) \\ &\quad + \frac{1}{2}(\boldsymbol{Y} - \boldsymbol{\mu}_{k_2})^T \boldsymbol{\Sigma}_{k_2}^{-1} (\boldsymbol{Y} - \boldsymbol{\mu}_{k_2}) - \boldsymbol{\lambda}_{k_2}^T \mathcal{M}^{-1}(\boldsymbol{Y};\boldsymbol{\lambda}_{k_2}) > \log\left(\frac{\tau_{k_2} |\boldsymbol{\Sigma}_{k_1}|^{1/2}}{\tau_{k_1} |\boldsymbol{\Sigma}_{k_2}|^{1/2}} \right) \Bigg| \boldsymbol{Y} \sim MVN(\boldsymbol{\mu}_{k_2}, \boldsymbol{\Sigma}_{k_2}) \Bigg]. \end{aligned} \tag{7}$$

In **ManlyMix**, function `Manly.overlap()` estimates $\omega_{k_1|k_2}$ by sampling from corresponding distributions.

Variability assessment

The variability assessment of parameter estimates from Manly mixture model can be made by taking the inverse of the empirical observed information matrix $\boldsymbol{I}_e(\hat{\boldsymbol{\Psi}})$ (McLachlan and Basford, 1988) given by

$$\boldsymbol{I}_e(\hat{\boldsymbol{\Psi}}) = \sum_{i=1}^{n} \nabla q_i(\hat{\boldsymbol{\Psi}}) \nabla q_i^T(\hat{\boldsymbol{\Psi}}), \tag{8}$$

where $q_i(\mathbf{\Psi}) = \sum_{k=1}^{K} \pi_{ik} \left[\log \tau_k + \log \phi(\mathcal{M}(\boldsymbol{x}_i; \boldsymbol{\lambda}_k); \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) + \boldsymbol{\lambda}_k \boldsymbol{x}_i\right]$ and ∇ stands for the gradient operator. We take partial derivatives in the gradient vector $\nabla q_i(\mathbf{\Psi})$ and obtain

$$\frac{\partial q_i(\mathbf{\Psi})}{\partial \tau_k} = \frac{\pi_{ik}}{\tau_k} - \frac{\pi_{iK}}{\tau_K}, \qquad \frac{\partial q_i(\mathbf{\Psi})}{\partial \boldsymbol{\mu}_k} = \pi_{ik} \boldsymbol{\Sigma}_k^{-1} \left(\mathcal{M}(\boldsymbol{x}_i; \boldsymbol{\lambda}_k) - \boldsymbol{\mu}_k\right),$$

$$\frac{\partial q_i(\mathbf{\Psi})}{\partial \text{vech}\{\boldsymbol{\Sigma}_k\}} = \boldsymbol{G}^T \text{vec}\left\{\frac{\pi_{ik}}{2} \boldsymbol{\Sigma}_k^{-1} \left((\mathcal{M}(\boldsymbol{x}_i; \boldsymbol{\lambda}_k) - \boldsymbol{\mu}_k)(\mathcal{M}(\boldsymbol{x}_i; \boldsymbol{\lambda}_k) - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1} - \boldsymbol{I}_p \right) \right\},$$

$$\frac{\partial q_i(\mathbf{\Psi})}{\partial \boldsymbol{\lambda}_k} = -\pi_{ik} \boldsymbol{D}_k \boldsymbol{\Sigma}_k^{-1} \left(\mathcal{M}(\boldsymbol{x}_i; \boldsymbol{\lambda}_k) - \boldsymbol{\mu}_k\right) + \pi_{ik} \boldsymbol{x}_i,$$

where $\boldsymbol{I}_p$ is the identity matrix of size p, vech$\{\cdot\}$ operator extracts the unique elements out of a symmetric $p \times p$ matrix and constructs a vector of length $p(p+1)/2$. $\boldsymbol{G}$ is a matrix with zero's and one's that enables the adoption of unique elements in a symmetric matrix (Melnykov, 2013). vec$\{\cdot\}$ is an operator which lines up all columns of a matrix one by one to form a vector. Finally,

$$\boldsymbol{D}_k = \text{diag}\{(1 + (x_{i1}\lambda_{k1} - 1)e^{\lambda_{k1} x_{i1}})/\lambda_{k1}^2, \ldots, (1 + (x_{ip}\lambda_{kp} - 1)e^{\lambda_{kp} x_{ip}})/\lambda_{kp}^2\}.$$

The estimated covariance matrix can be found as $\boldsymbol{I}_e^{-1}(\hat{\mathbf{\Psi}})$, *i.e.*, by inverting the information matrix. Function `Manly.var()` in the package calculates the covariance matrix based on an estimated model provided by function `Manly.EM()`.

Forward and backward selection

In the Manly mixture model, there are $K \times p$ skewness parameters λ_{kj} corresponding to K components and p variables. Such a mixture is called a full Manly mixture model. Oftentimes, some coordinates are close to being normally distributed and the corresponding skewness parameters are unnecessary. Forward and backward selection procedures are adopted to eliminate such parameters and improve the model efficiency. These algorithms also prevent model-overfitting and conduct diagnostics with fitted skewness parameters. If underlying data groups are normally distributed, the selection procedures produce Gaussian mixture models.

The selection is based on the Bayesian information criterion (BIC) (Schwarz, 1978), which is the most commonly used criterion in finite mixture modeling (McLachlan and Peel, 2000). The smaller BIC is, the better fit provided by a mixture is. The forward selection procedure starts from the Gaussian mixture model and adds one λ_{kj} at a time until no improvement in BIC value can be obtained. The produced model is called Manly forward model (denoted as Manly F in this paper) with the details of the method outlined in Algorithm 1. The backward model selection algorithm given in Algorithm 2 works in the opposite direction. It starts with the full Manly mixture and drops one skewness parameter λ_{kj} at a time until no lower BIC can be reached. The obtained model is called the Manly backward model (Manly B). The selection algorithms are available in **ManlyMix** through

Data: $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_n$
Result: estimated model parameters by Manly forward model
Initialization: Gaussian mixture model
while *the current model* $\boldsymbol{M}_{current}$ *has not reached the full Manly mixture model* **do**
  1. find all zero skewness parameters in the current model $\boldsymbol{M}_{current}$, $\lambda_1, \ldots, \lambda_t$;
  2. construct new models $\boldsymbol{M}_{new,1}, \ldots, \boldsymbol{M}_{new,t}$ to compare with;
  3. $\boldsymbol{M}_{new,j}$ sets the previous nonzero $K \times p - t$ skewness parameters and λ_j to be non-zero;
  4. call function `Manly.EM()` to run the EM algorithm for each new model;
  5. initialize with the parameters of model $\boldsymbol{M}_{current}$ to speed the algorithm;
  if *at least one new model has lower BIC than the original model* $\boldsymbol{M}_{current}$ **then**
    find the smallest BIC among the new models;
    the corresponding new model $\boldsymbol{M}_{new}$is selected and let $\boldsymbol{M}_{current} \leftarrow \boldsymbol{M}_{new}$.
  else
    break;
    the current model $\boldsymbol{M}_{current}$ is the final solution reached by Manly forward algorithm.
  end
end

Algorithm 3: Manly forward selection algorithm.

Data: $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_n$
Result: estimated model parameters by Manly backward model
Initialization: full Manly mixture model $\boldsymbol{M}_{full}$ with $K \times p$ non-zero skewness parameters
while *the current model* $\boldsymbol{M}_{current}$ *has not reached Gaussian mixture model* **do**
  1. find all non-zero skewness parameters in the current model $\boldsymbol{M}_{current}$, $\lambda_1, \ldots, \lambda_s$;
  2. construct new models $\boldsymbol{M}_{new,1}, \ldots, \boldsymbol{M}_{new,s}$ to compare with;
  3. $\boldsymbol{M}_{new,j}$ sets the previous $K \times p - s$ skewness parameters and λ_j to be zero;
  4. call function `Manly.EM()` to run the EM algorithm for each new model;
  5. initialize with the parameters of model $\boldsymbol{M}_{current}$ to speed the algorithm;
  if *at least one new model has lower BIC than the original model* $\boldsymbol{M}_{current}$ **then**
    find the smallest BIC among the new models; the corresponding new model $\boldsymbol{M}_{new}$ is selected and let $\boldsymbol{M}_{current} \leftarrow \boldsymbol{M}_{new}$.
  else
    break;
    the current model $\boldsymbol{M}_{current}$ is the final solution reached by Manly backward algorithm.
  end
end

Algorithm 4: Manly backward selection algorithm.

setting `method = "forward"` or `method = "backward"` in the `Manly.select()` function.

Manly *K*-means clustering

Manly *K*-means clustering is constructed based on the classification EM (CEM) algorithm (Celeux and Govaert, 1992), which is a modification of the EM algorithm with an additional classification step. This step involves the Bayesian decision rule (*i.e.*, $z_i^{(s)} = \arg\max_k \pi_{ik}^{(s)}$) introduced immediately after the E-step.

It can be noticed that the traditional *K*-means algorithm is equivalent to the CEM algorithm based on the mixture model provided by

$$g(\boldsymbol{x}; \boldsymbol{\Psi}) = \frac{1}{K} \sum_{k=1}^{K} \phi(\boldsymbol{x}; \boldsymbol{\mu}_k, \sigma^2 \boldsymbol{I}).$$

The model underlying the traditional *K*-means imposes very restrictive assumptions of the homoscedasticity and spherical structure of components. We alleviate these assumptions by allowing each component to have the covariance matrix $\sigma_k^2 \boldsymbol{I}$ and applying Manly transformation to the data. These changes result in the model given by

$$g(\boldsymbol{x}; \boldsymbol{\Psi}) = \frac{1}{K} \sum_{k=1}^{K} \phi(\mathcal{M}(\boldsymbol{x}; \boldsymbol{\lambda}_k); \boldsymbol{\mu}_k, \sigma_k^2 \boldsymbol{I}) \exp\{\boldsymbol{\lambda}_k^T \boldsymbol{x}\}. \tag{9}$$

Following the same procedure as the Manly mixture EM algorithm, each $\boldsymbol{\lambda}_k$ can be obtained separately by straightforward numeric optimization of the function $\tilde{Q}_k$ written as

$$\tilde{Q}_k(\boldsymbol{\lambda}_k | \boldsymbol{\Psi}^{(s-1)}) = -\frac{p n_k^{(s)}}{2} \log \left\{ \sum_{i=1}^{n} \xi_{ik}^{(s)} \left(n_k^{(s)} \mathcal{M}(\boldsymbol{x}_i; \boldsymbol{\lambda}_k) - \sum_{j=1}^{n} \xi_{jk}^{(s)} \mathcal{M}(\boldsymbol{x}_j; \boldsymbol{\lambda}_k) \right)^T \times \left(n_k^{(s)} \mathcal{M}(\boldsymbol{x}_i; \boldsymbol{\lambda}_k) - \sum_{j=1}^{n} \xi_{jk}^{(s)} \mathcal{M}(\boldsymbol{x}_j; \boldsymbol{\lambda}_k) \right) \right\} + \boldsymbol{\lambda}_k^T \sum_{i=1}^{n} \xi_{ik}^{(s)} \boldsymbol{x}_i + const,$$

where fuzzy classifications $\pi_{ik}^{(s)}$ are replaced by hard assignments in the form of indicators $\xi_{ik}^{(s)} = I(z_i^{(s)} = k)$. If $z_i^{(s)} = k$ holds true, $\xi_{ik}^{(s)}$ takes a value of 1; otherwise $\xi_{ik}^{(s)}$ is equal to 0. The current size of the kth cluster is $n_k^{(s)} = \sum_{i=1}^{n} \xi_{ik}^{(s)}$.

In this way, each step of the Manly *K*-means algorithm updates the partition and parameter

estimates. The partition update is given by

$$z_i^{(s)} = \arg\min_k \left\{ ||\mathcal{M}(\boldsymbol{x}_i; \boldsymbol{\lambda}_k^{(s-1)}) - \boldsymbol{\mu}_k^{(s-1)}||^2 / (2(\sigma_k^2)^{(s-1)}) - (\boldsymbol{\lambda}_k^{(s-1)})^T \boldsymbol{x}_i + \frac{p}{2} \log(\sigma_k^2)^{(s-1)} \right\},$$

while the parameters are estimated through the following expressions:

$$\begin{aligned} \boldsymbol{\lambda}_k^{(s)} &= \arg\max_{\boldsymbol{\lambda}_k} \tilde{Q}_k^{(s)}(\boldsymbol{\lambda}_k), \qquad \boldsymbol{\mu}_k^{(s)} = \sum_{i=1}^n \xi_{ik}^{(s)} \mathcal{M}(\boldsymbol{x}_i; \boldsymbol{\lambda}_k^{(s)}) / n_k^{(s)}, \quad \text{and} \\ (\sigma_k^2)^{(s)} &= \sum_{i=1}^n \xi_{ik}^{(s)} (\mathcal{M}(\boldsymbol{x}_i; \boldsymbol{\lambda}_k^{(s)}) - \boldsymbol{\mu}_k^{(s)})^T (\mathcal{M}(\boldsymbol{x}_i; \boldsymbol{\lambda}_k^{(s)}) - \boldsymbol{\mu}_k^{(s)}) / (p n_k^{(s)}). \end{aligned} \tag{10}$$

The Manly *K*-means algorithm is incorporated in the R package **ManlyMix** through the function `Manly.Kmeans()`. It can be used when the number of data points in each cluster is about the same and the transformed clusters are close to being spherical. It shows faster performance as the inversion of potentially large covariance matrices is not needed.

Package functionality and illustrative examples

All functions available in the package **ManlyMix** are listed with brief descriptions in Table 2. In this section, we demonstrate the utility of each function through a synthetic dataset and the analysis of two real-life datasets: *Iris* (Anderson, 1935; Fisher, 1936) and *AIS* (Cook and Weisberg, 1994).

Function	Description
`Manly.EM()`	Runs the EM algorithm for a Manly mixture model
`Manly.select()`	Runs forward and backward selection methods for a Manly mixture model
`Manly.Kmeans()`	Runs the Manly *K*-means clustering
`Manly.overlap()`	Estimates the overlap values for a Manly mixture
`Manly.sim()`	Simulates datasets from Manly mixture models
`Manly.var()`	Performs variability assessment of Manly mixture model parameter estimates and returns confidence intervals
`Manly.plot()`	Constructs a plot to display model-fitting and clustering
`ClassAgree()`	Calculates the confusion matrix and number of misclassifications
`Manly.model()`	Serves as a wrapper function for Manly mixture modeling

Table 2: Summary of functions implemented in **ManlyMix**.

Illustrative example 1

In this subsection, a Manly mixture is constructed with user-specified parameters. The overlap values of this mixture is estimated through function `Manly.overlap()`. Then function `Manly.sim()` simulates a dataset from the mixture along with a true membership vector.

Step a: Mixture specification

Now we demonstrate the procedure to construct a Manly mixture step by step. First, the user need to specify the number of components (assigned to `K`) and variables (assigned to `p`). In this case, we have a three-component bivariate mixture.

```
library(ManlyMix)
K <- 3
p <- 2
set.seed(123)
```

If the mixture probability density function of interest is written as

$$\begin{aligned} g(\boldsymbol{x}) =& 0.25 e^{0.2x_1 + 0.25x_2} \phi\left(\begin{pmatrix} \frac{e^{0.2x_1}-1}{0.2} \\ \frac{e^{0.25x_2}-1}{0.25} \end{pmatrix}; \begin{pmatrix} 4.5 \\ 7 \end{pmatrix}, \begin{pmatrix} 0.4 & 0 \\ 0 & 0.4 \end{pmatrix} \right) \\ &+ 0.3 e^{0.5x_1 + 0.35x_2} \phi\left(\begin{pmatrix} \frac{e^{0.5x_1}-1}{0.5} \\ \frac{e^{0.35x_2}-1}{0.35} \end{pmatrix}; \begin{pmatrix} 4 \\ 8 \end{pmatrix}, \begin{pmatrix} 1 & -0.2 \\ -0.2 & 0.6 \end{pmatrix} \right) \\ &+ 0.45 e^{0.3x_1 + 0.4x_2} \phi\left(\begin{pmatrix} \frac{0.3e^{x_1}-1}{0.3} \\ \frac{e^{0.4x_2}-1}{0.4} \end{pmatrix}; \begin{pmatrix} 5 \\ 5.5 \end{pmatrix}, \begin{pmatrix} 2 & -1 \\ -1 & 2 \end{pmatrix} \right), \end{aligned} \tag{11}$$

we construct the mixture by assigning the model parameter values to `la` (matrix input of size $K \times p$), `tau` (vector input of length K), `Mu` (matrix input of size $K \times p$) and `S` (array input of dimensionality $p \times p \times K$), respectively.

```
tau <- c(0.25, 0.3, 0.45)
Mu <- matrix(c(4.5, 4, 5, 7, 8, 5.5),3)
la <- matrix(c(0.2, 0.5, 0.3, 0.25, 0.35, 0.4),3)
S <- array(NA, dim = c(p, p, K))
S[,,1] <- matrix(c(0.4, 0, 0, 0.4), 2)
S[,,2] <- matrix(c(1, -0.2, -0.2, 0.6), 2)
S[,,3] <- matrix(c(2, -1, -1, 2), 2)
```

Step b: Overlap assessment

It is desirable to be capable of understanding the degree of interaction among mixing components to assess clustering complexity. Function `Manly.overlap()`, employing the measure of pairwise overlap, is implemented for this purpose. It has the following syntax:

```
Manly.overlap(tau, Mu, S, la, N = 1000)
```

with arguments `la`, `tau`, `Mu`, `S` and `N`. Here, `N` represents the number of samples simulated from the given mixture for pairwise overlap estimation. The larger `N` is, the more precise the calculation is. By default, 1000 samples are employed. Four objects are returned by the function, including the misclassification probability matrix `$OmegaMap`, pairwise overlap `$OverlapMap`, average mixture overlap `$BarOmega`, and maximum mixture overlap `$MaxOmega`. Here, element `$OmegaMap[k2,k1]` corresponds to $\omega_{k_1|k_2}$ in Equation 7. In this case, for example, $\omega_{3|2} = 0.046$ means that a random variable coming from the second component has approximate probability of 0.046 to be misclassified to group 3. $\omega_{3|3} = 0.933$ represents the probability that a point belonging to group 3 is correctly assigned to this group. Each row of `$OmegaMap` sums up to 1. Then, pairwise overlaps ω_{k_1,k_2} given in Equation 6 are provided in the `$OverlapMap`. Among all pairwise overlaps ($\omega_{1,2}$, $\omega_{1,3}$ and $\omega_{2,3}$), $\omega_{2,3}$ yields the maximum value of 0.097 and produces `$MaxOmega`. The average of these three values, on the other hand, results in `$BarOmega` being `0.08066667`.

```
A <- Manly.overlap(tau, Mu, S, la)
print(A)
## $OmegaMap
##       [,1]  [,2]  [,3]
## [1,] 0.909 0.058 0.033
## [2,] 0.038 0.916 0.046
## [3,] 0.016 0.051 0.933
##
## $OverlapMap
##   Components Overlap
## 1     (1, 2)   0.096
## 2     (1, 3)   0.049
## 3     (2, 3)   0.097
##
## $BarOmega
## [1] 0.08066667
##
## $MaxOmega
## [1] 0.097
```

It can be seen that in the considered case, function `Manly.overlap()` calculates all characteristics based on the input of true model parameters. If parameters `la`, `tau`, `Mu` and `S` are estimated, `Manly.overlap()` provides estimates of misclassification probabilities and overlap values. As for high-dimensional data, we can not readily visually assess the interaction between data groups, such output helps approximate the proximity of clusters and discover properties associated with them.

Step c: Data generation

Function `Manly.sim()` simulates Manly mixture datasets based on user-specified model parameters. It employs the built-in R function `rmultinom()` for assigning data points to K mixture components according to the mixing proportion τ_k's. Then the function simulates normally distributed data points by function `rnorm()`. The covariance structures $\mathbf{\Sigma}_k$ are applied to the data points before back-transforming them to Manly distributed components.

The `Manly.sim()` command has the following syntax:

```
Manly.sim(n, la, tau, Mu, S)
```

The user can input `n` as the desired sample size. Here, a dataset of 30 observations is simulated from Equation 11 and data matrix `$X` as well as its true membership vector `$id` are returned.

```
n <- 30
B <- Manly.sim(n, la, tau, Mu, S)
print(B)
## $X
##           [,1]     [,2]
##  [1,] 3.259485 3.882271
##  [2,] 3.247362 4.269247
```

Part of the output is intentionally omitted.

```
## [30,] 3.310186 2.974554
##
## $id
##  [1] 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3
```

Illustrative example 2: *Iris* dataset

The *Iris* dataset (Anderson, 1935; Fisher, 1936) has 150 observations and 4 variables that represent sepal length, sepal width, petal length, and petal width. Three species, *Iris setosa*, *Iris versicolor*, and *Iris virginica*, have equal representation, consisting of 50 observations each. The function `Manly.EM()` fits a Manly mixture to the *Iris* dataset and 95% confidence intervals of the model MLE are provided by `Manly.var()`. The Manly F and Manly B models are obtained by `Manly.select()`. The Manly *K*-means algorithm clusters the dataset through `Manly.Kmeans()`.

Step a: Data preparation

`Manly.EM()` requires input of a matrix object `X`, where rows of `X` represent p-variate observations. If `X` is univariate data with vector input, it will be automatically transformed into a matrix of just one column. Thus, `X` has the dimensionality $n \times p$. In this case, we transform the *Iris* dataset into a matrix of dimensionality 150×4 and assign it to `X`.

```
library(ManlyMix)
K <- 3
p <- 4
X <- as.matrix(iris[,-5])
```

Step b: Initialization of the EM algorithm

Good initialization strategy of the EM algorithm is important to improve chances of finding a correct result. There are two ways for the user to initialize the `Manly.EM()` function. One is by means of providing the initial partition of the data `id` (vector input of length n) and skewness parameters `la`. Here, it needs to be noticed that the specification of `la` matrix serves as an indicator of whether the transformation is applied to a specific variable and component or not. For example, for the *Iris* dataset, assumes that all variables in all components enjoy normality except for the first variable in the first component, `la` needs to be set as `la <-matrix(c(0.1,rep(0,11)),3,4)`, with `0.1` (that can be any non-zero value) serving as the starting point in Nelder-Mead optimization. If no `la` is provided, the skewness parameters are all set equal to 0 and a Gaussian mixture model will be fitted. The other way of starting the algorithm is to enter initial model parameters, including `la`, `tau`, `Mu`, and `S`. The algorithm employs these parameters to compute the posterior probabilities in the first E-step.

Here, we adopt the first strategy. The initial partition of the *Iris* data is obtained by running the traditional *K*-means algorithm and specifying `la` is a matrix of size 3×4, with all elements set to a non-zero value of `0.1`.

```
set.seed(123)
id.km <- kmeans(X, K)$cluster
la <- matrix(0.1, K, p)
```

Step c: EM algorithm for Manly mixture modeling

`Manly.EM()` runs the EM algorithm for modeling based on Manly mixtures given in Equation 1. The command has the following syntax:

```
Manly.EM(X, id = NULL, la = NULL, tau = NULL, Mu = NULL, S = NULL,
         tol = 1e-5, max.iter = 1000).
```

The parameters `tol` and `iter` correspond to the stopping rule for the EM algorithm. `tol` specifies the tolerance level of the EM algorithm. If the relative difference of the Q function values from two consecutive steps is smaller than `tol`, the EM algorithm is terminated. By default, `tol` is set equal to 10^{-5}. `max.iter` stands for the maximum number of iterations allowed for the EM algorithm. The default value of `max.iter` is `1000`. We fit the *Iris* dataset by both Gaussian mixture (assigned to object `G`) and Manly mixture (assigned to object `M`).

```
G <- Manly.EM(X, id = id.km)
colnames(G$la) <- colnames(X)
print(G$la)
##      Sepal.Length Sepal.Width Petal.Length Petal.Width
## [1,]            0           0            0           0
## [2,]            0           0            0           0
## [3,]            0           0            0           0

M <- Manly.EM(X, id.km, la)
colnames(M$la) <- colnames(X)
print(M$la)
##      Sepal.Length Sepal.Width Petal.Length Petal.Width
## [1,]   -0.1158602  0.05907443   -0.2382086   -4.033529
## [2,]   -0.1254022 -0.65079974   -0.3848938    0.479587
## [3,]   -0.1282339  0.64271380    0.3343054   -1.134275

print(M$id)
## [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
## [75] 3 3 3 3 3 3 3 3 3 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2 2 2 2 2 2 2 2 2 2 3
## [112] 2 2 2 2 2 2 2 2 3 2 2 2 3 2 2 3 3 2 2 2 2 2 3 2 2 2 2 3 3 2 3 2 2 2 3 3 3
## [149] 2 2
```

The estimated model parameters returned by the function `Manly.EM()` include `$la` (matrix output of size $K \times p$), `$tau` (vector output of length K), `$Mu` (matrix output of size $K \times p$) and `$S` (array output of dimensionality $p \times p \times K$). They correspond to the parameters $\boldsymbol{\lambda}_k$, τ_k, $\boldsymbol{\mu}_k$ and $\boldsymbol{\Sigma}_k$ in Equation 3 and 4, respectively. In this example, it can be observed that the returned `$la` for the Gaussian mixture have all elements fixed at zero, while the Manly mixture has estimated the skewness parameters for each component and variable. For example, the skewness parameter associated with the sepal length variable of the first cluster is estimated to be -0.1158602. It can be observed that most of the estimated parameters are relatively close to zero, which indicates approximate normality of the *Iris* data.

Some other parameters returned by `Manly.EM()` are the $n \times K$ matrix of posterior probabilities `$gamma` calculated from Equation 2 in the last E-step and the membership vector `$id` assigned by the Bayes decision rule in Equation 5. In this case, the output of `$id` demonstrates the model-based clustering solution of the *Iris* dataset.

The characteristics of the fitted model are demonstrated in terms of the model log-likelihood `$ll` and BIC `$bic`. The number of iterations run by the EM algorithm until convergence is recorded through `$iter`. In this example, the EM algorithm reaches convergence after 13 iterations and the model BIC is 618.46. Finally, a dummy indicator `$flag` reports the validity of the fitted model, where 0 represents the successful convergence of the EM algorithm and 1 stands for the failure of convergence. A warning message is given if `$flag` is equal to 1. It may happen when one cluster disappears or shrinks so that some parameter estimates are NA's. Such issue is related to spurious solutions (McLachlan and Peel, 2000) where one or more components model a local pattern in data rather than a systematic one.

Step d: Variability assessment of Manly mixture model

Variability assessment of the model parameters allows practitioners to study the specific nature of the fitted model as well as detected clustering solutions. We provide the user with function `Manly.var()`, which calculates the inverse of the empirical observed information matrix given in Equation 8 and returns the variance-covariance matrix of the estimated MLE from `Manly.EM()` function. It also outputs the confidence intervals of each parameter.

The command has the following syntax:

```
Manly.var(X, model = NULL, conf.CI = NULL)
```

`X` represents the data matrix and `model` is the object of class `"ManlyMix"`. `conf.CI` is user-specified confidence level, which needs to take a value between `0` and `1`. Here by setting `model = M`, we take the MLE of the fitted Manly mixture obtained from step c and evaluate its variability. The number of unique model parameters is $K - 1 + 2K \times p + K \times p(p+1)/2 = 56$ for the *Iris* dataset. `conf.CI =`

`0.95` calculates 95% confidence intervals for these 56 parameters. Thus `Manly.var()` function returns a 56×56 covariance matrix (assigned to `V`) and 56 confidence intervals (assigned to `CI`).

```
result <- Manly.var(X, model = M, conf.CI = 0.95)
```

In the code output of 95% confidence intervals, the first column represents the point estimates of the 56 model parameters, while the second and third columns stand for the lower and upper bounds of confidence intervals, respectively.

```
print(result$CI)
##          Estimates         Lower        Upper
##  [1,]  0.333333333   0.257887628  0.408779039
##  [2,]  0.264119102   0.175676489  0.352561716
##  [3,]  3.794594378  -8.303528084 15.892716840
```

Part of the output is intentionally omitted.

```
## [54,]  0.642713799  -0.407079526  1.692507125
## [55,]  0.334305435  -0.128841410  0.797452280
## [56,] -1.134275340  -2.079185136 -0.189365544
```

Step e: Forward and backward selection algorithms

Step e targets detecting the normally distributed variables in *Iris*. `Manly.select()` provides the selection algorithm for eliminating unnecessary skewness parameters in `M$la`. These skewness parameters are fixed to be equal to zero and the log-likelihood is maximized based on the rest of parameters. The use of the function is shown below:

```
Manly.select(X, model, method, tol = 1e-5, max.iter = 1000,
             silent = FALSE)
```

The argument `model` is the initial model to start the selection procedure with. `method` is set to either `"forward"` or `"backward"` for the implementation of Algorithm 1 or Algorithm 2, respectively. The selection criterion for each step is based on `$bic` values obtained from all candidate models that are of class `"ManlyMix"`. `silent` is an argument that controls the code output. By default, `silent` provides the steps of selection and BIC values for all candidate models. Thus, the user can monitor the selection procedures. The output can be turned off by setting `silent = TRUE`. We first discuss the implementation of the forward selection on the *Iris* dataset. The algorithm is initialized by the Gaussian mixture model `G` obtained in step c.

```
MF <- Manly.select(X, model = G, method = "forward")
## step 1 :
##         current BIC = 580.8389
##         alternative BICs = 585.6791 585.0607 582.1893 585.7369 585.2193 583.7374
##         585.7081 583.963 579.8978 573.4626 585.8161 585.8407
## step 2 :
##         current BIC = 573.4626
##         alternative BICs = 578.3191 577.6844 574.813 578.3719 577.843 576.3611
##         578.3282 576.5866 572.5215 578.4397 578.4643
## step 3 :
##         current BIC = 572.5215
##         alternative BICs = 577.378 576.7713 575.8067 577.4308 576.8833 575.3526
##         577.3871 575.6213 577.4799 577.3221
```

The forward selection takes three steps for the algorithm to find the best model (assigned to `MF`). In step 3, there is no alternative BIC value that is smaller than the current model BIC, so the forward selection algorithm stops searching over non-zero λ_{kj}'s. Compared to the Gaussian mixture fit, Manly F model improves by 8 in BIC value.

On the contrary, the backward selection starts with the full Manly mixture `M` and drops one skewness parameter at a time.

```
MB <- Manly.select(X, model = M, method = "backward")
## step 1 :
##         current BIC = 618.4553
##         alternative BICs = 613.5161 612.7184 613.8658 613.448 614.3828 616.6445
##         613.5442 616.3626 617.0157 625.7431 613.1927 610.7879
## step 2 :
```

```
##         current BIC = 610.7879
##         alternative BICs = 605.9157 605.9075 607.3512 605.8475 606.3851 607.8112
##         605.9437 607.0513 609.9457 618.1426 605.7915
```

Part of the output is intentionally omitted.

```
## step 10 :
##         current BIC = 575.3526
##         alternative BICs = 572.5215 576.3611 582.729
## step 11 :
##         current BIC = 572.5215
##         alternative BICs = 573.4626 579.8978
```

After 11 steps, the backward selection produces the Manly B model, which enjoys the same BIC value as the Manly F model.

Step f: Diagnostics

The skewness parameters of the Manly F and Manly B models are investigated in the following example. It is observed that the forward selection adopts only two λ_{kj}'s in the model. They correspond to the petal width variable of the first species and the petal length variable of the third one. For all other components and variables, the data appear to be nearly normally distributed. The same two skewness parameters are found by the backward selection. It is worth mentioning, however, that Manly F and Manly B models can produce different results.

```
colnames(MF$la) <- colnames(X)
print(MF$la)
##     Sepal.Length Sepal.Width Petal.Length Petal.Width
## [1,]            0           0    0.0000000       -4.04
## [2,]            0           0    0.0000000        0.00
## [3,]            0           0    0.5615625        0.00
colnames(MB$la) <- colnames(X)
print(MB$la)
##      Sepal.Length Sepal.Width Petal.Length Petal.Width
## [1,]            0           0    0.0000000   -4.034815
## [2,]            0           0    0.0000000    0.000000
## [3,]            0           0    0.5619671    0.000000
```

Step g: Manly K-means algorithm

The Manly K-means algorithm written in Equation 9 is implemented in function `Manly.Kmeans()`, which has the following syntax:

```
Manly.Kmeans(X, id = NULL, la = NULL, Mu = NULL, S = NULL,
             initial = "k-means", K = NULL, nstart = 100,
             method = "ward.D", tol = 1e-5, max.iter = 1000).
```

`Manly.Kmeans()` has most of the arguments and returned values the same as those of the function `Manly.EM()`. As the Manly K-means algorithm assumes that all clusters are of the same size, the mixing proportions `tau` are not needed in this function. `S` is a vector of length K that represents variance within each cluster, as the transformed data groups are assumed to be spherical. The parameters returned by the function `$la`, `$Mu`, and `$S` correspond to $\boldsymbol{\lambda}_k$, $\boldsymbol{\mu}_k$ and σ_k^2 given in Equation 10. The log-likelihood and BIC values are not provided since the parameter estimates are not MLE's. `Manly.Kmeans()` has several initialization choices: (1) by providing `id` and `la`; (2) by providing `la`, `Mu`, and `S`; (3) by specifying the number of clusters `K` and letting `initial = "k-means"`; It takes the default traditional K-means clustering result and passes it into the CEM algorithm; (4) by specifying the number of clusters `K` and letting `initial = "hierarchical"`; It adopts the hierarchical clustering solution as the initial dataset partition. `nstart` is responsible for controlling the number of random starts tried in initialization choice (3) with a default value equal to `100`. `method` sets the linkage method in initialization choice (4) with a default of `method = "ward.D"`, which represents the Ward's linkage (Ward, 1963). Here, the initialization choice of `Manly.Kmeans()` is (1), which is the same as that of `Manly.EM()`.

```
MK <- Manly.Kmeans(X, id.km, la)
colnames(MK$la) <- colnames(X)
print(MK$la)
##      Sepal.Length Sepal.Width Petal.Length Petal.Width
## [1,]  -0.37975529  -0.5815382  -0.81530022  -2.5572830
```

```
## [2,]  -0.27067058  -0.4103692  -0.31001602  -0.5367999
## [3,]  -0.02896526   0.1138177  -0.05694487   0.2617650

print(MK$S)
## [1] 0.002717844 0.006156015 0.160435910
```

Illustrative example 3: *AIS* dataset

In this subsection, dataset *AIS* (Cook and Weisberg, 1994) is studied for illustrative purposes. The Australian Institute of Sports (AIS) dataset was first introduced by Cook and Weisberg (1994). It contains information collected from 202 athletes, among which 100 are females and 102 are males. There are 13 variables, including the gender, sport kind and 11 numeric measurements of the athletes. We adopt the same variables and analysis as Lee and McLachlan (2013). The goal of the analysis is to cluster the athletes into two groups: males and females by constructing models based on three measurements: the body mass index ("BMI"), lean body mass ("LBM"), and the percentage of body fat ("Bfat"). Function `ClassAgree()` compares the estimated and true partitions. Function `Manly.plot()` is introduced for visual analysis of Manly mixture fitted results.

Step a: model fit

The *AIS* dataset is analyzed by six mixture models: the traditional *K*-means (`kmeans()`), Manly *K*-means (`Manly.Kmeans()`), Gaussian mixture model (`Manly.EM()`), Manly mixture model (`Manly.EM()`), Manly forward model and Manly backward model (both available through `Manly.select()`).

```
library(ManlyMix)
data("ais"); set.seed(123)
X <- as.matrix(ais[,c(8, 10, 11)])
id <- as.numeric(ais[,1])
n <- dim(X)[1]
p <- dim(X)[2]
K <- max(id)
Kmeans <- kmeans(X, K)
id.km <- Kmeans$cluster
```

By running the following code, we not only obtain the fitted models, but also test the package from different aspects. The number of parameters in the models are 7 (*K*-means), 19 (Gaussian), 25 (Manly), 23 (Manly F), 22 (Manly B) and 14 (Manly *K*-means). The computing times are 0.001, 0.004, 0.083, 0.8, 1.143, 0.024, respectively. These results are rather efficient compared to those from other packages (see Appendix).

```
MK <- Manly.Kmeans(X, id = id.km, la = matrix(0.1, K, p))
G <- Manly.EM(X, id = id.km, la = matrix(0, K, p))
M <- Manly.EM(X, id = id.km, la = matrix(0.1, K, p))
MF <- Manly.select(X, G, method = "forward", silent = TRUE)
MB <- Manly.select(X, M, method = "backward", silent = TRUE)
```

Now we consider the fitted model parameters to perform a comprehensive analysis and diagnostics of the *AIS* dataset. From the following output, it is observed that the Manly F model drops two skewness parameters from the full Manly mixture model while Manly B drops three. This yields the conclusion that the "Bfat" variable in the first group and "LBM" variable in the second one are close to be normal. Through the one-to-one correspondence between skewness parameters and dataset variables, **ManlyMix** is proved to be particularly useful for model variable diagnostics.

```
colnames(MF$la) <- colnames(X)
MF$la
##               BMI       Bfat        LBM
## [1,] -0.08671894  0.0000000 0.01002851
## [2,] -0.12882354 -0.1902031 0.00000000

colnames(MB$la) <- colnames(X)
MB$la
##               BMI       Bfat LBM
## [1,] -0.09362427  0.0000000   0
## [2,] -0.12720459 -0.1933216   0
```

BIC values for the four models are 3595.35 (Gaussian), 3543.00 (Manly), 3538.42 (Manly F) and 3533.63 (Manly B). It shows considerable improvement in terms of BIC from Manly mixture models.

They provide better fits for the data, among which Manly backward is the best model selected according to BIC.

Step b: classification table

Classification results from the six models are compared using function `ClassAgree()` in step b. Function `ClassAgree()` adopts input of both the estimated and true id vectors with the following syntax:

```
ClassAgree(est.id, trueid)
```

`ClassAgree()` permutes the partition labels to achieve the lowest number of misclassifications. Then, based on the switched labels, it returns the confusion matrix and number of misclassifications. In the analysis of the *AIS* dataset, the following output is produced by `ClassAgree()`.

```
ClassAgree(id.km, id)
## $ClassificationTable
##       est.id
## trueid  1  2
##      1 98  2
##      2 12 90
##
## $MisclassificationNum
## [1] 14

ClassAgree(MK$id, id)
## $ClassificationTable
##       est.id
## trueid  1  2
##      1 95  5
##      2  7 95
##
## $MisclassificationNum
## [1] 12

ClassAgree(G$id, id)
## $ClassificationTable
##       est.id
## trueid   1   2
##      1 100   0
##      2   8  94
##
## $MisclassificationNum
## [1] 8

ClassAgree(M$id, id)
## $ClassificationTable
##       est.id
## trueid   1   2
##      1  98   2
##      2   2 100
##
## $MisclassificationNum
## [1] 4

ClassAgree(MF$id, id)
## $ClassificationTable
##       est.id
## trueid  1  2
##      1 99  1
##      2  3 99
##
## $MisclassificationNum
## [1] 4

ClassAgree(MB$id, id)
## $ClassificationTable
```

```
##       est.id
## trueid  1  2
##      1 99  1
##      2  4 98
##
## $MisclassificationNum
## [1] 5
```

Rows and columns represent the true and estimated partitions, respectively. The diagonal and off-diagonal elements in the table correspond to correct and incorrect classifications, respectively. The lowest number of misclassifications (4 misclassifications) is obtained by the Manly mixture and Manly forward models. One worth-mentioning fact is that these two models enjoy the clustering solution as good as the unrestricted skew-*t* mixture, which is reported to be the best model by Lee and McLachlan (2013). The Manly backward model comes second with 5 misclassifications. The remaining three models, traditional *K*-means, Manly *K*-means and Gaussian mixture model show worse performance.

Step c: visualization tool

In order to investigate the behavior of each model, contour plots with classified data points need to be analyzed. `Manly.plot()` allows conducting the visual analysis of a dataset fitted by Manly mixture model. The command has the following syntax:

```
Manly.plot(X, var1 = NULL, var2 = NULL, model = NULL, x.slice = 100,
           y.slice = 100, x.mar = 1, y.mar = 1, col = "lightgrey", ...).
```

If both `var1` and `var2` are provided, they represent variables on the *X*-axis and *Y*-axis of a contour plot, respectively. Argument `model` is the object of class `"ManlyMix"`. The parameters of `model` object are used to calculate the density and draw contour lines. The estimated membership vector `model$id` is reflected through different colors. `x.slice` and `y.slice` options control the number of grid points for which a density is calculated. The larger these two values are, the more grid values are considered. Thus, the contour lines look smoother. `x.mar` and `y.mar` specify plot margins. The parameter `col` specifies the color of contour lines with the default color being light grey. Other variables in the built-in R function `contour()` can also be used as specified. On the other hand, if only `var1` is provided, a density plot of this variable is constructed. `x.slice` and `x.mar` have the same functionality as those in the contour plot. The parameter `col` stands for density line color with the default being light grey. `...` allows other arguments from the built-in R function `hist()` to be passed.

In this case, we conduct the same analysis as that in Lee and McLachlan (2013) and adopt the two variables "LBM" and "Bfat" for constructing contour plots. The margins of the plots are set to be `3` on the *X*-axis and `13` on the *Y*-axis. The light grey contour lines have width equal to `3.2`. Labels and axes are suppressed. The function is first applied to the four fitted models Gaussian mixture, Manly mixture, Manly F and Manly B in step a.

```
Manly.plot(X, var1 = 3, var2 = 2, model = G, x.mar = 3, y.mar = 13,
           xaxs="i", yaxs="i", xaxt="n", yaxt="n", xlab="", ylab = "",
           nlevels = 10, drawlabels = FALSE, lwd = 3.2,
           col = "lightgrey", pch = 19)
Manly.plot(X, var1 = 3, var2 = 2, model = M, x.mar = 3, y.mar = 13,
           xaxs="i", yaxs="i", xaxt="n", yaxt="n", xlab="", ylab = "",
           nlevels = 10, drawlabels = FALSE, lwd = 3.2,
           col = "lightgrey", pch = 19)
Manly.plot(X, var1 = 3, var2 = 2, model = MF, x.mar = 3, y.mar = 13,
           xaxs="i", yaxs="i", xaxt="n", yaxt="n", xlab="", ylab = "",
           nlevels = 10, drawlabels = FALSE, lwd = 3.2,
           col = "lightgrey", pch = 19)
Manly.plot(X, var1 = 3, var2 = 2, model = MB, x.mar = 3, y.mar = 13,
           xaxs="i", yaxs="i", xaxt="n", yaxt="n", xlab="", ylab = "",
           nlevels = 10, drawlabels = FALSE, lwd = 3.2,
           col = "lightgrey", pch = 19)
```

Function `Manly.plot()` enjoys sufficient flexibility to adopt other parsimonious models. Parameters obtained by traditional *K*-means and Manly *K*-means can be adjusted according to the object of class `"ManlyMix"` so that `$id`, `$tau`, `$Mu`, `$la`, `$S` are extracted in their correct forms.

```
Kmeans$id <- id.km
Kmeans$tau <- MK$tau <- rep(1 / K, K)
Kmeans$Mu <- Kmeans$centers
Kmeans$la <- matrix(0, K, p)
```

```
Kmeans$S <- array(0, dim = c(p, p, K))
for(k in 1:K)
  diag(Kmeans$S[,,k]) <- Kmeans$tot.withinss / n / p
s2 <- MK$S
MK$S <- array(0, dim = c(p, p, K))
for(k in 1:K)
  diag(MK$S[,,k]) <- s2[k]
Manly.plot(X, var1 = 3, var2 = 2, model = Kmeans, x.mar = 3, y.mar = 13,
           xaxs="i", yaxs="i", xaxt="n", yaxt="n", xlab="", ylab = "",
           nlevels = 4, drawlabels = FALSE, lwd = 3.2,
           col = "lightgrey", pch = 19)
Manly.plot(X, var1 = 3, var2 = 2, model = MK, x.mar = 3, y.mar = 13,
           xaxs="i", yaxs="i", xaxt="n", yaxt="n", xlab="", ylab = "",
           nlevels = 10, drawlabels = FALSE, lwd = 3.2,
           col = "lightgrey", pch = 19)
```

Figure 1 combines all output plots from `Manly.plot()`.

It can be observed that the two components have a slight overlap, so the clustering problem is not over-complicated. However, the red cluster is highly skewed and has a heavy tail. This imposes difficulties for the traditional *K*-means, Manly *K*-means, and Gaussian mixture model. Manly mixture model shows great flexibility and captures the skewness pattern in both components. Manly forward drops the skewness parameters associated with variable "Bfat" in the female cluster (black component) and "LBM" in the male group (red component). Manly backward drops both skewness parameters that correspond to the female cluster and uses a black ellipsoid. It also drops the "LBM" variable in the male cluster (red component). The above results reveal the applicability and effectiveness of function `Manly.plot()` on real-life datasets.

Alternative coding d: wrapper function

Wrapper function `Manly.model()` enables practitioners to run analysis in a simple and convenient way. The function has the following syntax:

```
Manly.model(X, K = 1:5, Gaussian = FALSE, initial = "k-means",
            nstart = 100, method = "ward.D",  short.iter = 5,
            select = "none", silent = TRUE, plot = FALSE,
            var1 = NULL, var2 = NULL, VarAssess = FALSE,
            conf.CI = NULL, overlap = FALSE, N = 1000, tol = 1e-5,
            max.iter = 1000, ...).
```

Argument `K` is an integer vector providing the numbers of clusters to be tested for the data. The default setting tests 1, 2, 3, 4, or 5 clusters. It calls the `Manly.EM()` function to fit all five models. The one with the lowest BIC value is chosen to be the best model. `Gaussian` option specifies whether skewness parameters are adopted or not. If `TRUE`, Gaussian mixtures are fitted. With the default value being `FALSE`, it runs full Manly mixture models. `initial` specifies the initialization strategy used. It has three input options: (1) `initial = "k-means"` is the default initialization strategy, which passes the traditional *K*-means result into the EM algorithm as the initial partition. `nstart` is passed into the built-in R function `kmeans` for specifying the number of random starts (the default `nstart = 100`); (2) if `initial = "hierarchical"`, the hierarchical clustering initialization is used. The linkage method is passed by `method` argument into R function `hclust`. The default is Ward's linkage; (3) if `initial = "emEM"`, the emEM (Biernacki et al., 2003) initialization is run. Short runs of EM are conducted based on random starts and the one that corresponds to the highest log-likelihood is picked for running until convergence. `nstart` controls the number of random starts. The number of iterations for the short EM is specified by `short.iter` with a default value set to `5` iterations.

`select` argument has three input values: `"none"`, `"forward"` and `"backward"`. If `select = "none"`, then the object returned by function `Manly.EM()` is adopted directly. If `select = "forward"`, the `Gaussian` option is automatically adjusted to `Gaussian = TRUE`. It calls function `Manly.select(...,method = "forward")` to improve the original Gaussian fit. On the other hand, if `select = "backward"`, `Gaussian` option is automatically set to `Gaussian = FALSE`. The full Manly mixture is followed by the backward selection `Manly.select(...,method = "backward")`. `silent` argument controls the output in function `Manly.select()`. The default setting suppresses the output. `plot` determines whether `Manly.plot()` function is called or not. If `plot = TRUE`, then `Manly.plot()` runs and arguments `var1` and `var2` allow user to specify which variable(s) to plot. Argument `VarAssess` provides the option of using `Manly.var()` for variability assessment. Notice here that it only provides assessment for a full Manly mixture model. `conf.CI` specifies the confidence level of the confidence intervals returned. The `overlap` option, if specified to be `TRUE`, adopts the `Manly.overlap()` function and estimates pairwise overlap values for the returned model. `N` is the number of Monte Carlo simulations run in

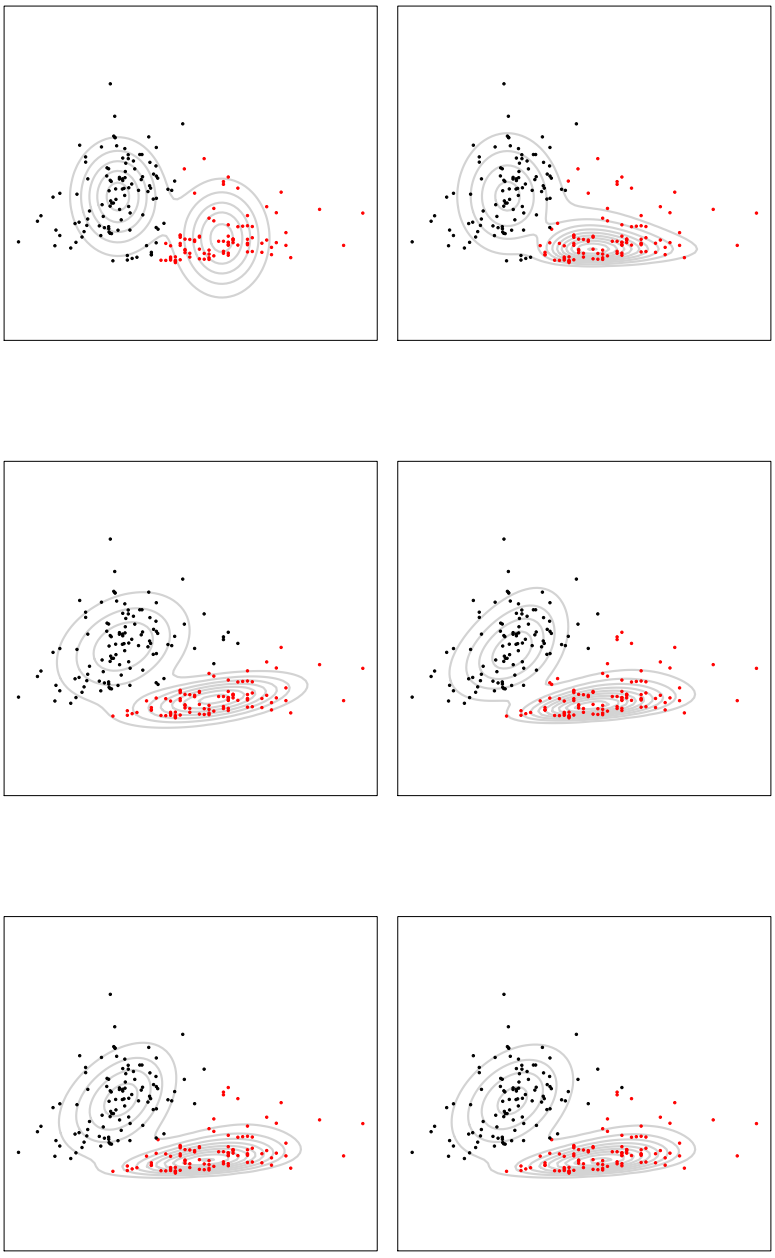


Figure 1: *AIS* dataset: fitted contour plots from function `Manly.plot()` based on the two variables "LBM" (X-axis) and "Bfat" (Y-axis). The model locations are: *K*-means (first row left), Manly *K*-means (first row right), Gaussian mixture (second row left), Manly mixture (second row right), Manly forward (third row left) and Manly backward (third row right).

`Manly.overlap()`.

Three objects are returned by function `Manly.model()`: `$model`, `$VarAssess`, and `$Overlap`. `$model` is the final model of class "ManlyMix" by `Manly.EM()` or `Manly.select()`. `$VarAssess` returns the variance-covariance matrix and confidence intervals by `Manly.var()` function. `$Overlap` returns the object by `Manly.overlap()`.

For *AIS* dataset, suppose the user wants to obtain the Manly F or Manly B model and take a look at their contour plots. A compact version of the code is given by:

```
MF <- Manly.model(X, K = 2, initial = "k-means", nstart = 100,
                  select = "forward", plot = TRUE, var1 = 3, var2 = 2,
                  x.mar = 3, y.mar = 13, xaxs="i", yaxs="i",
                  xaxt="n", yaxt="n", xlab="", ylab = "",
                  nlevels = 4, drawlabels = FALSE, lwd = 3.2,
                  col = "lightgrey", pch = 19)
MB <- Manly.model(X, K = 2, initial = "k-means", nstart = 100,
                  select = "backward", plot = TRUE, var1 = 3, var2 = 2,
                  x.mar = 3, y.mar = 13, xaxs="i", yaxs="i",
                  xaxt="n", yaxt="n", xlab="", ylab = "",
                  nlevels = 4, drawlabels = FALSE, lwd = 3.2,
                  col = "lightgrey", pch = 19)
```

Here, `MF$model` and `MB$model` obtained are the same as those in step a. The contour plots are generated automatically and can be found in Figure 1.

Alternative coding e: initialization with model parameters

Functions `Manly.EM()` can take initial model parameters as initialization of the algorithm. It is especially useful for the emEM initialization. The practitioner can construct a large number of short EM runs, select the one with the highest log-likelihood and obtain its estimated parameters. Then, the EM algorithm initialized by these parameters is run until convergence. Here is a small example on the *AIS* dataset. 100 short EM algorithms run for 5 iterations each. As we can see, the obtained object M is the same as that from step a.

```
ll <- -Inf
init <- NULL
nstart <- 100
iter <- 0
repeat {
          id.km <- kmeans(X, centers = K, iter.max = 1)$cluster
          temp <- Manly.EM(X, id = id.km, la = matrix(0.1, K, p), max.iter = 5)
          if(temp$ll > ll) {
                  ll <- temp$ll
                  init <- temp
          }
          iter <- iter + 1
          if(iter == nstart)
                break
}
M <- Manly.EM(X, tau = init$tau, Mu = init$Mu, S = init$S, la = init$la)
```

Illustrative example 4: *acidity* dataset

Since one reviewer is interested in seeing a showcase of a univariate Manly mixture, we illustrate its utility on the *acidity* dataset (Crawford, 1994). It provides the acidity measure of 155 lakes in the Northeastern United States. There are two clusters, but the true partition is unknown.

Step a: model fit

We run the following models: the traditional K-means (`kmeans()`), Manly K-means (`Manly.Kmeans()`), Gaussian mixture model (`Manly.EM()`), Manly mixture model (`Manly.EM()`), Manly forward model, and Manly backward model (both available through `Manly.select()`).

```
library(ManlyMix)
data("acidity"); set.seed(123)
K <- 2
p <- 1
X <- acidity
```

```
Kmeans <- kmeans(X, K)
id.km <- Kmeans$cluster
MK <- Manly.Kmeans(X, id = id.km, la = matrix(0.1, K, p))
G <- Manly.EM(X, id = id.km, la = matrix(0, K, p))
M <- Manly.EM(X, id = id.km, la = matrix(0.1, K, p))
MF <- Manly.select(X, G, method = "forward", silent = TRUE)
MB <- Manly.select(X, M, method = "backward", silent = TRUE)
```

The model BIC values for Gaussian, Manly, Manly F and Manly B are 394.51, 389.84, 389.84 and 389.84, respectively. There is an indication of skewness as both the Manly F and Manly B models fail to drop any skewness parameters. The Manly models improve by 5 in the BIC value.

Step b: visualization tool

To visually assess the fit provided by all models, we use the command `Manly.plot()` with univariate input. The fitted density plots associated with histogram of the data are provided in Figure 2.

```
Kmeans$id <- id.km
Kmeans$tau <- MK$tau <- rep(1 / K, K)
Kmeans$Mu <- Kmeans$centers
Kmeans$la <- matrix(0, K, p)
Kmeans$S <- array(0, dim = c(p, p, K))
for(k in 1:K)
    Kmeans$S[,,k] <- Kmeans$tot.withinss / n / p
s2 <- MK$S
MK$S <- array(0, dim = c(p, p, K))
for(k in 1:K)
    MK$S[,,k] <- s2[k]
Manly.plot(X = acidity, model = Kmeans, var1 = 1, main = "",
           ylim = c(0, 0.75), xlab = "", xaxt = "n", ylab = "",
           yaxt = "n", x.slice = 200, col = "red")
Manly.plot(X = acidity, model = MK, var1 = 1, main = "", ylim = c(0, 0.75),
           xlab = "", xaxt = "n", ylab = "", yaxt = "n",
           x.slice = 200, col = "red")
Manly.plot(X = acidity, model = G, var1 = 1, main = "", ylim = c(0, 0.75),
           xlab = "", xaxt = "n", ylab = "", yaxt = "n",
           x.slice = 200, col = "red")
Manly.plot(X = acidity, model = M, var1 = 1, main = "", ylim = c(0, 0.75),
           xlab = "", xaxt = "n", ylab = "", yaxt = "n",
           x.slice = 200, col = "red")
Manly.plot(X = acidity, model = MF, var1 = 1, main = "", ylim = c(0, 0.75),
           xlab = "", xaxt = "n", ylab = "", yaxt = "n",
           x.slice = 200, col = "red")
Manly.plot(X = acidity, model = MB, var1 = 1, main = "", ylim = c(0, 0.75),
           xlab = "", xaxt = "n", ylab = "", yaxt = "n",
           x.slice = 200, col = "red")
```

Manly models provide the most reasonable fit of the data. The first component is slightly skewed to the right and only the Manly models pick up the high density at its peak. The second component is slightly skewed to the left. The density fits provided by *K*-means and Manly *K*-means are insufficient due to the assumption of equal size components.

Alternative coding c: wrapper function

The wrapper function `Manly.model()` is capable of combining steps a and b in one command. The following code directly yields the Manly F or Manly B model:

```
MF <- Manly.model(X, K = 2, Gaussian = TRUE, initial = "k-means",
                  nstart = 100, select = "forward", plot = TRUE,
                  var1 = 1, main = "", ylim = c(0, 0.75),
                  xlab = "", xaxt = "n", ylab = "", yaxt = "n",
                  x.slice = 200, col = "red")
MB <- Manly.model(X, K = 2, Gaussian = FALSE, initial = "k-means",
                  nstart = 100, select = "backward", plot = TRUE,
                  var1 = 1, main = "", ylim = c(0, 0.75),
                  xlab = "", xaxt = "n", ylab = "", yaxt = "n",
                  x.slice = 200, col = "red")
```

The Manly F and Manly B density plots given in Figure 2 are generated automatically.

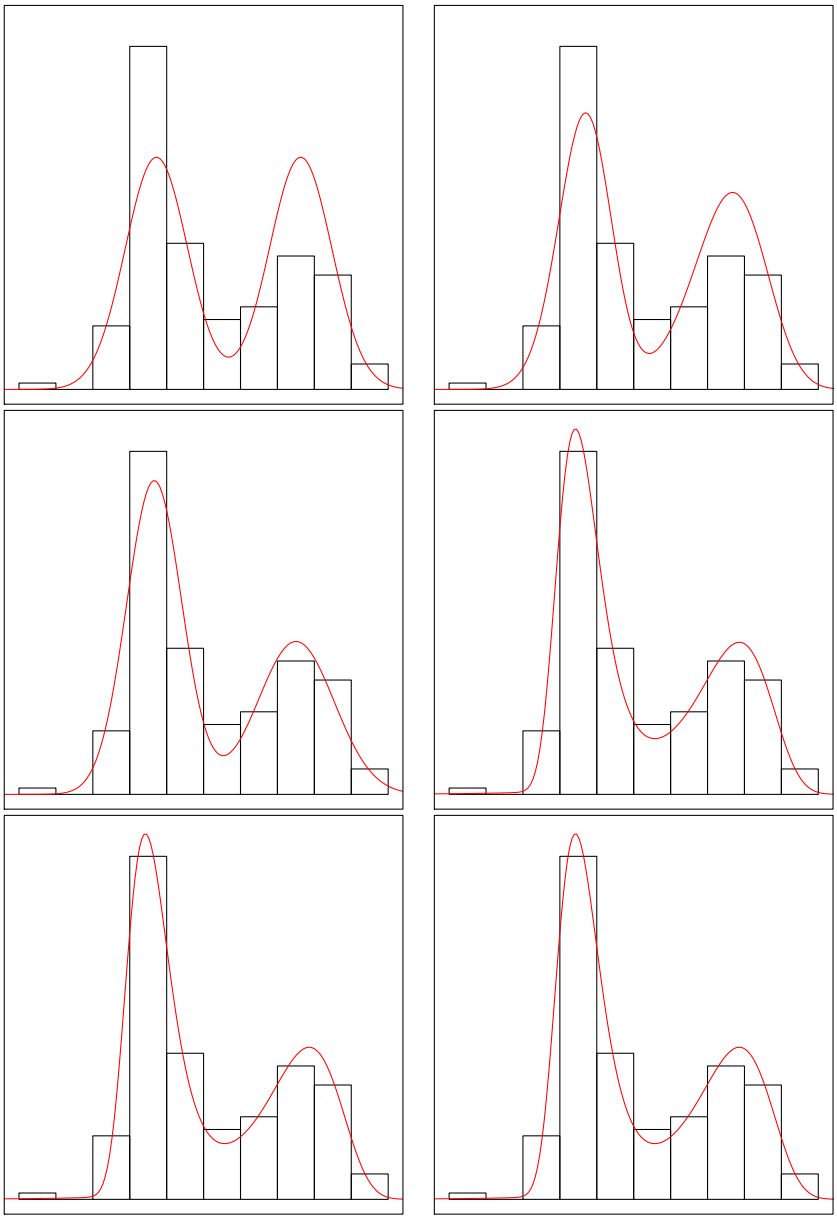


Figure 2: *acidity* dataset: fitted density plots from function `Manly.plot()`: *K*-means (first row left), Manly *K*-means (first row right), Gaussian mixture (second row left), Manly mixture (second row right), Manly forward (third row left) and Manly backward (third row right).

Demo examples

For users who need further information about the package, we have constructed 16 demo examples listed in Table 3 that provide a comprehensive demonstration of **ManlyMix** capabilities. Among the examples, 11 of them are designed to demonstrate the capability and utility of each function and 5 of them run comprehensive analysis of classification datasets. Each demo can be accessed by its name and the users can reproduce themselves.

As an illustration of how these demos can be employed, the code of the first example can be approached through running the following code in R.

```
library(ManlyMix)
demo(EMalgorithm1)
```

Function	Demo example(s)
`Manly.EM()`	`demo(EMalgorithm1)`, `demo(EMalgorithm2)`
`Manly.select()`	`demo(ForwardSelection)`, `demo(BackwardSelection)`
`Manly.Kmeans()`	`demo(ManlyKmeans1)`, `demo(ManlyKmeans2)`
`Manly.overlap()`	`demo(Overlap)`
`Manly.sim()`	`demo(DataSimulation)`
`Manly.var()`	`demo(VarAssess)`
`Manly.plot()`	`demo(DensityPlot)`, `demo(ContourPlot)`
Comprehensive analysis	`demo(utility)`, `demo(ais)`, `demo(seeds)`, `demo(bankruptcy)` `demo(acidity)`

Table 3: Summary of demo examples included in **ManlyMix**.

Summary

The R package **ManlyMix** is discussed and illustrated in detail. The provided functions enable practitioners to analyze heterogeneous data and conduct cluster analysis with Manly mixture models. The algorithms behind functions are introduced and explained carefully. Illustrative examples based on challenging real-life datasets are studied to demonstrate the usefulness and efficiency of the package. Promising results suggest that **ManlyMix** is not only a powerful package for clustering and classification, but also a diagnostic tool to investigate skewness and deviation from normality in data. Demo examples are provided for each function in **ManlyMix** for the users to study.

Appendix

The six competitors for mixture modeling of skewed data given in Table 1 are applied to the *AIS* dataset in Section 28.3.3, including t mixture with Box-Cox transformation (flowClust), scale skew-normal (SSN) and skew-t (SST) mixtures, restricted skew-normal (rMSN) and skew-t (rMST) mixtures, and unrestricted skew-t mixture (uMST). All models are initialized by the partition obtained by the traditional K-means clustering. The algorithms stop when the stopping criterion meets the tolerance level of $1e-5$. For more information about the behavior of different models, we refer the reader to the recent paper by Zhu and Melnykov (2016a), where a comprehensive simulation study is conducted to compare model performance.

Table 4 provides model-based clustering results. The number of parameters of the models are 20 (flowClust), 25 (SSN), 25 (SST), 25 (rMSN), 27 (rMST), and 27 (uMST). The computing times are 0.012, 1.553, 4.737, 0.024, 0.136, 3547.527, respectively. The BIC values of the models are 3551.125, 3576.886, 3558.227, 3566.007, 3562.151, 3591.241. flowClust enjoys the lowest BIC value, which is still higher than that of Manly B. uMST yields the lowest number of misclassifications, which is as good as the full Manly mixture model and Manly forward model.

Acknowledgement

The research is partially funded by the University of Louisville EVPRI internal research grant from the Office of the Executive Vice President for Research and Innovation.

Table 4: Classification tables for the *AIS* dataset. Rows and columns represent the true and estimated partitions, respectively. The bold font highlights correct classifications.

	flowClust		SSN		SST	
Group	1	2	1	2	1	2
1	**99**	1	**99**	1	**99**	1
2	8	**94**	7	**95**	5	**97**
	rMSN		rMST		uMST	
Group	1	2	1	2	1	2
1	**100**	0	**99**	1	**98**	2
2	8	**94**	6	**96**	2	**100**

Bibliography

E. Anderson. The Irises of the Gaspe peninsula. *Bulletin of the American Iris Society*, 59:2–5, 1935. [p181, 183]

C. Biernacki, G. Celeux, and G. Govaert. Choosing starting values for the EM algorithm for getting the highest likelihood in multivariate Gaussian mixture models. *Computational Statistics and Data Analysis*, 413:561–575, 2003. [p190]

G. Celeux and G. Govaert. A classification EM algorithm for clustering and two stochastic versions. *Computational Statistics and Data Analysis*, 14:315–332, 1992. doi: 10.1016/0167-9473(92)90042-E. [p180]

D. Cook and S. Weisberg. *An Introduction to Regression Graphics*. John Wiley & Sons, New York., 1994. [p181, 187]

S. L. Crawford. An application of the laplace method to finite mixture distribution. *Journal of the American Statistical Association*, pages 259–267, 1994. doi: 10.1080/01621459.1994.10476467. [p192]

A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood for incomplete data via the EM algorithm (with discussion). *Journal of the Royal Statistical Society B*, 39:1–38, 1977. [p177]

R. A. Fisher. The use of multiple measurements in taxonomic poblems. *The Annals of Eugenics*, 7: 179–188, 1936. doi: 10.1111/j.1469-1809.1936.tb02137.x. [p181, 183]

N. A. Gillespie and M. C. Neale. A finite mixture model for genotype and environment interactions: Detecting latent population heterogeneity. *Twin Res Hum Genet*, 9(3):412–23, 2006. doi: 10.1375/183242706777591380. [p176]

S. X. Lee and G. J. McLachlan. Model-based clustering and classification with non-normal mixture distributions. *Statistical Methods and Applications*, 22(4):427–454, 2013. [p187, 189]

S. X. Lee and G. J. McLachlan. **EMMIXuskew**: Fitting unrestricted multivariate skew *t* mixture models. *Journal of Statistical Software*, 55(12), 2014. doi: 10.1007/s11222-012-9362-4. [p176]

K. Lo, F. Hahne, R. R. Brinkman, and R. Gottardo. **flowClust**: a bioconductor package for automated gating of flow cytometry data. *BMC Bioinformatics*, 10(145), 2009. doi: 10.1186/1471-2105-10-145. [p176]

R. Maitra and V. Melnykov. Simulating data to study performance of finite mixture modeling and clustering algorithms. *Journal of Computational and Graphical Statistics*, 19(2):354–376, 2010. doi: 10.1198/jcgs.2009.08054. [p178]

B. F. J. Manly. Exponential data transformations. *Biometrics Unit*, 25:37–42, 1976. [p177]

G. J. McLachlan and K. E. Basford. *Mixture Models: Inference and Applications to Clustering*. Marcel Dekker, New York, 1988. doi: 10.2307/2348072. [p178]

G. J. McLachlan and T. Krishnan. *The EM Algorithm and Extensions*. Wiley, New York, 2nd edition, 2008. doi: 10.1002/9780470191613. [p177]

G. J. McLachlan and D. Peel. *Finite Mixture Models*. John Wiley & Sons, New York, 2000. [p179, 184]

V. Melnykov. On the distribution of posterior probabilities in finite mixture models with application in clustering. *Journal of Multivariate Analysis*, 122:175–189, 2013. doi: 10.1016/j.jmva.2013.07.014. [p179]

S. Michael and V. Melnykov. Finite mixture modeling of gaussian regression time series with application to dendrochronology. *Journal of Classification*, 33(3):412–441, 2016. doi: 10.1007/s00357-016-9216-4. [p176]

B.-J. Park and D. Lord. Application of finite mixture models for vehicle crash data analysis. *Accident Analysis & Prevention*, 41(4):683–691, 2009. doi: 10.1016/j.aap.2009.03.007. [p176]

M. Prates, C. Cabral, and V. Lachos. **mixsmsn**: Fitting finite mixture of scale mixture of skew-normal distributions. *Journal of Statistical Software*, 54:1–20, 2013. doi: 10.18637/jss.v054.i12. [p176]

P. Schlattmann. *Medical applications of finite mixture models*. Springer, 2009. [p176]

G. Schwarz. Estimating the dimensions of a model. *The Annals of Statistics*, 6:461–464, 1978. [p179]

K. Wang, A. Ng, and G. McLachlan. **EMMIXskew**: The EM algorithm and skew mixture distribution, 2013. R package version 1.0.1. [p176]

J. H. Ward. Hierarchical grouping to optimize an objective function. *Journal of the American Statistical Association*, 58:236–244, 1963. [p186]

X. Zhu and V. Melnykov. Manly transformation in finite mixture modeling. *Computational Statistics and Data Analysis*, 2016a. doi: 10.1016/j.csda.2016.01.015. [p176, 195]

X. Zhu and V. Melnykov. **ManlyMix**: An R package for model-based clustering with manly mixture models, 2016b. R package version 0.1.9. [p176]

Xuwen Zhu
Department of Mathematics, The University of Louisville
Louisville, KY 40208
USA
xuwen.zhu@louisville.edu

Volodymyr Melnykov
Department of Information Systems, Statistics, and Management Science, The University of Alabama
Tuscaloosa, AL 35487
USA
vmelnykov@cba.ua.edu

adegraphics: An S4 Lattice-Based Package for the Representation of Multivariate Data

by Aurélie Siberchicot, Alice Julien-Laferrière, Anne-Béatrice Dufour, Jean Thioulouse and Stéphane Dray

Abstract The **ade4** package provides tools for multivariate analyses. Whereas new statistical methods have been added regularly in the package since its first release in 2002, the graphical functions, that are used to display the main outputs of an analysis, have not benefited from such enhancements. In this context, the **adegraphics** package, available on CRAN since 2015, is a complete reimplementation of the **ade4** graphical functionalities but with large improvements. The package uses the S4 object system (each graph is an object) and is based on the graphical framework provided by **lattice** and **grid**. We give a brief description of the package and illustrate some important functionalities to build elegant graphs.

Introduction

In many fields, data consists in tables containing measurements of several variables for a number of samples. In this context, multivariate analyses provide tools to summarize the main structures of multivariate data. After a dimension reduction step, these methods provide a graphical display of the primary relationships between variables and similarities between samples. Multivariate methods are routinely used in various fields including marketing, psychometry or ecology and have been implemented in several R packages (e.g. **vegan** (Oksanen et al., 2017), **MASS** (Venables and Ripley, 2002), **FactoMineR** (Lê et al., 2008), see the *Multivariate* Task View of CRAN). The **ade4** package (Dray and Dufour, 2007) is an alternative, distributed on CRAN since 2002, that implements more than 30 different methods, mainly oriented to the analysis of ecological data, and around 40 graphical functions to display the results of analyses.

During the last 15 years, a major effort was made in the implementation of new statistical methods in **ade4** but the graphical functionalities have not benefited from such improvements. At this time, graphs from **ade4** were black and white and lacked flexibility. It became increasingly difficult to customize the graphical outputs as the results of analyses are increasingly complex. In parallel, R has evolved and now proposes new paradigms of development (the S4 object system) and new graphical environments (**lattice** (Sarkar, 2008), `grid` (Murrell, 2005)). In this context, we started the development of a new add-on package: **adegraphics**. It is available on CRAN since 2015 and is designed to provide enhanced graphical functionalities for representing outputs of multivariate analysis, especially from **ade4**.

For users, **adegraphics** provides a flexible environment to produce, edit and easily manipulate graphs. To ensure continuity with the graphical functions of **ade4**, functions and parameters names have been largely preserved in **adegraphics**, so that regular users can easily migrate to the new package. Moreover, the graphical functions of **ade4** are not removed to preserve the functioning of other R packages and scripts that depend on **ade4**. The use of the new **adegraphics** functions is preferred and encouraged for future developments. Keeping the old version of the functions in **ade4** has some practical effects. In a classical case, when multivariate analyses are performed with **ade4**, it is imperative to load **ade4** before **adegraphics**. This solves the issues of conflicting names between functions by ensuring that the versions from **adegraphics** will be used.

```
> library(ade4)
> library(adegraphics)
```

The **adegraphics** package can be installed from CRAN. A development release is available on GitHub (`https://github.com/sdray/adegraphics`) and a detailled description of the package is given in the vignette available at `https://cran.r-project.org/web/packages/adegraphics/vignettes/adegraphics.html` or in any R session by typing `vignette("adegraphics")`. A reproducible version of the code in this paper is available online at `http://lbbe-shiny.univ-lyon1.fr/Reproducible_Research/TRJ.Siberchicot.2017/`.

This paper is divided in three parts: simple graphs, multiple graphs, and multivariate analysis graphs. The simple graphs part includes a presentation of basic elements and graphical parameters, with examples of graph manipulation and spatial representations. The multiple graphs part shows how to do automatic collections of graphs, and how to create and customize multiple graphs. The

multivariate analysis part shows an example of how the plot of a coinertia analysis can be customized and improved with **adegraphics**.

A simple example

We illustrate some principles and functionalities of **adegraphics** by considering the analysis of the `mafragh` data set available in **ade4**. It contains a phyto-ecological survey collected in an Algerian plain, called *La Mafragh* (de Bélair and Bencheikh-Lehocine, 1987; Pavoine et al., 2011).

```
> data("mafragh")
> names(mafragh)
```

```
[1] "xy"              "flo"             "neig"            "env"
[5] "partition"       "area"            "tre"             "traits"
[9] "nb"              "Spatial"         "spenames"        "Spatial.contour"
```

The aim of the original study (de Bélair and Bencheikh-Lehocine, 1987) was to propose an integrated management project of the Mafragh marshy coastal plain. `mafragh$flo` is a data frame giving the abundance of 56 plant species (columns) in 97 sites (rows) distributed in the plain. `mafragh$env` is a data frame with 97 rows and 11 columns describing the physico-chemical and topographic characteristics, and the granulometry of soil samples taken at the same sampling sites. `mafragh$partition` is a factor defined by de Bélair and Bencheikh-Lehocine (1987), classifying the 97 sites in 7 ecologically coherent regions according to observed vegetation units (ecoregions).

Basic elements and simple graphs

Classes, objects and calling functions

The **adegraphics** package uses object-oriented programming (OOP) in R: graphs are S4 objects that can be displayed or stored for later modification.

In **adegraphics**, there are two main parent classes of objects, one to store simple graphs (i.e. with only one represented information) called "ADEg" and the other, called "ADEgS", to store multiple graphs. The "ADEg" class has 5 sub-classes dedicated to different representation types: the "ADEg.T" class can be used to represent raw data such as distance matrices or contingency tables, the "ADEg.S2" class is associated with bidimensional data and can show factorial maps, the "ADEg.S1" class is used to represent a numeric score such as only one factorial axis in a unidimensional graph, the "ADEg.C1" class allows to represent unidimensional data associated with a supplementary information into two dimensions and the "ADEg.Tr" class is a peculiar class to represent data in a ternary plot. Each of these sub-classes itself has several child classes. The list of classes evolves and can be enriched to satisfy future developments. The **adegraphics** package provides user-friendly functions to create each type of graphical object. Some examples of these functions are listed in the Table 1 for each of the 5 sub-classes.

"ADEg" sub-classes	Example of sub-classes	Example of associated user functions
"ADEg.T"	"T.image", "T.value", ...	`table.image`, `table.value`, ...
"ADEg.S2"	"S2.class", "S2.corcircle", "S2.density", "S2.value", ...	`s.class`, `s.corcircle`, `s.density`, `s.value`, ...
"ADEg.S1"	"S1.boxplot", "S1.distri", "S1.match", ...	`s1d.boxplot`, `s1d.distri`, `s1d.match`, ...
"ADEg.C1"	"C1.curve", "C1.gauss", "C1.hist", ...	`s1d.curve`, `s1d.gauss`, `s1d.hist`, ...
"ADEg.Tr"	"Tr.class", "Tr.label", "Tr.match", "Tr.traject"	`triangle.class`, `triangle.label`, `triangle.match`, `triangle.traject`

Table 1: Examples of classes and associated graphical functions in `adegraphics`

In our example, a normed principal component analysis (PCA) is applied (using the `dudi.pca` function of **ade4**) on the `mafragh$env` data table. This table contains the measurements of 11 environmental variables for 97 sites. The first four axes are kept and the results are stored in object `pca1`.

```
> pca1 <- dudi.pca(mafragh$env, scale = TRUE, center = TRUE, scannf = FALSE, nf = 4)
```

The sites can be displayed on the factorial map formed by the first two axes. A graphical object is created (by the call to the `s.label` function) but not printed because of the argument `plot = FALSE`. Correlations between environmental variables and the first two PCA axes are represented on a correlation circle (with the user function `s.corcircle`, Figure 1). Both graphs (`g_sl1` and `g_sc1`) are stored in S4 objects of different sub-classes.

```
> g_sl1 <- s.label(pca1$li, plot = FALSE)
> class(g_sl1)

[1] "S2.label"
attr(,"package")
[1] "adegraphics"

> g_sc1 <- s.corcircle(pca1$co)
> class(g_sc1)

[1] "S2.corcircle"
attr(,"package")
[1] "adegraphics"
```

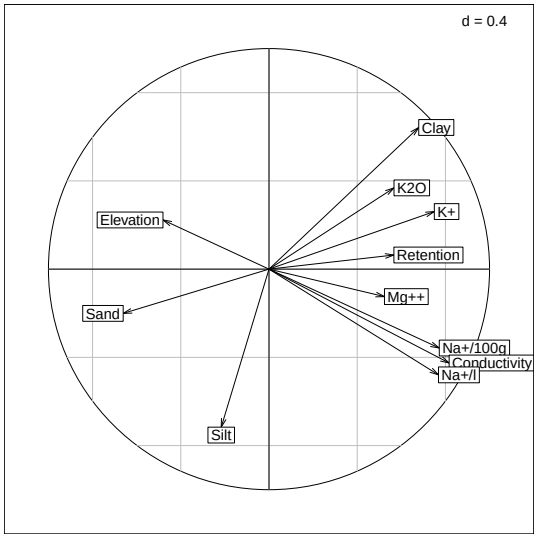


Figure 1: Default representation of the correlation circle of the 11 environmental variables on the first two PCA axes (`g_sc1`). The direction and length of arrows show correlations between variables and principal components. This graph helps to describe the Mafragh environment. The first axis is a salinity/elevation gradient with high salinity and low altitude on the right, opposed to high elevation and low salinity on the left. The second axis refines this gradient by opposing clay and potassium ions (up) to sodium ions and silts (down).

Each simple graphical object created with **adegraphics** (i.e. object belonging to class "ADEg") is defined by 8 attributes. These attributes are reachable by the @ operator and their names are given by the `slotNames` function.

```
> slotNames(g_sc1)

[1] "data"         "trellis.par"  "adeg.par"     "lattice.call" "g.args"
[6] "stats"        "s.misc"       "Call"
```

The `data` slot contains information about the data used in the plot and the `Call` slot contains the matched call.

```
> g_sc1@Call

s.corcircle(dfxy = pca1$co, xax = 1, yax = 2, labels = row.names(as.data.frame(pca1$co)),
    fullcircle = TRUE, facets = NULL, plot = TRUE, storeData = TRUE,
    add = FALSE, pos = -1)
```

Slots s.misc and stats store lists of internal parameters and internal preliminary computations. The lattice.call slot contains all the information required to create a "trellis" object (see the **lattice** package) associated to the **adegraphics** instruction. Note that an **adegraphics** object can be converted to a **lattice** one using the gettrellis function. The trellis.par, adeg.par and g.args slots manage several graphical parameters and are detailed in the next section.

Graphical parameters

A great flexibility is provided in **adegraphics** by the management of many graphical parameters. Parameter names are more intuitive and their number is greater than in **ade4**. There are 3 kinds of parameters in **adegraphics** user functions.

First, parameters implemented in **adegraphics** are itemized by the adegpar function (that works like the basic par function). When the graphical object is created, these parameters are stored in the adeg.par attribute. For example, the adeg.par attribute of the g_sc1 object is yielded by g_sc1@adeg.par or by getparameters(g_sc1,2). These parameters can be applied locally on a graph (as arguments to a user function) or globally in the working environment (using the adegpar function) and so applied on all graphs created thereafter.

Second, **lattice** parameters are managed in **adegraphics** and stored in the trellis.par attribute of the created object. For example, the trellis.par attribute of the g_sc1 object is returned by g_sc1@trellis.par or by getparameters(g_sc1,1). These parameters are itemized by the **lattice** function trellis.par.get and can also be applied locally or globally (see the trellis.par.set function of **lattice**).

Last, some general parameters are also managed like xlim and ylim, xlab and ylab, main, scales. These parameters are stored in the g.args attribute of the created object.

The **adegraphics** vignette explains more precisely how to manage these various graphical parameters and which one can be applied according to the class of the created object. The vignette appendix lists the correspondences between the graphical parameters used in **ade4** and their translation in **adegraphics**.

In the example below, the representation of environmental variables onto the first factorial map of pca1 in the correlation circle (Figure 1) is customized: boxes around labels are removed and a subtitle is added. The background color of this new graph (called g_sc2) can be reached as the element col of the pbackground list of the adeg.par attribute. Here, the background color of g_sc2 is "white". Note that g_sc2 is not printed right now (plot = FALSE).

```
> g_sc2 <- s.corcircle(pca1$co, plabels.boxes.draw = FALSE,
     psub.text = "Correlations of the environmental variables", plot = FALSE)
> g_sc2@adeg.par$pbackground$col

[1] "white"
```

Manipulating a simple graph

There are several methods in **adegraphics** to manipulate the graphical objects created and stored. For example, the update method allows to modify a posteriori some graphical parameters of a graph previously created with **adegraphics**, without recreating it. Besides, users can zoom in or out a stored graph thanks to the zoom method. Note that this function is only allowed for some sub-classes of **adegraphics**. Simply, the plot, print and show methods can be used to display a stored **adegraphics** object. Other functions and methods are described in the vignette of the **adegraphics** package.

In the example below, the grid and background color of the previously created g_sc2 object are changed using the update method (Figure 2). As above, the background color parameter is reachable through the adeg.par slot of the g_sc2 object; now it is "grey90".

```
> update(g_sc2, pgrid.col = "white", pbackground.col = "grey90")
> g_sc2@adeg.par$pbackground$col

[1] "grey90"
```

Representation of spatial information

In **ade4**, classes were implemented to manage spatial information: the "area" class for geographic maps and the "neig" class to store spatial neighborhood objects. The **adegraphics** package abandoned

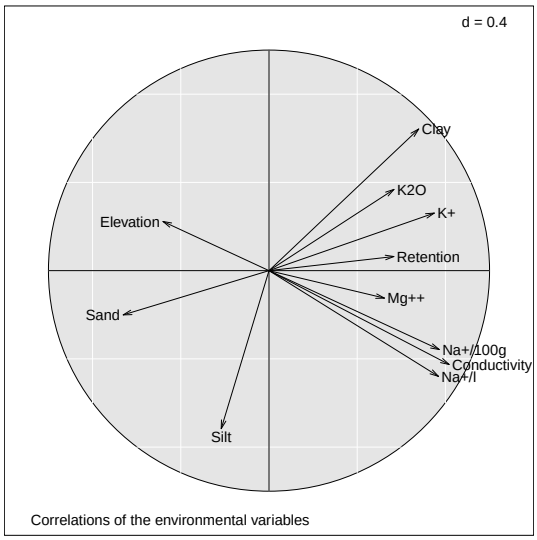


Figure 2: Customization of the g_sc2 object using the update method. Labels are more readable and the graph is highlighted by changing the colors and adding a title.

these outdated implementations and adopted more efficient classes. Maps are now considered as objects inherited from classes of the **sp** package (Pebesma and Bivand, 2005; Bivand et al., 2013b). "nb" and "listw" objects, from the **spdep** package (Bivand and Piras, 2015; Bivand et al., 2013a), are used to manage spatial neighborhood graphs. Note that only "ADEg.S2" objects (bidimensional plot of xy coordinates, implemented in s.* functions) of **adegraphics** support the representation of spatial objects by the use of arguments Sp and nb. See also the useful s.Spatial function to easily draw thematic maps. **adegraphics** provides specific parameters to manage the customization of spatial objects: the pSp parameters affect maps and pnb affects neighborhood.

In our example, a thematic map is simply obtained by displaying the score of sites on the first PCA axis (pca1$li[,1]) on a spatial map used as background. The s.value function is hereafter used to create the g_sv1 object (Figure 3, left) and takes as argument the mafragh$Spatial.contour object (an object of a class from the **sp** package) in the Sp parameter and a color palette in the col parameter. As many colors as classes of value are stored in a valuecolors vector created with the brewer.pal function of the **RColorBrewer** package (Neuwirth, 2014).

```
> class(mafragh$Spatial.contour)

[1] "SpatialPolygons"
attr(,"package")
[1] "sp"

> library(RColorBrewer)
> valuecolors <- rev(brewer.pal(6, "RdBu"))
> g_sv1 <- s.value(mafragh$xy, z = pca1$li[, 1], Sp = mafragh$Spatial.contour,
     method = "color", symbol = "circle", col = valuecolors, pgrid.draw = FALSE,
     ppoints.cex = 0.4)
```

A similar graph (called g_sp1) can easily be built (Figure 3, right) using the s.Spatial function and a "SpatialPolygonsDataFrame" object (from the **sp** package). The spobj object couples a "SpatialPolygons" object with the scores of sites on the first PCA axis.

```
> library(sp)
> spobj <- SpatialPolygonsDataFrame(Sr = mafragh$Spatial, data = pca1$li[,
     1, drop = FALSE], match.ID = FALSE)
> class(spobj)

[1] "SpatialPolygonsDataFrame"
attr(,"package")
[1] "sp"

> g_sp1 <- s.Spatial(spobj, col = valuecolors)
```

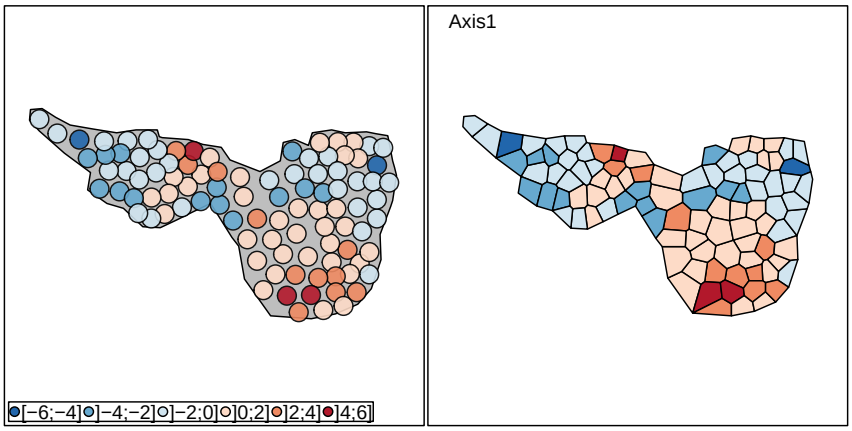


Figure 3: Thematic map: the scores on the first PCA axis are mapped in the geographical space. The `g_sv1` object (left) is created by a call to the `s.value` function and the `g_sp1` object (right) by the `s.Spatial` function. The color of sites is given by the value of scores on the first axis. The PCA first axis described by the correlation circle in Figure 2 helps interpret this figure: regions with high elevation (blue color, negative coordinates) are located at the extreme west and east, while the lowest points are located in the center of the map. Salinity (particularly potassium ions) is high in the southern region (red color, positive coordinates) because local conditions (clay) prevent the drainage of sea waters that accummulate there. Salinity (sodium ions) is also high in a particular region of the north because it is just next to the sea shore and receives a lot of sea water that stays trapped here.

Dealing with multiple graphs

The "ADEgS" class stores a collection of "ADEg", "ADEgS" and/or "trellis" (from the **lattice** package) objects, as a list of graphical objects. It is defined by 4 attributes, given by the `slotNames` function applied on a "ADEgS" object:

- `ADEglist` contains the list of graphs (i.e. objects) in the collection;
- `positions` is a matrix that contains the positions of subgraphs in the display;
- `add` is a matrix that contains the information about subgraphs superposition;
- `Call` contains the matching call.

"ADEgS" objects can be simply manipulated with methods associated to lists (i.e. `$`, `[]`, `[[]]`). The `names` function outputs the labels of subgraphs contained in an "ADEgS" object and the `length` function returns the number of subgraphs stored in the object.

We implemented different ways to generate collections of graphs. "ADEgS" objects can be created by a call to a simple function or built step-by-step using longer code. They can be customized in the same way as "ADEg" objects.

Automatic collections

Several user-friendly functionalities are implemented in **adegraphics** to facilitate the creation of collections of graphs, repeating (without using a for-loop command) a simple graph for different groups of individuals, variables or axes. The building of these collections is very simple for the user (definition of an argument in the call of a function) and leads to the creation of an "ADEgS" object.

For instance, as in the **ggplot2** package (Wickham, 2016), a `facets` argument is available to split a dataset into groups of individuals and to produce one graph per group. The `g_sv2` object (Figure 4) displays the scores of sites of the first PCA axis (as in the `g_sv1` object, left of Figure 3) for each of the 7 ecoregions, called `C1`, . . . ,`C7`, defined in `magragh$partition`.

```
> g_sv2 <- s.value(mafragh$xy, z = pca1$li[, 1], facets = mafragh$partition,
     Sp = mafragh$Spatial.contour, method = "color", symbol = "circle",
     col = valuecolors, pgrid.draw = FALSE, ppoints.cex = 0.4)
```

The `g_sv2` object is an "ADEgS" containing 7 subgraphs belonging to the "S2.value" class. The subgraphs names are given by the `names` function and are equal to the level names of `mafragh$partition`. Combined with the `$` symbol, these names can be used to select only one subgraph to the "ADEgS" object. For example, `g_sv2$C1` is the extraction of the `C1` subgraph of `g_sv2`.

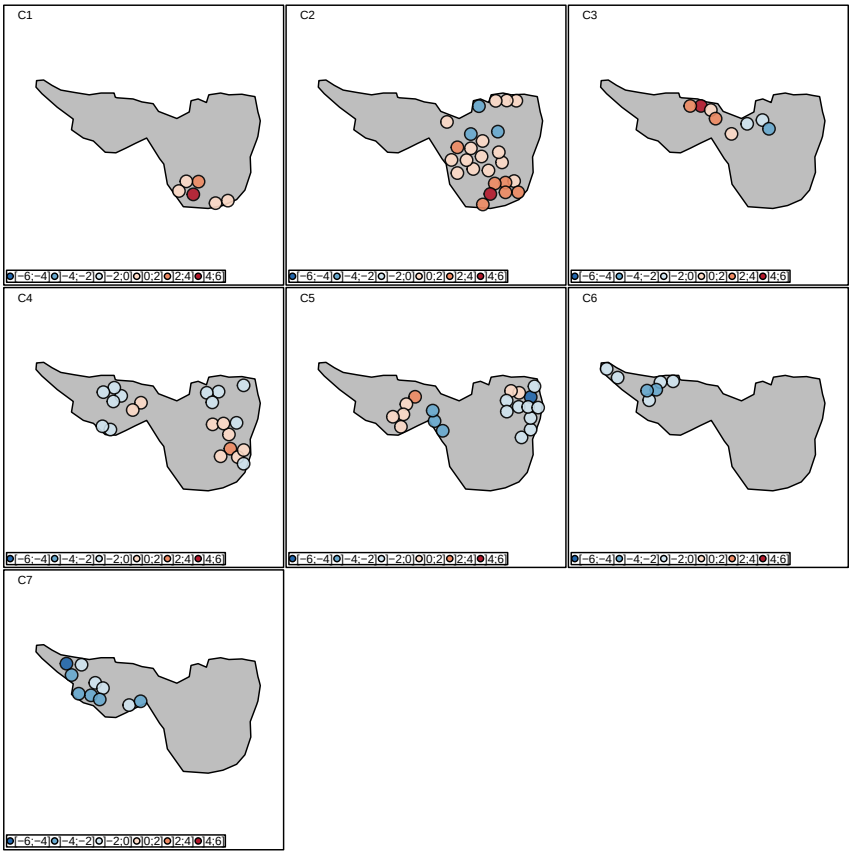


Figure 4: Automatic creation of an "ADEgS" object using the `facets` argument. The `g_sv2` object is created by only a call to the `s.value` function. Seven thematic maps (one for each ecoregion) are displayed to facilitate the read of PCA results. These 7 ecoregions were defined by de Bélair and Bencheikh-Lehocine (1987) from a correspondence analysis of the plant species data table (`mafragh$flo`) and phytosociological interpretations. Each ecoregion takes into account the distribution of plant species of various floristic groups, but also pedological characteristics of sampling sites, and geographical areas. For example, ecoregion `C1` is defined by 6 pedological sampling sites (numbered `43`, `44`, `45`, `46`, `49` and `51` in `g_sv2$C1@data$dfxy`) and the *Bolboschoenus maritimus* and *Schoenoplectus litoralis* plant communities (high abundances in `mafragh$flo[c(43, 44, 45, 46, 49, 51), ]`).

```
> class(g_sv2)

[1] "ADEgS"
attr(,"package")
[1] "adegraphics"

> names(g_sv2)

[1] "C1" "C2" "C3" "C4" "C5" "C6" "C7"

> class(g_sv2$C1)

[1] "S2.value"
attr(,"package")
[1] "adegraphics"
```

On the other hand, multiple graphs can also be produced by splitting graphs by columns. This is achieved when a data frame with several variables is given as an argument to a function that usually requires a vector. For instance, the abundance of four focused species is easily displayed along the `mafragh` sites. The `g_sv3` object (Figure 5) is created with the `s.value` function, using the `mafragh$flo[, selectedspecies]` data frame as `z` parameter. A line surrounding the entire `mafragh` region (contained in the `mafragh$Spatial.contour` object) is added thanks to the `Sp` argument.

```
> selectedspecies <- c(9, 12, 31, 34)
> floselectedspecies <- mafragh$flo[, selectedspecies]
```

```
> colnames(floselectedspecies) <- mafragh$spenames$code[selectedspecies]
> colnames(floselectedspecies)

[1] "Juma" "Scli" "Boof" "Plco"

> g_sv3 <- s.value(mafragh$xy, z = floselectedspecies, symbol = "circle",
     centerpar = list(cex = 0.1), ppoints.cex = 0.7, pgrid.draw = FALSE, psub.cex = 2,
     porigin.draw = FALSE, plegend.drawK = FALSE, Sp = mafragh$Spatial.contour)
```

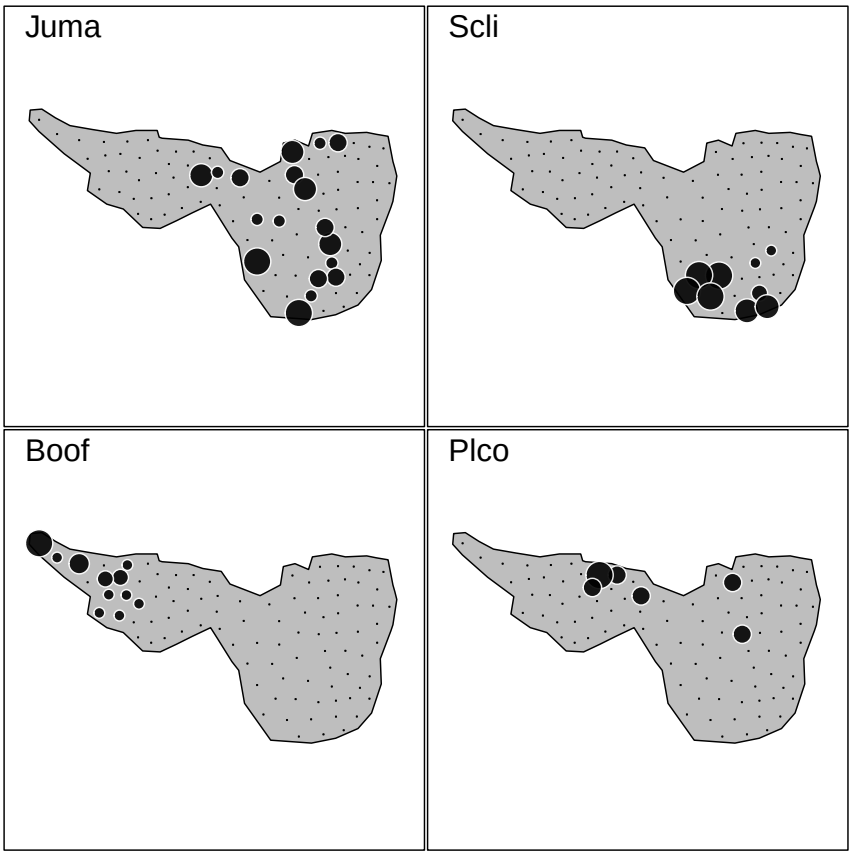


Figure 5: Automatic creation of an "ADEgS" object. The `g_sv3` object is created by only a call to the `s.value` function using a multivariate argument `z`. Four maps of abundance are produced, one for each of four species of interest: *Juncus maritimus* (`Juma`), *Schoenoplectus litoralis* (`Scli`), *Borago officinalis* (`Boof`) and *Plantago coronopus* (`Plco`).

Step-by-step creation

"ADEgS" objects can also be created by the manipulation of several simple graphs. Several functions in **adegraphics** allow superposition, combination or insertion of two objects: `superpose`, `+`, `cbindADEg`, `rbindADEg` and `insert`.

Following our example, a Correspondence Analysis (CA, called `coa1`) is performed on the abundance of the 56 plant species of `mafragh` (contained in the `mafragh$flo` data table) with the `dudi.coa` function of the **ade4** package, displayed but not shown with the `s.label` function and stored in a `g_sl2` object. Note here the `plabel.optim` parameter set to `TRUE` which automatically optimize the label positions and remove label boxes.

```
> coa1 <- dudi.coa(mafragh$flo, scannf = FALSE, nf = 2)
> g_sl2 <- s.label(coa1$co, labels = mafragh$spenames$code, plabel.optim = TRUE,
     plot = FALSE)
```

To help interpret the results of this CA, three among four maps of species abundance created in the `g_sv3` object can be inserted, one at a time, close to the matching species point on the `g_sl2` object. The `insert` function takes as parameters a first graph to insert, a second one in which to insert and a

posi argument to define the insertion position. Each of the three inserted subgraph is extrated from the g_sv3 object by the [[]] method. The final output called g_in1 is an object of "ADEgS" class (Figure 6).

```
> g_in1 <- insert(g_sv3[[1]], g_sl2, posi = c(0.06, 0.16), plot = FALSE)
> g_in1 <- insert(g_sv3[[2]], g_in1, posi = c(0.15, 0.75), plot = FALSE)
> g_in1 <- insert(g_sv3[[3]], g_in1, posi = c(0.57, 0.77))
> class(g_in1)

[1] "ADEgS"
attr(,"package")
[1] "adegraphics"
```

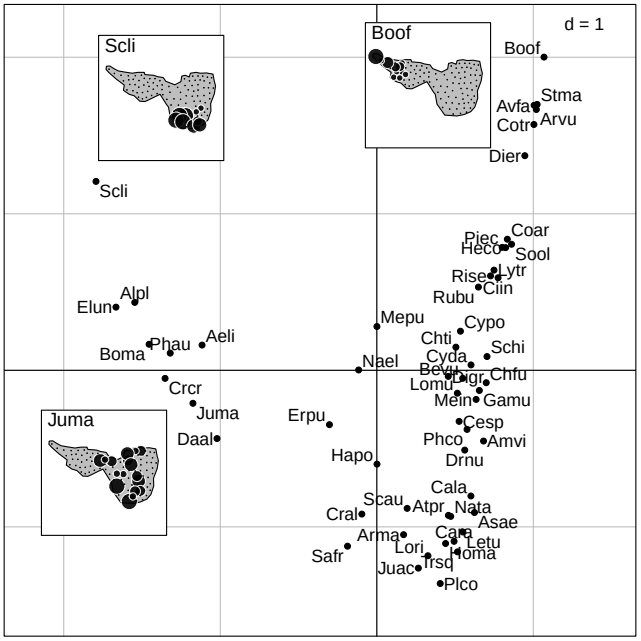


Figure 6: Creation of an "ADEgS" by the manipulation of four "ADEg" objects. This display can be produced by the insert or ADEgS function. Three geographical maps are superimposed on the first factor map of the CA of the plant species table. The spatial distribution of *Juncus maritimus* (Juma), *Schoenoplectus litoralis* (Scli) and *Borago officinalis* (Boof) are very different, and the CA scores clearly take these differences into account, even though spatial coordinates of sampling sites are not used in the analysis. They correspond to ecoregions C1, C2 and C6, respectively.

This object has four elements (i.e. graphs). As for a standard list, the function names can be used to modify labels of each g_in1 subgraph:

```
> names(g_in1)

[1] "g1"  "g2"  "X"   "X.1"

> names(g_in1) <- c("CA", "Juma", "Scli", "Boof")
> names(g_in1)

[1] "CA"   "Juma" "Scli" "Boof"
```

The insert function is a shortcut that calls a very flexible function designed to generate "ADEgS" objects. The general ADEgS function allows to arrange multiple graphs. It takes as arguments (i) a list of several "ADEg", "ADEgS" and/or "trellis" objects and (ii) information about their layout.

Below, the g_in2 object, created by the ADEgS function, is strictly identical to g_in1. The positions of each of the four subgraphs is yielded by the positions attribute of the g_in1 object.

```
> g_in1@positions

     [,1] [,2] [,3] [,4]
     0.00 0.00 1.00 1.00
```

```
positions 0.06 0.16 0.26 0.36
positions 0.15 0.75 0.35 0.95
positions 0.57 0.77 0.77 0.97
```

```
> g_in2 <- ADEgS(list(CA = g_sl2, Juma = g_sv3[[1]], Scli = g_sv3[[2]],
     Boof = g_sv3[[3]]), positions = rbind(c(0, 0, 1, 1), c(0.06,
     0.16, 0.26, 0.36), c(0.15, 0.75, 0.35, 0.95), c(0.57, 0.77,
     0.77, 0.97)), plot = FALSE)
```

Customizing an "ADEgS"

Like simple graphs ("ADEg"), the multiple graphs generated with **adegraphics** can be customized during or after the creation of the object, thanks to the update function. It is possible to apply changes to all or only one subgraph contained in the "ADEgS". To modify a graphical parameter for all subgraphs of the "ADEgS", the syntax is the same as for a simple "ADEg". To modify a graphical parameter for only a given subgraph, the parameter name must be preceded by the name of the subgraph, separated by a dot. This supposes that the subgraph names (available by a call to the names function) is known.

For example, the background color of the g_in1 object can be updated for all subgraphs:

```
> update(g_in1, pbackground.col = "grey90")
```

or only for the one called CA:

```
> update(g_in1, CA.pbackground.col = "grey90")
```

The "ADEgS" created hereafter by the ADEgS function is a good example how to manipulate "ADEg" objects and deal with **adegraphics** graphical parameters. The g_adegs1 object is a personal graph (Figure 7) to explore pca1 results. It arranges four "ADEg" objects in a personal layout.

First, default global graphical parameters are stored in a oldadegpar object in order to be restored at the end by the adegpar function.

```
> oldadegpar <- adegpar()
```

Two graphs called g_gau1d and g_lab1d are created to represent the scores of ecoregions and environmental variables on the first pca1 axis. Parameters about unidimensional displays, affecting this two graphs, are globally modified by the call of the adegpar function.

```
> adegpar(p1d = list(horizontal = FALSE, rug.tck = 1, margin = 0.07))
> g_gau1d <- s1d.gauss(pca1$li[, 1], fac = mafragh$partition, col = c(1:6, 8),
     p1d.reverse = TRUE, p1d.rug.margin = 0.1, plabels.cex = 2, plot = FALSE)
> g_lab1d <- s1d.label(pca1$co[, 1], labels = rownames(pca1$co), plabels.cex = 2,
     plot = FALSE)
```

Then the thematic map of the scores of sites on the first pca1 axis (g_sv1, Figure 3 at left) is kept back.

Finally the graph of the pca1 eigenvalues is created (called g_eig) with the relevant plotEig function of **adegraphics**. Here, only the first axis is kept to interpret the results (see the black bar).

```
> g_eig <- plotEig(pca1$eig, xax = 1, yax = 1, nf = 1, pbackground.box = TRUE,
     plot = FALSE)
```

The g_adegs1 can now be created, arranging the four graphs previously stored.

```
> g_adegs1 <- ADEgS(list(g_gau1d, g_lab1d, g_sv1, g_eig),
     layout = matrix(c(1, 2, 3, 1, 2, 4), nrow = 2, byrow = TRUE), plot = FALSE)
```

After the "ADEgS" creation, some graphical parameters can be a posteriori updated: the grid is removed on all graphs, a title is added to the third (called g3 in the g_adegs1 object) and fourth (called g4 in the g_adegs1 object) graphs and these titles are enlarged.

```
> names(g_adegs1)
```

```
[1] "g1" "g2" "g3" "g4"
```

```
> update(g_adegs1, pgrid.draw = FALSE, psub.cex = 1.6,
     g3.psub = list(text = "Map of scores of the first PCA axis"),
     g4.psub = list(text = "PCA eigenvalues"))
> adegpar(oldadegpar)
```

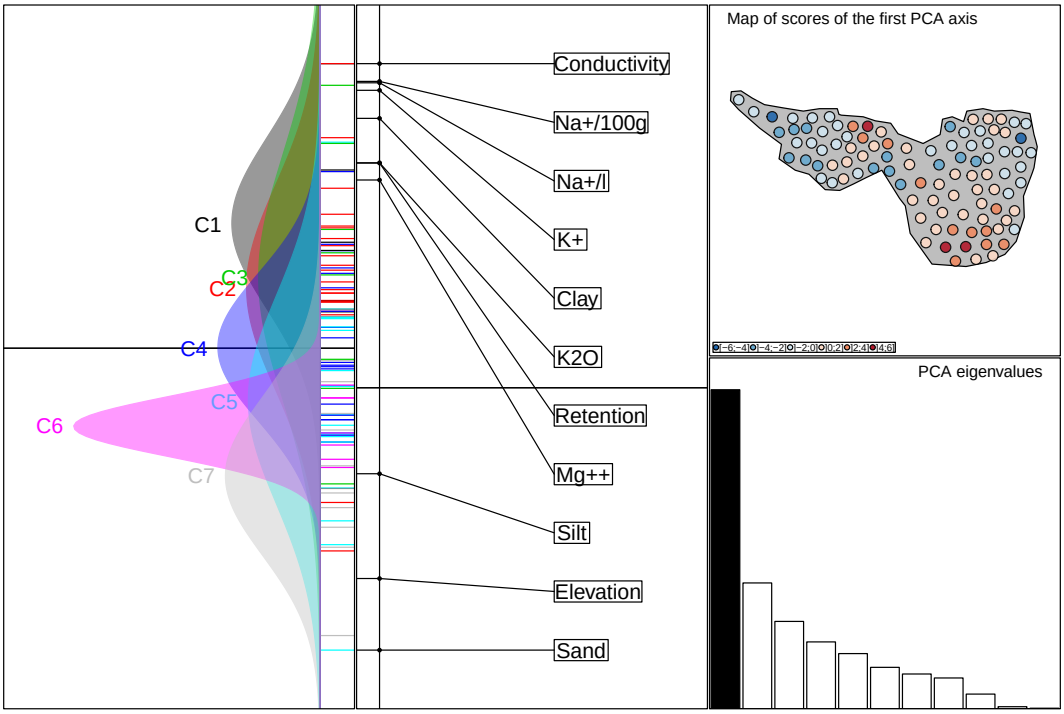


Figure 7: Creation of an "ADEgS" by the manipulation of four "ADEg" objects. Some graphical parameters are globally set before creating the graphs, others are defined when each subgraph is created, and others are updated after the final creation. The left part of the figure is the one-dimensional biplot of the PCA first axis. Gauss curves on the left show the distribution of sampling site PCA scores for the seven ecoregions. The labels on the right show the physico-chemical characteristics of each ecoregion, as evidenced by the PCA. For example, the sampling sites of ecoregion `C1` are characterized by a high conductivity and high levels of salinity, while sites of ecoregions C5, C6 and C7 are higher and more sandy. The top right map recalls that this first axis has a strong spatial structure (see legend to Figure 1), and the bottom right barplot shows that this corresponds to a large part (41%) of the total inertia.

Multivariate analysis outputs

All functions formerly available in **ade4** to display the outputs of multivariate analyses have been reimplemented in **adegraphics**. Most of them return an "ADEgS" object.

In our example, the PCA on environmental variables is now applied with the CA weights (`coa1$lw`) and stored in `pca2`. Then, a coinertia analysis is built to identify relationships between environmental variables (`pca2`) and species distributions (`coa1`). The results are stored in a `coi1` object and are graphically summarized by a call to the `plot` method.

```
> pca2 <- dudi.pca(mafragh$env, row.w = coa1$lw, scannf = FALSE, nf = 2)
> coi1 <- coinertia(coa1, pca2, scannf = FALSE, nf = 3)
> g_coi1 <- plot(coi1)
```

The resulting graphical object (`g_coi1`, Figure 8) is an "ADEgS" containing 6 subgraphs displaying the main outputs required to interpret the results.

```
> length(g_coi1)

[1] 6
```

In **ade4**, the graphical outputs provided by the `plot` function were initially designed to obtain a quick view on the main outputs. However, the number of graphs and their relatively small size did not allow a deep interpretation of the results. Hence, it was often required to write some new R code to redraw only the useful information in a publication-ready format. In **adegraphics**, the process is simplified because results are produced as "ADEgS" objects that can be manipulated as standard lists. Each graph is an element of this list and can be easily extracted using the names of elements (for the `$` operator) or their index in the list (for `[[]]`). The way a graph is created with **adegraphics** is reachable with the `Call` attribute of this object.

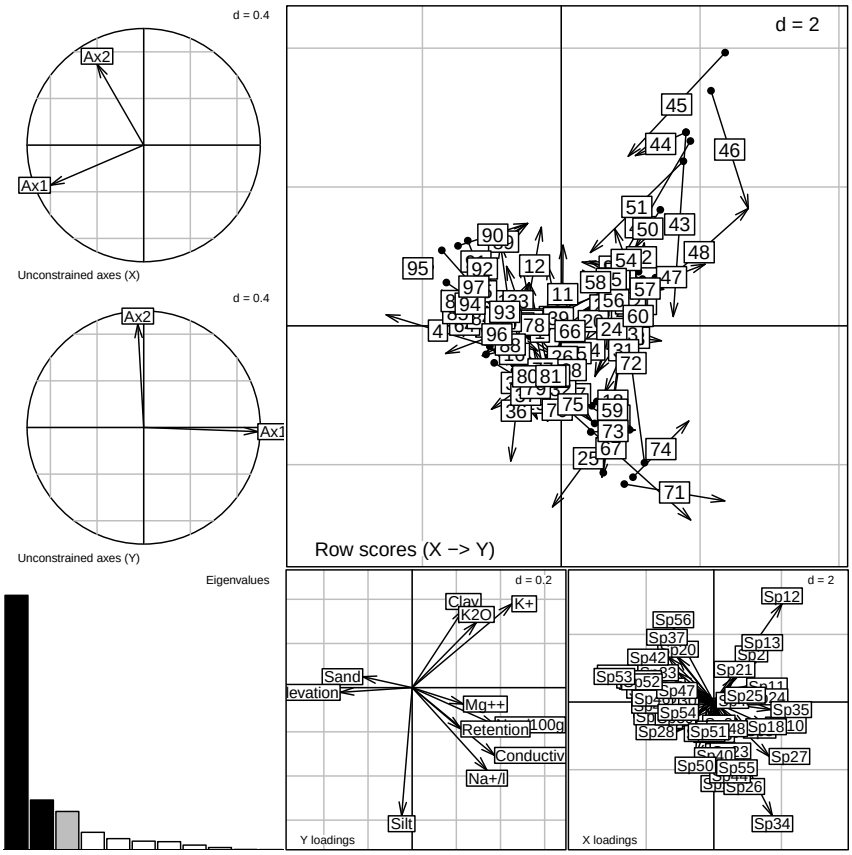


Figure 8: Default representation of a coinertia analysis. The `g_coi1` object is a collection of 6 graphs produced by the `plot.coinertia` method implemented in **adegraphics**.

```
> names(g_coi1)

[1] "Xax"       "Yax"       "eig"       "XYmatch"   "Yloadings" "Xloadings"

> g_coi1$XYmatch      # the same as g_coi1[[4]]

> g_coi1@Call

plot.coinertia(x = coi1)

> g_coi1$XYmatch@Call

s.match(dfxy1 = coi1$mX, dfxy2 = coi1$mY, xax = 1, yax = 2, plot = FALSE,
    storeData = TRUE, pos = -3, psub = list(text = "Row scores (X -> Y)"),
    labels = row.names(as.data.frame(coi1$mX)), arrows = TRUE,
    facets = NULL, add = FALSE)
```

In some subgraphs of Figure 8, information is not easy to read and not necessarily relevant. Thanks to the graphical facilities developped in **adegraphics**, a new multiple graph can be created to better explore coinertia results.

First, to improve the top-right subgraph (`g_coi1$XYmatch`) which is unreadable, a scatter plot is created (`g_scl`) to represent the scores of sites on the first factorial map of the coinertia, not for each sites (as in `g_coi1$XYmatch`) but grouped and colored by ecoregion. To build the new graph, it is usefull to know how `g_coi1$XYmatch` was created thanks to its `Call` attribute (see above). Then the eigenvalues barplot of the `coi1` coinertia (the subgraph called `g_coi1$eig`, located at the bottom left of `g_coi1`, in Figure 8) is inserted at the topleft of `g_scl`.

```
> g_scl1 <- s.class(coi1$mX, fac = mafragh$partition, ellipseSize = 0, chullSize = 0,
     starSize = 0.5, ppoints.cex = 0, col = c(1:6, 8), plot = FALSE)
> g_scl2 <- s.class(coi1$mY, fac = mafragh$partition, ellipseSize = 0, chullSize = 0,
     starSize = 0.5, ppoints.cex = 0, col = c(1:6, 8), plot = FALSE)
```

```
> g_scs1 <- superpose(g_scl1, g_scl2)
> g_sm1 <- s.match(g_scl1@stats$means, g_scl2@stats$means, plabels.cex = 0,
    plines.lwd = 2, plot = FALSE)
> g_scl <- superpose(g_scs1, g_sm1)
> g_scl <- insert(g_coi1$eig, g_scl, posi = "topleft", ratio = 0.3, plot = FALSE)
```

Next, the `g_coi1$Yloadings` subgraph is stored in a new `g_yload` object and then updated: boxes around labels and grid are removed; the subtitle is enlarged.

```
> g_yload <- g_coi1$Yloadings
> update(g_yload, plabels.box.draw = FALSE, pgrid.draw = FALSE, psub.cex = 1.2,
    plot = FALSE)
```

Then a `g_sl3` object is inspired by the bottom-right subgraph (`g_coi1$Xloadings`) of `g_coi1`: a `s.label` function is used instead of the `s.arrow` one, labels are extracted from the `mafragh$spenames$code` vector, labels positions are optimized, points and grid are removed.

```
> g_sl3 <- s.label(dfxy = coi1$c1, psub = list(text = "X loadings"),
    labels = mafragh$spenames$code, plabels.optim = TRUE, plabels.cex = 1.2,
    psub.cex = 1.2, ppoints.cex = 0, pgrid.draw = FALSE, plot = FALSE)
```

The `g_adegs2` object is at last created with the `ADEgS` function.

```
> g_adegs2 <- ADEgS(list(g_scl, g_yload, g_sl3),
    layout = list(mat = matrix(c(1, 1, 2, 1, 1, 3), nrow = 2, byrow = TRUE)))
```

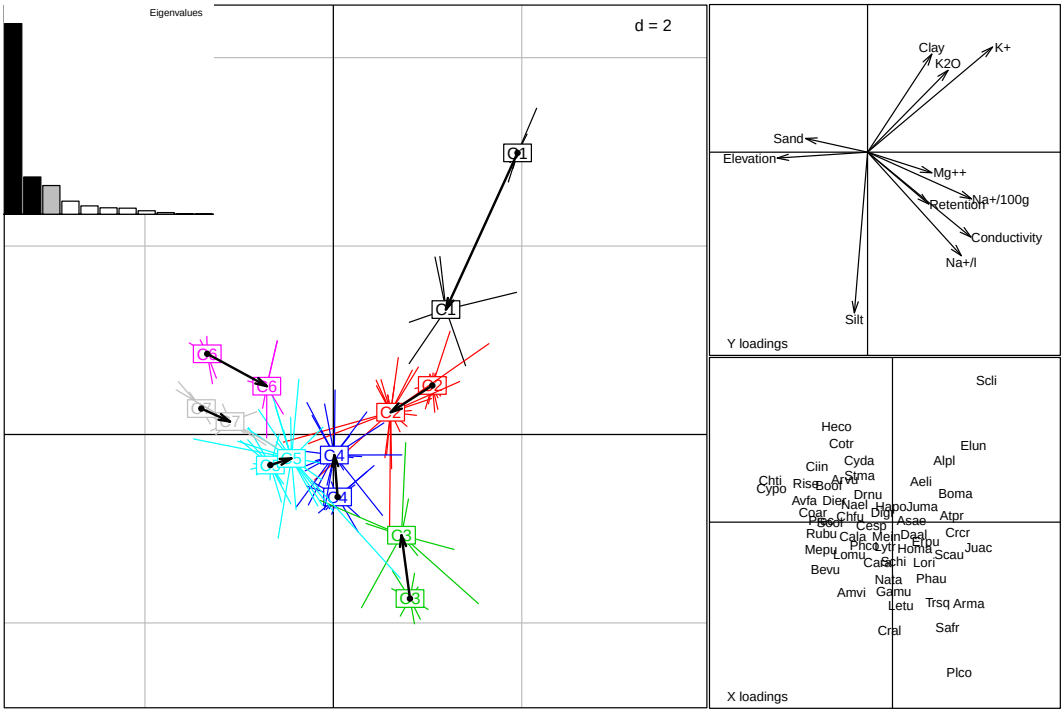


Figure 9: A customized coinertia analysis. The `g_adegs2` object is an improvement to the default `g_coi1` one. Each graphical parameter can be customized before, during and/or after the object creation and the "ADEgS" building becomes easy. This figure helps interpret the relationships between plant species and environmental parameters. For example, *Schoenoplectus Litoralis* (`Scli`) is found preferentially in sites with high clay and potassium rates. On the opposite, *Plantago coronopus* (`Plco`) is found in sites with high levels of sodium ions and high conductivity. *Cynosurus polybracteatus* (`Cypo`) and *Chrozophora tinctoria* (`Chti`) are found in sandy sites with high elevation above sea level. The discrepancy between plant and environmental parameters in the 7 ecoregions is shown in the left graph. For each ecoregion, the coordinates of sites are linked to their gravity center, forming an irregular star. The gravity centers of site coordinates from the plants table and from the environmental parameters table are linked by an arrow. The length of this arrow is therefore representative of the discrepancy between plants and environmental parameters. Regions `C1` and `C6` have to the longest arrows, which means that the discrepancy is higher in these three ecoregions.

Conclusions

The **adegraphics** package is a complete reimplementation of **ade4** graphical functionalities with large improvements. The structure of this new package is quite rigorous (formalism provided by S4 classes) but very flexible and easily extendable to support new features. The graphical engine of the **lattice** package allows to produce high quality graphs for data exploration and publication. This **lattice**-based implementation provides an efficient and highly-customizable object-based environment to build graphs and it also facilitates the connections between packages. The "ADEgS" class allows to integrate simple "trellis" objects from the **lattice** package and thus offers the possibility to integrate, in the same graphical object, various graphs created by different packages.

This code is executed with R 3.4.2, **ade4** 1.7-8 and **adegraphics** 1.0-8. Other examples are available in the vignette and in the help pages of the package.

Bibliography

R. S. Bivand and G. Piras. Comparing Implementations of Estimation Methods for Spatial Econometrics. *Journal of Statistical Software*, 63(18):1–36, 2015. URL https://doi.org/10.18637/jss.v063.i18. [p202]

R. S. Bivand, J. Hauke, and T. Kossowski. Computing the Jacobian in Gaussian Spatial Autoregressive Models: An Illustrated Comparison of Available Methods. *Geographical Analysis*, 45(2):150–179, 2013a. URL https://doi.org/10.1111/gean.12008. [p202]

R. S. Bivand, E. J. Pebesma, and V. Gomez-Rubio. *Applied Spatial Data Analysis with R, Second edition*. Springer, NY, 2013b. URL https://doi.org/10.1007/978-1-4614-7618-4. [p202]

G. de Bélair and M. Bencheikh-Lehocine. Composition et déterminisme de la végétation d'une plaine côtière marécageuse : La Mafragh (Annaba, Algérie). *Bulletin d'écologie*, 18(4):393–407, 1987. [p199, 204]

S. Dray and A.-B. Dufour. The ade4 Package: Implementing the Duality Diagram for Ecologists. *Journal of Statistical Software*, 22(4):1–20, 2007. URL https://doi.org/10.18637/jss.v022.i04. [p198]

S. Lê, J. Josse, and F. Husson. FactoMineR: An R Package for Multivariate Analysis. *Journal of Statistical Software*, 25(1):1–18, 2008. URL https://doi.org/10.18637/jss.v025.i01. [p198]

P. Murrell. *R Graphics*. Chapman & Hall / CRC Press, 2005. URL http://www.e-reading.club/bookreader.php/137370/C486x_APPb.pdf. [p198]

E. Neuwirth. *RColorBrewer: ColorBrewer Palettes*, 2014. URL https://CRAN.R-project.org/package=RColorBrewer. R package version 1.1-2. [p202]

J. Oksanen, F. G. Blanchet, M. Friendly, R. Kindt, P. Legendre, D. McGlinn, P. R. Minchin, R. B. O'Hara, G. L. Simpson, P. Solymos, M. H. H. Stevens, E. Szoecs, and H. Wagner. *vegan: Community Ecology Package*, 2017. URL https://CRAN.R-project.org/package=vegan. R package version 2.4-5. [p198]

S. Pavoine, E. Vela, S. Gachet, G. de Bélair, and M. B. Bonsall. Linking patterns in phylogeny, traits, abiotic variables and space: a novel approach to linking environmental filtering and plant community assembly. *Journal of Ecology*, 99(1):165–175, 2011. URL https://doi.org/10.1111/j.1365-2745.2010.01743.x. [p199]

E. J. Pebesma and R. S. Bivand. Classes and methods for spatial data in R. *R News*, 5(2):9–13, 2005. URL https://www.r-project.org/doc/Rnews/Rnews_2005-2.pdf. [p202]

D. Sarkar. *Lattice: Multivariate Data Visualization with R*. Springer, New York, 2008. URL https://doi.org/10.1007/978-0-387-75969-2. [p198]

W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S*. Springer, New York, fourth edition, 2002. URL https://doi.org/10.1007/978-0-387-21706-2. [p198]

H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer, New York, second edition, 2016. URL https://doi.org/10.1007/978-3-319-24277-4. [p203]

Aurélie Siberchicot
Univ Lyon, Université Claude Bernard Lyon 1, CNRS,
Laboratoire de Biométrie et Biologie Evolutive,

F-69100, Villeurbanne, France.
ORCiD: 0000-0002-7638-8318
aurelie.siberchicot@univ-lyon1.fr

Alice Julien-Laferrière
Univ Lyon, Université Claude Bernard Lyon 1, CNRS,
Laboratoire de Biométrie et Biologie Evolutive,
F-69100, Villeurbanne, France.
alice.julien.laferriere@gmail.com

Anne-Béatrice Dufour
Univ Lyon, Université Claude Bernard Lyon 1, CNRS,
Laboratoire de Biométrie et Biologie Evolutive,
F-69100, Villeurbanne, France.
ORCiD: 0000-0002-9339-4293
anne-beatrice.dufour@univ-lyon1.fr

Jean Thioulouse
Univ Lyon, Université Claude Bernard Lyon 1, CNRS,
Laboratoire de Biométrie et Biologie Evolutive,
F-69100, Villeurbanne, France.
ORCiD: 0000-0001-7664-0598
jean.thioulouse@univ-lyon1.fr

Stéphane Dray
Univ Lyon, Université Claude Bernard Lyon 1, CNRS,
Laboratoire de Biométrie et Biologie Evolutive,
F-69100, Villeurbanne, France.
ORCiD: 0000-0003-0153-1105
stephane.dray@univ-lyon1.fr

carx: an R Package to Estimate Censored Autoregressive Time Series with Exogenous Covariates

by Chao Wang and Kung-Sik Chan

Abstract We implement in the R package **carx** a novel and computationally efficient quasi-likelihood method for estimating a censored autoregressive model with exogenous covariates. The proposed quasi-likelihood method reduces to maximum likelihood estimation in absence of censoring. The **carx** package contains many useful functions for practical data analysis with censored stochastic regression, including functions for outlier detection, model diagnostics, and prediction with censored time series data. We illustrate the capabilities of the **carx** package with simulations and an elaborate real data analysis.

Introduction

Censored data are frequently encountered in diverse fields including environmental monitoring, medicine, economics, and social sciences. Censoring may arise, for example, when a measuring device is subject to some detection limits beyond which the device cannot yield a reliable measurement. Censoring can also occur due to regulations on price change, e.g., limits on maximal intra-daily price change in a stock market.

There exists an extensive literature on regression analysis with censored responses since the pioneering work of Buckley and James (1979). Considerable efforts have also been spent implementing existing methods for estimating various models with censored observations, many of which have been implemented in R. For instance, Henningsen (2010) introduced the **censReg** package (Henningsen, 2013), which covers standard regression models with censored responses including the standard Tobit model (Tobin, 1958), maximum likelihood estimation with cross-sectional data, and random-effects maximum likelihood procedure for panel-data using Gauss-Hermite quadrature. The Tobit model is also implemented in other packages with possibly different estimation methods, including `tobit()` in **AER** (Kleiber and Zeileis, 2008), `cenmle()` in **NADA** (Lee, 2013), `tobit()` in **VGAM** (Yee, 2015), `MCMCtobit()` in **MCMCpack** (Martin et al., 2011), etc.

While there exists an extensive literature on estimating regression models with censored responses and associated software, there are few studies with censored time series response data. More generally, the problem of stochastic regression with both the response and covariates being possibly censored is relatively under-explored. Zeger and Brookmeyer (1986) studied maximum likelihood estimation of a regression model with the errors driven by an autoregressive model of known order $p \geq 0$ ($AR(p)$). Owing to censoring, the "state" vector is generally of variable dimension which can increase rapidly with increasing censoring and AR order. Thus, the maximum likelihood estimation becomes quickly numerically intractable with increasing censoring even for moderately high AR order (Wang and Chan, 2017a). Zeger and Brookmeyer (1986) also briefly discussed a pseudo-likelihood approach but did not further develop it. Park et al. (2007) proposed an imputation method to estimate a censored autoregressive moving average (ARMA) process. Their method imputes each censored value by some random value simulated from their conditional distribution given the observed data and the censoring information, and treats the imputed time series as the complete data with which estimation can be done by any standard method. However, they focused on the AR(1) model and relied on simulation studies to demonstrate their method, with no derivation of theoretical properties.

In term of publicly available R packages facilitating estimation with censored time series data, we are aware of only three such packages to date, namely, **cents** (McLeod et al., 2014), **ARCensReg** (Schumacher et al., 2016), and our **carx** (Wang and Chan, 2017b). The **cents** package includes the `fitcar1()` function, for fitting an AR(1) model in the presence of censored and/or missing data, and the `cenarma()` function which, according to the authors, implements a quasi-EM algorithm whose M-step is carried out by the `arima()` function and the E-step via the Durbin-Levinson recursions. However, there is little documentation about these functions, rendering it hard to understand and use the **cents** package. The **ARCensReg** package offers similar functionality as our **carx** package. But their estimation is implemented via a stochastic approximation version of the EM (SAEM), which is different from our approach. In addition, it seems to be developed after the **carx**, as a dataset in **carx** is included in **ARCensReg**.

Motivated by the need for developing a computationally efficient method for estimating censored stochastic regression models, Wang and Chan (2017a) have recently introduced such a method for

censored autoregressions with exogenous covariates (CARX). The basic idea of our new approach assumes that the score of the complete-data conditional log-likelihood of Y_t^* (the uncensored counterpart of Y_t) given $Y_{t-j}^*, j = 1, \ldots, p$ (and the covariates) has a closed-form expression and so does its expectation given the possibly censored time series $Y_{t-j}, j = 0, \ldots, p$, evaluated at the same set of model parameters. Setting the preceding conditional mean score to zero then provides an unbiased estimating equation for estimating the model. The proposed method reduces to maximum likelihood estimation in the absence of censoring, hence it is referred to as quasi-likelihood estimation. Furthermore, the consistency and asymptotic normality of the quasi-likelihood estimator have been established under some mild regularity conditions (Wang and Chan, 2017a).

In this paper we aim to introduce the R package **carx**, in which quasi-likelihood estimation of a CARX model is implemented for the important special case of normal innovations. The main functionality of the package is to provide an intuitive interface with comprehensive documentation to enable the user to estimate the parameters of a CARX model. In addition, some utility functions for model summary, model diagnostics, outlier detection, and prediction with censored time series data are also included in the package.

In addition, we have also implemented a new object class for censored time series, i.e., `"cenTS"`. The `"cenTS"` class inherits the extensible time series class `"xts"` in the R package **xts** (Ryan and Ulrich, 2017). Some functionalities, including plotting and summary, for the `"cenTS"` class have been implemented. The `"cenTS"` class is expected to be extended in future and is hoped to be used as a standard data structure for censored time series data.

In the following sections we first elaborate the CARX model and review the quasi-likelihood estimation method, then present the functionality and main functions of the R package **carx** and illustrate the package with data analyses using both simulated and real data examples. Some simulation studies assessing the empirical performance of model selection by minimizing the AIC and the accuracy of the proposed forecasting method and real data example are also reported.

The CARX model

In this section we briefly review the quasi-likelihood method for estimating a CARX model, and refer the reader to Wang and Chan (2017a) for details and some theoretical properties of the estimator. We first formulate the problem by specifying the model, then outline the estimation method and discuss some specific topics including model prediction, model diagnostics, and outlier detection.

Model specification

Let $\{Y_t^*\}_{t=0}^{\infty}$ denote a real-valued time series of interest with Y_t^* being not observable if it falls inside a censoring region $C_t \subset R$ which may be time-varying. The censoring region C_t is generally an interval of the form $(-\infty, l_t)$, (u_t, ∞), or (l_t, u_t) corresponding to left, right, and interval censoring, respectively (Huang and Rossini, 1997; Park et al., 2007). (Left and/or right censoring is allowed by **carx** but interval censoring is not yet implemented in **carx**.) In practice, when an observation is censored, it is often recorded as the nearest censoring limit, as it is typically known whether it is left or right censored. The **carx** package assumes the censoring limits to be independent of the underlying process, and automatically treats any missing data as resulting from left censoring with their corresponding censoring limit $l = \infty$.

In practice, Y_t^* is often found to be correlated with some vector covariate, say, X_t. We assume a linear regression relationship between Y_t^* and X_t, with the regression errors following an $AR(p)$ model, where p is the AR order.

The Censored Auto-Regressive model with eXgenous variables (CARX) specifies that the uncensored response $\{Y_t^*\}$ is an autoregressive (AR) process given by

$$\left(Y_t^* - X_t^\intercal \beta\right) - \sum_{j=1}^{p} \psi_j \left(Y_{t-j}^* - X_{t-j}^\intercal \beta\right) = \varepsilon_t, \tag{1}$$

and Y_t^*'s are linked to the observations as follows

$$Y_t = \begin{cases} l_t, & \text{if } Y_t^* \text{ is left censored,} \\ u_t, & \text{if } Y_t^* \text{ is right censored,} \\ Y_t^*, & \text{otherwise,} \end{cases} \tag{2}$$

where β is the vector of regression coefficients, $\psi_i, i = 1, 2, \ldots, p$, are the AR parameters, $\{\varepsilon_t\}$ is an independent and identically distributed (iid) process with mean 0, variance σ_ε^2, and independent of

$\{X_t\}$.

The ε_t's are also known as the innovations in the time series literature. Eqn. (1) is equivalent to the regression model $Y_t^* = X_t^\intercal \beta + \eta_t$, where the regression errors η_t are correlated over time and follow an AR(p) process with the ψ's being the AR coefficients. In the package, the innovations are assumed to be normal although it is shown by Wang and Chan (2017a) that the proposed estimation method is robust to mild departure from the normality assumption.

Parameter estimation

Let $\psi = (\psi_1, \cdots, \psi_p)^\intercal$. Throughout, $\theta = (\beta^\intercal, \psi^\intercal, \sigma_\varepsilon)^\intercal$ denotes a generic parameter vector, while θ_0 denotes the true parameter vector. Let $\{(Y_t, X_t)\}_{t=1}^n$ be data generated from the CARX model with parameter θ_0. The quasi-likelihood estimation procedure is motivated by maximum likelihood estimation and leverages on (i) the availability of the closed-form expression of $\ell_t(\theta) = \ell\left(Y_t^* | Y_{t-j}^*, j = 1, \ldots, p, X_{t-k}, k = 0, \ldots, p; \theta\right)$ (which holds, for instance, for the case of normal errors as implemented in **carx**) and (ii) $\sum_{t=p+1}^n S_t(\theta) = 0$ is an unbiased estimating equation, where $S_t(\theta)$ is the first derivative of $\ell_t(\theta)$ with respect to θ. Since Y_t^* are unobservable, we replace $S_t(\theta)$ by $E_\theta\left(S_t(\theta) | Y_{t-k}, X_{t-k}, k = 0, \ldots, p\right)$ resulting in the following estimating equation:

$$\sum_{t=p+1}^{n} E_\theta\left(S_t(\theta) | Y_{t-k}, X_{t-k}, k = 0, \ldots, p\right) = 0. \tag{3}$$

The quasi-likelihood method estimates θ by solving Eq (3). Note that, in the absence of censoring, solving the preceding estimating equation reduces to maximum likelihood estimation, asymptotically.

The following iterative scheme for solving Eq (3) was proposed by Wang and Chan (2017a).

Step(1) Initialize the parameter estimate by some consistent estimate, denoted by $\theta^{(0)}$.

Step(2) For each $k = 1, \ldots,$ obtain an update of estimate $\theta^{(k)}$ by

$$\theta^{(k)} = \text{argmax}_\theta\, Q\left(\theta | \theta^{(k-1)}\right), \tag{4}$$

where

$$Q\left(\theta | \theta^{(k-1)}\right) = \sum_{t=p+1}^{n} Q_t\left(\theta | \theta^{(k-1)}\right), \tag{5}$$

$$Q_t\left(\theta | \theta^{(k-1)}\right) = \mathrm{E}_{\theta^{(k-1)}}\left(\ell_t(\theta) | Y_{t-k}, X_{t-k}, k = 0, \ldots, p\right). \tag{6}$$

Step(3) Iterate Step (2) until $\|\theta^{(k)} - \theta^{(k-1)}\|_2 / \|\theta^{(k-1)}\|_2 < \epsilon$ for some positive tolerance $\epsilon \approx 0$. Let $\hat{\theta}$ be the estimate obtained from the last iteration.

The optimization in Step (2) for the case of normal innovations is elaborated in Section 2.4 of Wang and Chan (2017a). The value $Q(\hat{\theta}|\hat{\theta})$ evaluated at the convergence of the algorithm will be referred to as the maximum (quasi-)log-likelihood. In the absence of censoring, it reduces to the maximum log-likelihood, hence it will be used to replace the latter in evaluating information criteria such as the Akaike information criterion (AIC) (Konishi and Kitagawa, 2008).

In the **carx** package, the initial value for the preceding iterative algorithm is set to the conditional least squares estimate obtained with the censored data replaced by the corresponding censoring limit, which appears to work well in simulation examples reported in Wang and Chan (2017a).

Wang and Chan (2017a) proved the consistency and asymptotic normality of the quasi-likelihood estimator under mild regularity conditions. But the asymptotic covariance matrix of the estimator involves two intractable matrices. Consequently, Wang and Chan (2017a) proposed to use parametric bootstrap for drawing inference, including estimating the asymptotic covariance matrix and constructing confidence intervals of the unknown parameters.

Model prediction

It is of practical interest to predict the future values Y_{n+h}^* given the observations $\{(Y_t, X_t)\}_{t=1}^n$, where $h = 1, 2, \ldots, H$ and H is some fixed upper bound, for instance, $H = 14$ for bi-weekly forecast, assuming the data are sampled daily. This is generally a non-trivial problem in the presence of censoring, and can be handled by Monte Carlo simulation for its solution. Since X is an exogenous process, we consider the simple case of the prediction problem conditioned on the given future covariate values $\{X_{t+h}\}_{h=1}^H$.

We also assume normality of ε_t and known parameter θ_0, although the following discussion can be readily extended to non-normal innovations. Relaxation of these assumptions will be discussed at the end of this subsection. The prediction problem is equivalent to finding the conditional distribution

$$\begin{aligned}\mathcal{D}_{n,h} &= \mathcal{D}\left(Y^*_{n+h} | \{X_{n+i}\}_{i=1}^h, \{(Y_t, X_t)\}_{t=1}^n\right) \\ &= \mathcal{D}\left(Y^*_{n+h} | \{X_{n+i}\}_{i=1}^h, \{(Y_t, X_t)\}_{t=\tau}^n\right),\end{aligned}$$

where $\tau = \max\left(\{1\} \cup \left\{u : 1 \le u \le n-p+1, \text{ and none of } \{Y_t\}_{t=u}^{u+p-1} \text{ is censored}\right\}\right)$, due to the autoregressive nature of the regression errors $\eta_t = Y^*_t - X_t^\intercal \beta$ (Zeger and Brookmeyer, 1986).

There are two cases. Case 1: $\tau = n-p+1$, i.e., the most recent p Y_t's are uncensored so that the prediction problem admits a closed-form solution which is well-known; see, e.g., Cryer and Chan (2008, Chapter 9). Specifically, for any $h = 1, \ldots, H$, $\mathcal{D}_{n,h}$ is a normal distribution whose mean serves as the point predictor denoted by $\hat{Y}^*_{n+h}$ that can be recursively computed as follows: $\hat{Y}^*_{n+h} = X_{n+h}^\intercal \beta + \hat{\eta}_{n+h}$, with $\hat{\eta}_t = \sum_{l=1}^p \psi_l \hat{\eta}_{t-l}$ for $t > n$, and $\hat{\eta}_t = Y_t - X_t^\intercal \beta$ if $t \le n$. The prediction error, denoted by $\epsilon_{n+h} = Y_{n+h} - \hat{Y}^*_{n+h}$, can be written as $\epsilon_{n+h} = \varepsilon_{n+h} + \sum_{l=1}^p \psi_l \epsilon_{n+h-l} = \sum_{i=0}^h \omega_{h,i} \varepsilon_{n+h-i}$, where the coefficients $\omega_{h,i}$ can be recursively calculated by making use of the preceding identity and the initial condition $\omega_{h,0} = 1$. The prediction variance is given by $\text{var}(\epsilon_{n+h}) = {\sigma_\varepsilon}^2 \sum_{i=0}^h \omega_{h,i}^2$.

We now consider Case 2: $\tau < n-p+1$. Then $\mathcal{D}_{n,h}$ is a truncated multivariate normal distribution. Although the first and second moments of $\mathcal{D}_{n,h}$ admit closed-form solutions (Tallis, 1961; Genz et al., 2017), they are not useful for constructing predictive intervals as the predictive distributions are non-normal. Thus, we propose to use a sampling approach to estimate any interesting characteristic of the predictive distribution of Y^*_{n+h}. First, note that the regression errors $\{\eta_t = Y^*_t - X_t^\intercal \beta\}_{t=\tau}^n$ are jointly normal. Let η_c and η_o be the sub-vectors of $\eta_{\tau:n}$ such that the corresponding elements of $Y_{\tau:n}$ are censored and observed, respectively. Then given $\{(Y_t, X_t)\}_{t=\tau}^n$, η_c follows a truncated multivariate normal distribution, whose realizations can be readily simulated, and hence we can simulate $Y^*_t = X_t^\intercal \beta + \eta_t, \tau \le t \le n$. Then the realizations of $Y^*_{n+h}, h = 1, \ldots, H$ can be drawn from the multivariate normal predictive distribution stated in Case 1. Predictive intervals of Y^*_{n+h} can then be approximately constructed from a random sample from the predictive distribution of Y^*_{n+h}, using the percentile method.

Note that the proposed predictive scheme is conditional on the future covariate values $\{X_t\}_{h=1}^H$, which, in general, are non-deterministic. Extension to the case of stochastic $\{X_{t+h}\}_{h=1}^H$ is straightforward, provided that its stochastic generating mechanism is known, as drawing a realization from the predictive distribution of Y^*_{n+h} can be done in two steps. Step 1 consists of drawing a realization $\{x_{t+h}\}_{h=1}^H$, followed by drawing a future realization for Y^*_{n+h} given the data and $\{x_{t+h}\}_{h=1}^H$. In practice, θ_0 is unknown and it can be replaced by the quasi-likelihood estimator or a parametric bootstrap approach which can be readily implemented to incorporate parametric uncertainty in the prediction.

Model diagnostics

A main task in model diagnostics consists of checking whether or not the data are consistent with the model assumption that the innovations are independent and identically normally distributed of zero mean and constant variance. In the presence of censoring, how to define the residuals is unclear. For the simple case when Y^*_t is observed, the corresponding residual is universally defined as $Y^*_t - \hat{Y}^*_{t|t-1}$, where $\hat{Y}^*_{t|t-1}$ is the mean of $\mathcal{D}_{t-1,1}$, evaluated at the parameter estimate. In the case of censoring so that some Y^*_ts are unobserved, Wang and Chan (2017a) advocated the use of the simulated residuals (Gourieroux et al., 1987) for model diagnostics. The simulated residuals are constructed as follows. First, impute each unobserved Y^*_t by a realization from the conditional distribution $\mathcal{D}\left(Y^*_t | \{(Y_s, X_s)\}_{s=1}^t\right)$, evaluated at the parameter estimate. Then, refit the model with $\{(Y^*_t, X_t)\}$ so obtained, via conditional maximum likelihood; the residuals from the latter model are the simulated residuals $\hat{\varepsilon}_t$. Let the corresponding parameter estimate of θ be $\tilde{\theta}$. The corresponding (simulated) partial residuals for the X's, i.e., $X_t^\intercal \tilde{\beta} + \hat{\varepsilon}_t$, can be used to assess the relationship between Y and X, after adjusting for the autoregressive errors.

A simulation study reported in Wang and Chan (2017a) suggests that the asymptotic null distribution of the Ljung-Box test statistic, for checking residual autocorrelations, based on the simulated residuals is the same as that based on the uncensored data. Thus, standard diagnostic tools for residual analysis may be applicable with the simulated residuals.

Outlier detection

Real data are often marred by outliers. An outlier in a time series may result from a perturbation inducing an unknown shift in an observation or an innovation, resulting in the so-called additive or innovative outlier, respectively (Cryer and Chan, 2008). An innovative outlier (IO) may mask as a contiguous block of additive outliers (AO). Since it is harder to detect IOs in censored time series, we focus on detecting AOs with a new method for doing so in censored time series.

As the number of outliers and their locations are generally unknown, outlier detection is carried out one by one and iteratively. The procedure begins with an outlier-free CARX model. Then we check for the presence of additive outliers by a method to be described below. If an outlier is detected at time t_o, the covariate X will be augmented with the indicator variable I_{t_o} which equals 1 if $t = t_o$, and 0 otherwise. The augmented CARX model is then fitted, with which outlier detection is repeated until no more outliers are found.

More specifically, we describe a method to detect any remaining additive outliers given the data and a CARX model. For the sake of fast computation, we consider the predictive distribution of Y_t given the information from $t-p$ to t, i.e., $\tilde{\mathcal{D}}_t := \mathcal{D}\left(Y_t^* | X_t, \left\{\left(Y_{t-j}, X_{t-j}\right), j = 1, \ldots, p\right\}\right)$. Let $P_{\tilde{\mathcal{D}}_t}(E)$ be the probability of the event E evaluated with distribution $\tilde{\mathcal{D}}_t$ and n the sample size, for each $t = p+1, \cdots, n$, we calculate the following probability p_t.

$$p_t = \begin{cases} P_{\tilde{\mathcal{D}}_t}(Y_t > u_t), & \text{if } Y_t^* \text{ is right censored,} \\ P_{\tilde{\mathcal{D}}_t}(Y_t < l_t), & \text{if } Y_t^* \text{ is left censored,} \\ \min\{P_{\tilde{\mathcal{D}}_t}(Y_t > y_t), P_{\tilde{\mathcal{D}}_t}(Y_t < y_t)\}, & \text{otherwise.} \end{cases}$$

Let $t_o = \text{argmin}_{t=1,\cdots,n} p_t$, and the response at t_o is declared as an AO if $p_{t_o} < 0.025/n$, where the Bonferroni inequality is used to limit the family error rate to not exceed 5% (Cryer and Chan, 2008); otherwise, it is deemed that there are no remaining outliers.

The carx package

In this section we present the R package **carx** in which the estimation, prediction, and diagnostics procedures discussed in previous section are implemented, assuming the normality of ε_t. For more detail, see the documentation of the package. Examples will be given in the next section.

A class for censored time series

First, let us introduce a class `"cenTS"` designed to encapsulate a censored time series with its observed values as well as the left (lower) and right (upper) censoring limits. The `"cenTS"` inherits the extensible time series class `"xts"` in the R package **xts**. A `"cenTS"` object can be constructed by the following function call.

```
cenTS(value, order.by, lcl = NULL, ucl= NULL, value.name = "value", ...)
```

Note that the `value` (whose name can be set in `value.name`) and `order.by` denote the observed values and their corresponding indices respectively, and `lcl` and `ucl` denote the left (lower) and right (upper) censoring limits respectively. Other time series variables to be included as covariates in the regression can be supplied via additional arguments.

A `"cenTS"` object can be inspected by the `print()` and `plot()` methods. Any covariate time series can be retrieved by the `xreg()` method.

The default estimation method

The foremost function is the method for the S3 class `"carx"`, `carx()`, whose signature is the following.

```
    carx(formula, data = list(),  p = 1,
prmtrX = NULL, prmtrAR = NULL, sigma = NULL,
y.na.action = c("skip","as.censored"), addMu = TRUE,
tol = 1e-4, max.iter = 500,
CI.compute = FALSE, CI.level = 0.95,
b = 1000,b.robust = FALSE,b.show.progress = FALSE,
init.method = c("biased","consistent"),
cenTS = NULL, verbose = FALSE,...)
```

The carx() method provides a simple-to-use interface for the user to input a formula, a data set, and other arguments to estimate a CARX model.

The carx() method returns a "carx" object which stores the supplied data, the quasi-likelihood coefficient estimates, as well as other information. It allows many optional arguments to control the function behavior. The main arguments are listed below:

- formula is a formula representing the regression part of the model, such as y ~ x1 + x2.
- data denotes a data.frame which includes the following:
  - The response variable with variable name identified by the supplied formula.
  - Any covariate(s) with variable name(s) identified by the supplied formula.
  - A vector with name ci whose components take values from $\{-1, 0, 1\}$, where -1 (0,1) indicates that the corresponding element in the response variable is left-censored (not censored, right censored).
  - lcl representing the vector of left (lower) censoring limits. If not present, indicating no lower limit.
  - ucl representing the vector of right (upper) censoring limits. If not present, indicating no upper limit.
- p denotes the autoregressive order of the regression errors, default = 1.

The above arguments supply the data structure including the censoring information, and specify the CARX model to be estimated. Although the function contains many optional arguments for fine-tuning the fitting algorithm and obtaining more information about the fitted model such as confidence intervals, we merely discuss the following two arguments:

- prmtrX, prmtrAR, and sigma are used to specify the initial values of the regression coefficients β, the autoregressive parameters Ψ, and the innovation standard deviation σ_ε, respectively.
- y.na.action is a string indicating how to handle missing (NA) values in y. If it is set to "skip" (default), cases containing a missing value will be skipped, so that the estimating equation of future cases will be conditional on the most recent p complete cases after the skipped case. For "as.censored", the y value will be treated as left-censored with the left (lower) censoring limit replaced by positive infinity. The user may choose to use skip if there exist few long gaps in the response. Use "as.censored" in the presence of numerous, non-contiguous missing values in y. Note that the presence of any missing values in x will automatically hard-code y.na.action to be "skip".

Other methods

As "carx" is an S3 class, some generic methods have been implemented so that the estimation function can be easily called for practical use and more information about the model fitting can be easily extracted.

The function print() simply returns a plain output of the fitted model, while the summary() function provides a more elaborate summary of the fitted model including the estimates, their standard errors, 95% confidence limits and p-values, based on parametric bootstrap, for each model parameter, if CI.compute = TRUE. The model parameters can be conveniently extracted by the function coef(), which returns all coefficient estimates except that the error (innovation) standard deviation is returned as the sigma component of the list returned by carx(). logLik() returns the maximum (quasi-)log-likelihood $\sum_{t=p+1}^{n} \mathrm{E}_{\hat{\theta}} \left[\ell \left(Y_t^* | \mathcal{F}_t^*; \hat{\theta} \right) | \mathcal{G}_t \right]$, which can be used in lieu of the intractable maximum log-likelihood. For instance, the function AIC() computes the AIC of the model with the maximum log-likelihood replaced by maximum (quasi-)log-likelihood.

There are some other useful functions in the package. The method plot() draws the time plot of the censored response time series, superimposed with the fitted values (1-step-ahead predictors) from the supplied CARX model. The function predict() computes the multi-step-ahead point predictors and their associated prediction limits, based on a given model and future values of the covariates supplied by the user. The function fitted() returns the fitted values by calling the predict method. The function residuals() returns the simulated residuals of the fitted model. The outlier detection method discussed in Section 32.2.5 is implemented by the method outlier(). Model diagnostics based on the simulated residuals are visualized by the method tsdiag(), which consists of four subplots: the time plot of standardized simulated residuals, the residuals versus fitted values plot, the residual autocorrelation function plot, and the plot of the p-values of the Ljung-Box test statistics, for testing no residual autocorrelations.

Using the package

In this section we illustrate the various functions of the package through two examples, the first one is a simulated data set and the second a real data set. Note that an extensive simulation study about the performance of the proposed estimation method and some model diagnostics can be found in Wang and Chan (2017a) which shows the robustness of the proposed estimation method to slight departure from the normality assumption of the innovations. We first load the `carx` package by the following command.

```
> library(carx)
```

A function to simulate data

To begin, we introduce the function `carxSimCenTS()` for simulating data from a CARX model, whose signature and default values of arguments are shown below.

```
carxSimCenTS(nObs = 200, prmtrAR = c(-0.28,0.25),
  prmtrX = c(0.2,0.4), sigma = 0.60, lcl = -1, ucl = 1, x = NULL,
  seed = NULL, value.name = 'y', end.date = Sys.date())
```

The `carxSimCenTS()` function generates a simulated `"cenTS"` time series of length `nObs`, with the AR parameters $(\psi_i, i = 1, \ldots, p)$ supplied through the argument `prmtrAR`, the regression coefficients through `prmtrX`, and innovation standard deviation through `sigma`, the lower and upper censoring limits through `lcl` and `ucl` respectively. The regressors can be supplied via `x`, which, if is `NULL`, will be generated as independent standard normal variables. The user can also specify the seed of the random number generator by `seed` for ensuring repeatability. As `carxSimCenTS()` encapsulates the simulated data into a `"cenTS"` object, the construction of which need a time/date-based index. The default treats the data as daily observations, with the end date specified by `end.date`. The user can set the name of the censored time series via `value.name` but the names of the regressors are hard-coded as `X1`, `X2`, etc. There is another function `carxSim()`, which returns a `data.frame` consisting of `y`, `x`, `lcl`, `ucl` and `ci`. We will mainly use the `carxSimCenTS()` function for simulation as it encapsulates the data as a `"cenTS"` object.

A step-by-step illustration with a simulated series

We first simulate a `"cenTS"` series, using the `carxSimCenTS()` function with essentially the default setting, i.e., simulate interval-censored data from a regression model with a 2-dimensional covariate comprising independent standard normal components whose regression coefficients are 0.2 and 0.4, and AR(2) regression noise terms with the AR coefficients being $\psi_1 = -0.28, \psi_2 = 0.25$; the data are then censored unless they fall inside the interval $(-1, 1)$.

```
> datSim <- carxSimCenTS(seed = 0,end.date = as.Date('2015-08-01'))
```

A glimpse of the last few data cases of the series is instructive.

```
> tail(datSim)
```

```
                y lcl ucl ci      X1     X2
2015-07-26  1.000  -1   1  1 -0.5466  0.288
2015-07-27 -0.321  -1   1  0 -1.6887 -1.505
2015-07-28 -0.259  -1   1  0 -1.5724  1.519
2015-07-29  0.386  -1   1  0 -0.4050  0.367
2015-07-30  0.282  -1   1  0  0.3193  1.700
2015-07-31  0.181  -1   1  0  0.0404  0.644

Censoring rate: 0.205
```

The simulated series can be readily visualized using the plot function (see Figure 1).

```
> plot(datSim)
```

Then the parameters can be estimated by the `carx()` method, with the fitted model saved in the object named `modelSim`.

```
> modelSim <- carx(y ~ X1 + X2 - 1,data = datSim, p = 2, CI.compute = TRUE)
```

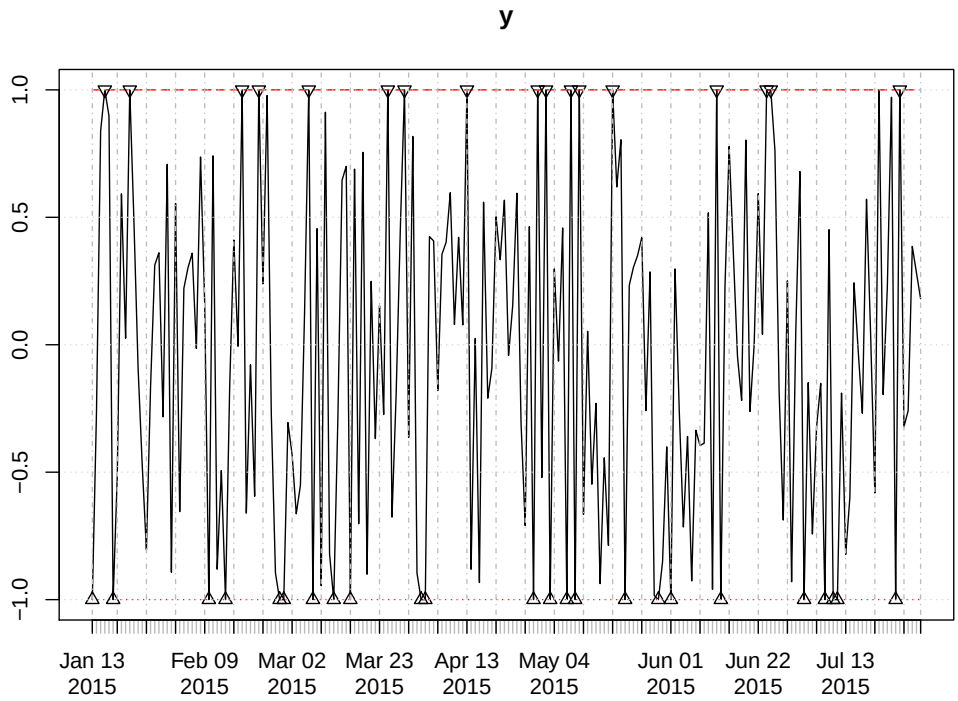


Figure 1: Time plot of the simulated `"cenTS"` series. The observed responses are connected as a solid black line, and the lower and upper censoring limits drawn as red dotted and dashed line with censored observations marked by triangles pointing up and down, respectively.

Note that `-1` in the formula specifies no intercept in the regression. Information about the fitted model can be obtained directly by typing the variable name `modelSim`.

```
> modelSim

Call:
carx.formula(formula = y ~ X1 + X2 - 1, data = datSim, p = 2,
    CI.compute = T)

Coefficients:
    X1     X2    AR1    AR2
 0.203  0.460 -0.234  0.279

Residual (innovation) standard deviation:
[1] 0.548

Censoring rate:
[1] 0.205

Sample size:
[1] 200

Number of parameters:
[1] 5

Quasi-log-likelihood:
[1] 20.1

AIC:
[1] -30.1

Confidence interval:
```

```
       2.50%  97.50%
X1     0.131  0.2766
X2     0.383  0.5446
AR1   -0.390 -0.0919
AR2    0.123  0.4066
sigma  0.483  0.6122

Variance-covariance matrix:
             X1       X2      AR1       AR2    sigma
X1     1.41e-03 1.98e-05 5.86e-05 -1.49e-04 9.95e-05
X2     1.98e-05 1.73e-03 1.28e-04  9.08e-05 2.45e-04
AR1    5.86e-05 1.28e-04 5.76e-03  2.08e-03 1.32e-04
AR2   -1.49e-04 9.08e-05 2.08e-03  5.15e-03 5.88e-05
sigma  9.95e-05 2.45e-04 1.32e-04  5.88e-05 1.05e-03
N.B.: Confidence intervals and variance-covariance matrix
are based on 1000 bootstrap samples.
```

A summary of the fitted model can be obtained by running the `summary()` function.

```
> summary(modelSim)

Call:
carx.formula(formula = y ~ X1 + X2 - 1, data = datSim, p = 2,
    CI.compute = T)


Coefficients:
      Estimate  StdErr lowerCI upperCI p.value
X1      0.2025  0.0375  0.1314    0.28  <2e-16 ***
X2      0.4602  0.0416  0.3826    0.54  <2e-16 ***
AR1    -0.2336  0.0759 -0.3903   -0.09   0.002 **
AR2     0.2792  0.0717  0.1232    0.41  <2e-16 ***
sigma   0.2025  0.0324  0.4834    0.61  <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

AIC:
[1] -30.1
```

Although it can be shown that the quasi-likelihood estimator is asymptotically normal under some regularity conditions (Wang and Chan, 2017a), the asymptotic variance-covariance matrix is intractable so it is computed via parametric bootstrap. The summary function prints out the coefficient estimates and innovation standard deviation estimate, together with their estimated (bootstrap) standard errors, and lower and upper 95% confidence limits. Note that the bootstrap computation time increases almost linearly with the bootstrap replicate size; the default is 1000. More specific information can be easily obtained by invoking various methods. For instance, `logLik` returns the quasi-log-likelihood of the data, `coef` returns the coefficients of the model, and the standard deviation of ε_t can be obtained by `modelSim$sigma`.

```
> logLik(modelSim)

[1] 20.1
attr(,"class")
[1] "logLik.carx"

> coef(modelSim)

    X1     X2    AR1    AR2
 0.203  0.460 -0.234  0.279

> modelSim$sigma

[1] 0.548
```

The `plot()` function provides a visual check of how the fitted values track the data, with the censoring limits superimposed on the diagram, see Figure 2.

```
> plot(modelSim)
```

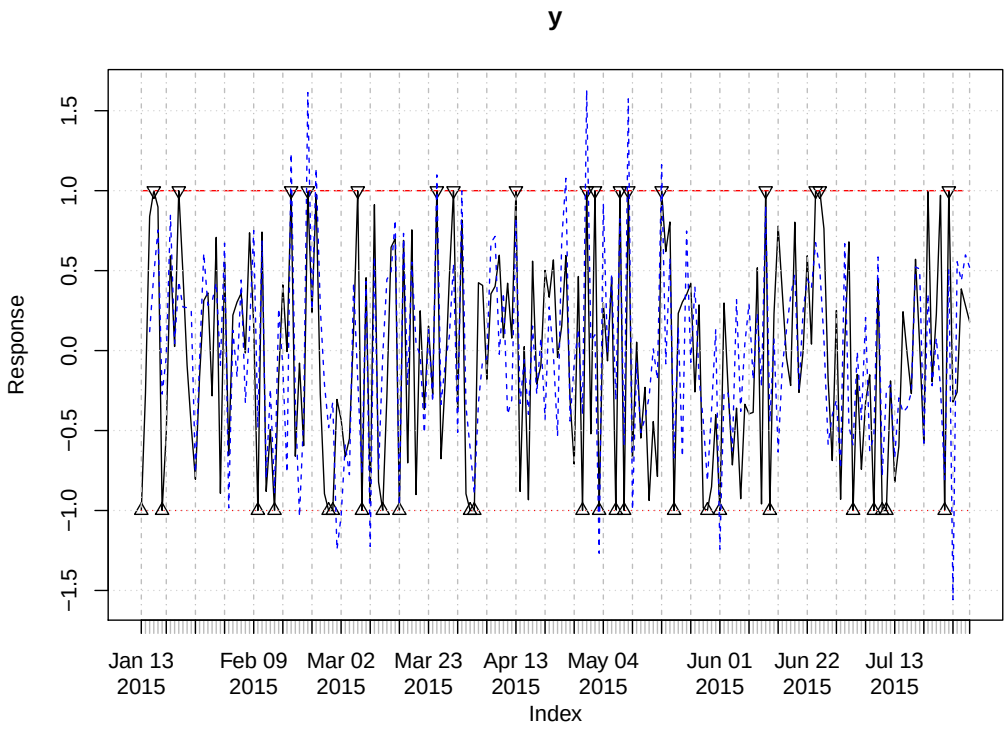


Figure 2: Time plot of the raw data and fitted values from the CARX model. The observed responses are connected as a solid black line, and the lower and upper censoring limits drawn as red dotted and dashed line with censored observations marked by triangles pointing up and down, respectively. The fitted values are connected as a blue dashed line.

Model diagnostics are facilitated by the `tsdiag()` function which is similar to the `tsdiag()` function in **TSA** package (Chan and Ripley, 2012). The `tsdiag()` function generates a plot of 4 sub-figures, namely, the time plot of the simulated residuals which is useful for visually checking the presence of residual temporal patterns and/or outliers, the simulated residuals versus fitted values plot which is useful for checking the adequacy of the linear regression model assumptiom, the residual autocorrelation function (ACF) plot that quantifies the residual correlations, and the plot of the p-values of the Ljung-Box tests for the presence of residual autocorrelation. The following command generates the diagnostics plot for the model fitted to the simulated data.

```
> tsdiag(modelSim)
```

The uppermost diagram in Figure 3 shows no apparent residual temporal patterns, which is also confirmed by the fact that none of the examined residual autocorrelations in the second sub-figure from the bottom are significant and that the bottom sub-figure shows that all p-values of the Ljung-Box test statistics based on the first k lags of residual autocorrelations are larger than 5% for all allowable $k \leq 23$. Moreover, the second sub-figure from the top shows that the linear regression assumption is justifiable and so is the constant innovation variance assumption. Hence, we can conclude that the model is correctly specified, as it should be, and it provides a good fit to the data.

A real data application

In this example we utilize the package to analyze the change of total phosphorus (P) concentrations in river water. Phosphorus is a major nutrient in river water, of which an excessive amount can result in environmental problem such as eutrophication. Phosphorus concentration in many rivers in Iowa has been monitored under the ambient water quality program conducted by the Iowa Department of Natural Resources (Libra et al., 2004). An analysis of the change of P concentration has been reported by Wang et al. (2016). Here we illustrate the analysis for a particular data set from an ambient site located in the West Fork Cedar River at Finchford, with the data available in a `"cenTS"` object named `pts` in the **carx** package.

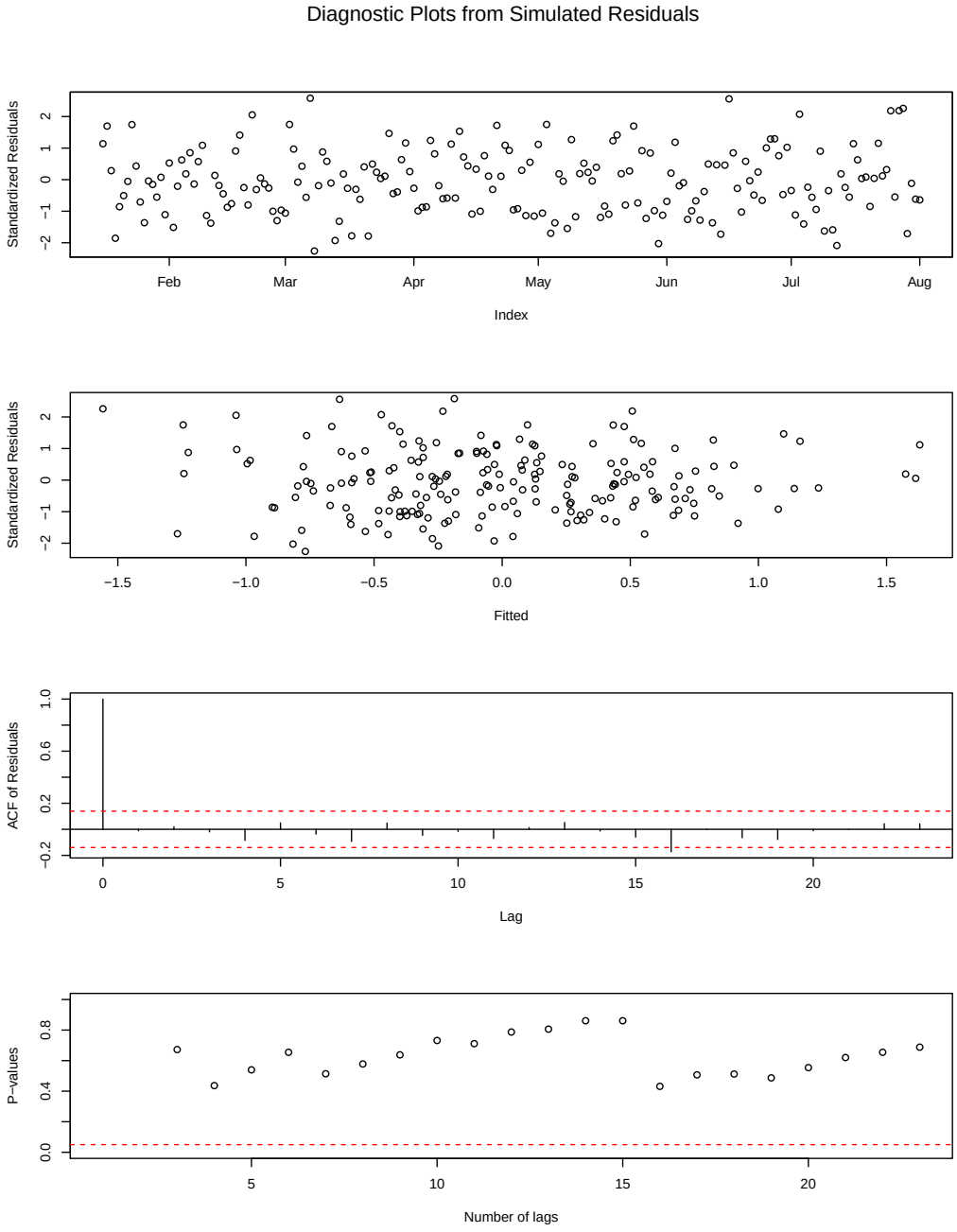


Figure 3: Diagnostic plots based on simulated residuals.

The P concentrations (in mg/l) were left-censored whenever they fell below certain time-varying detection limits, resulting in a censoring rate of 12.6%. The data were collected monthly from October 1998 to October 2013, but data collection was suspended between September 2008 to March 2009, owing to lack of funding. In the data set, there are serveral variables.

```
> names(pts)

[1] "logP"     "lcl"      "ci"       "tInMonth" "logQ"     "season"
```

The variable `logP` consists of the logarithmic P, `lcl` the corresponding censoring limits, `ci` the indicator variable of censoring, `tInMonth` is the time index (in month), `logQ` is the corresponding logarithmic water discharge data (Q) measured in cf/s, obtained from the U.S. Geological Survey, and season indicates to which `season` the month index belongs (see below for further details). P is generally correlated with the water discharge (Schilling et al., 2010). We will explore the relationship between P and Q. See Figure 4 for the time plots of P, Q, and the historical censoring limits.

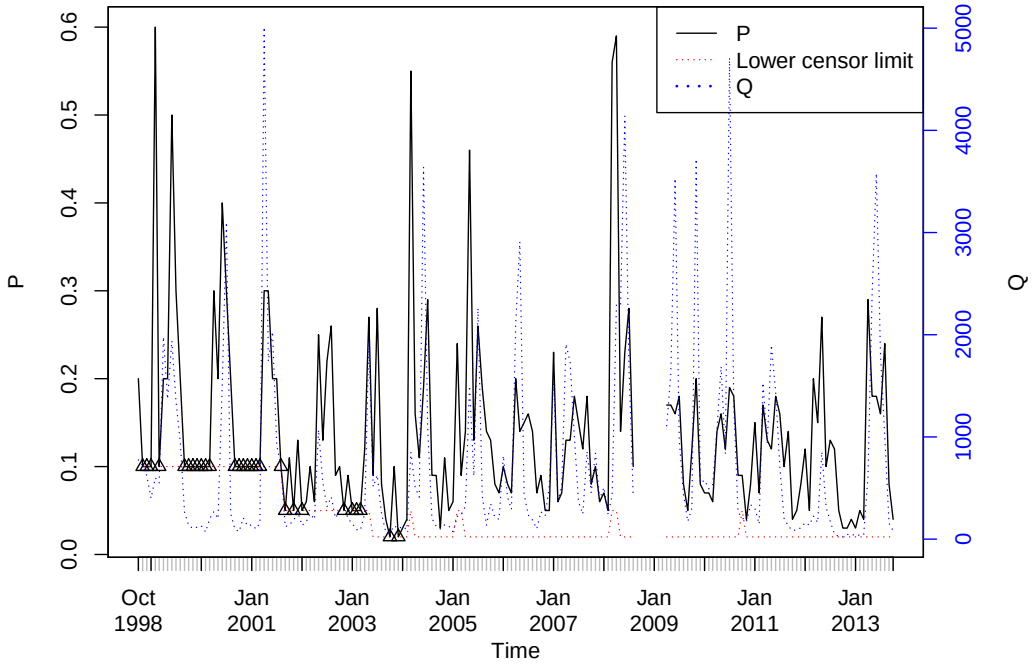


Figure 4: Time series plots of P (black line, scale shown on the left vertical axis), Q (blue line, scale shown on the right vertical axis) and the censor limits l_t (red line, in the same scale as that of P). Censored observations are marked by triangles.

It is also conjectured that the association between the logP and logQ may be seasonal. The variable `season` in `pts` is constructed to denote the quarter of the month with Quarter 1 consists of the first three months, namely, January, February, and March; Quarter 2 comprises the next three months, and so on. Figure 5 illustrates the seasonal relationship between `logP` and `logQ`. It is of interest whether there exists a linear trend in the logarithmic P. Preliminary analysis (not reported here) suggests the presence of significant autocorrelation in the regression errors. Thus, the general model takes the following form

$$\log(P_t) = \beta_1 t + f(\log(Q_t)) + \eta_t,$$

where f is some linear function that may be seasonal in the intercept and/or seasonal in the coefficient of `logQ`, and η_t follows an AR process.

Note that we need to determine whether the intercept and/or the regression coefficient are seasonal, and whether to include in the model a time trend, resulting in 8 combinations. Moreover, the AR order for the regression errors has to be specified. Assuming the maximal AR order to be m, we have to select among $8 \times m$ models, which can be done by selecting the best model that achieves the smallest AIC. Model selection by AIC is automated by the function `carxSelect()`. Here, the maximal AR order

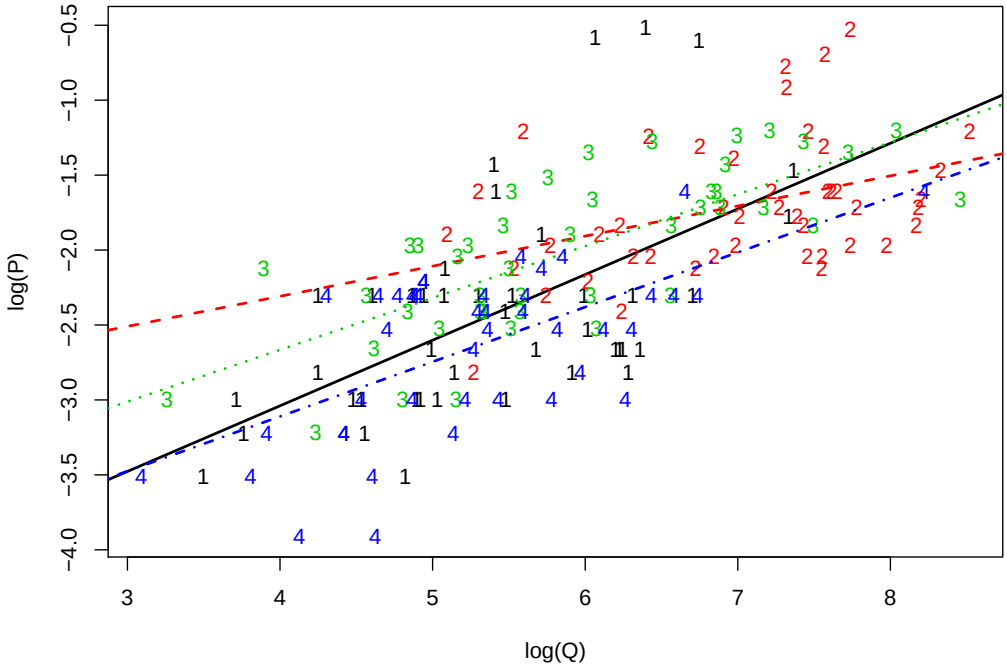


Figure 5: Scatter plot of `logP` versus `logQ`. The data are labeled by different numbers (1 for Quarter 1 and so on) and colors (black, red, green, blue for Quarter 1, 2, 3 and 4, respectively). Least squares quaterly regression lines (solid, dashed, dotted, and dash dotted) are superimposed with the same color as the data points.

is 3.

```
> arOrder <- 3
```

The list of models, named `M1` to `M8`, is specified in the following code.

```
> s1 <- logP ~ logQ
> s2 <- logP ~ tInMonth + logQ
> s3 <- logP ~ logQ:as.factor(season)
> s4 <- logP ~ tInMonth + logQ:as.factor(season)
> s5 <- logP ~ as.factor(season) + logQ - 1
> s6 <- logP ~ tInMonth + as.factor(season) + logQ - 1
> s7 <- logP ~ as.factor(season) + logQ:as.factor(season) - 1
> s8 <- logP ~ tInMonth + as.factor(season) +  logQ:as.factor(season) - 1
> fmls <- c(s1,s2,s3,s4,s5,s6,s7,s8)
> names(fmls) <- paste0('M',seq(1,8))
```

The model selection is performed by invoking the function `carxSelect()` which has two required arguments: a list of formulas and the maximal AR orders, plus an optional argument `detect.outlier`, which by default is `TRUE`, indicates whether outliers should be detected in the model. The function `carxSelect()` returns a `"carx"` object comprising an additional element `selectionInfo` which is a list containing more information about the selection result, including `aicMat` which is a matrix whose (i, j)th element is the AIC of the model represented by the i^{th} formula in the list and AR order j.

For the purpose of illustrating the prediction by the `predict()` method, we use all data up to the end of 2012 for model selection and fitting, and then check the model prediction against the observed data in 2013.

```
> cs <- carxSelect(fmls, arOrder, data = pts['1998/2012'],
+                  detect.outlier = TRUE, CI.compute = TRUE)
> print(round(cs$selectionInfo$aicMat,1))

     AR1   AR2   AR3
```

```
M1 -41.8 -39.6 -38.2
M2 -39.7 -38.1 -36.8
M3 -51.2 -47.5 -45.1
M4 -49.4 -45.9 -43.6
M5 -53.2 -49.6 -47.1
M6 -51.5 -48.2 -45.8
M7 -53.9 -50.8 -47.6
M8 -52.8 -49.8 -46.8
```

A summary of the model fit is shown below.

```
> summary(cs)

Call:
carx.formula(formula = formula(m0), data = m0$data, p = m0$p,
    CI.compute = ..1)


Coefficients:
                         Estimate    StdErr   lowerCI upperCI p.value
as.factor(season)1      -6.053239  0.633190 -7.325537 -4.9191  <2e-16 ***
as.factor(season)2      -3.455734  0.612764 -4.696565 -2.2607  <2e-16 ***
as.factor(season)3      -4.235837  0.414149 -5.028568 -3.4297  <2e-16 ***
as.factor(season)4      -4.854407  0.476015 -5.764154 -3.9324  <2e-16 ***
as.factor(season)1:logQ  0.633582  0.111002  0.436431  0.8566  <2e-16 ***
as.factor(season)2:logQ  0.248797  0.086893  0.073705  0.4247   0.006 **
as.factor(season)3:logQ  0.373538  0.068555  0.235532  0.5053  <2e-16 ***
as.factor(season)4:logQ  0.389721  0.087467  0.217672  0.5584  <2e-16 ***
AR1                      0.075072  0.090671 -0.137839  0.2227   0.596
sigma                    0.482897  0.028770  0.412907  0.5265  <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

AIC:
[1] -53.85639
```

The fitted model can be visualized by calling the `plot()` function, which is shown in Figure 6. The fitted values appear to track the data well.

```
> plot(cs)
```

We examine the goodness-of-fit of the fitted model via the `tsdiag()` function which generates 4 diagnostic plots in Figure 7. These plots indicate that the fitted model provides good fit to the data.

```
> tsdiag(cs)
```

The selected model can be interpreted as follows. The linear trend was not selected, suggesting no significant long-term change in the P concentrations. The intercept and the regression coefficient of logQ were seasonal. The regression errors appeared to be mildly auto-correlated and can be modeled as an order 1 AR process, although the AR coefficient was not significant.

Finally, we compute the prediction of `logP` via the `predict()` method and compare the prediction with the actual data from January to October 2013. Note the prediction makes use of observed discharge data in 2013. Figure 8 shows the point predictors (blue dashed line) against the actual values (black solid line) and the 95% prediction bands (red lines), which indicates that the prediction tracks the actual data well.

A simulation study on model selection

In this section, we report a simulation study on the effectiveness of model selection by minimizing the AIC. Recall this functionality is implemented by the `carxSelect()` function which outputs the model with the smallest AIC from a set of models of various AR orders up to some pre-specified maximum order.

We restrict the simulation study to the problem of selecting the AR order with the same model specification, i.e, the same set of regressors, which is conducted as follows. We simulated 1000 series

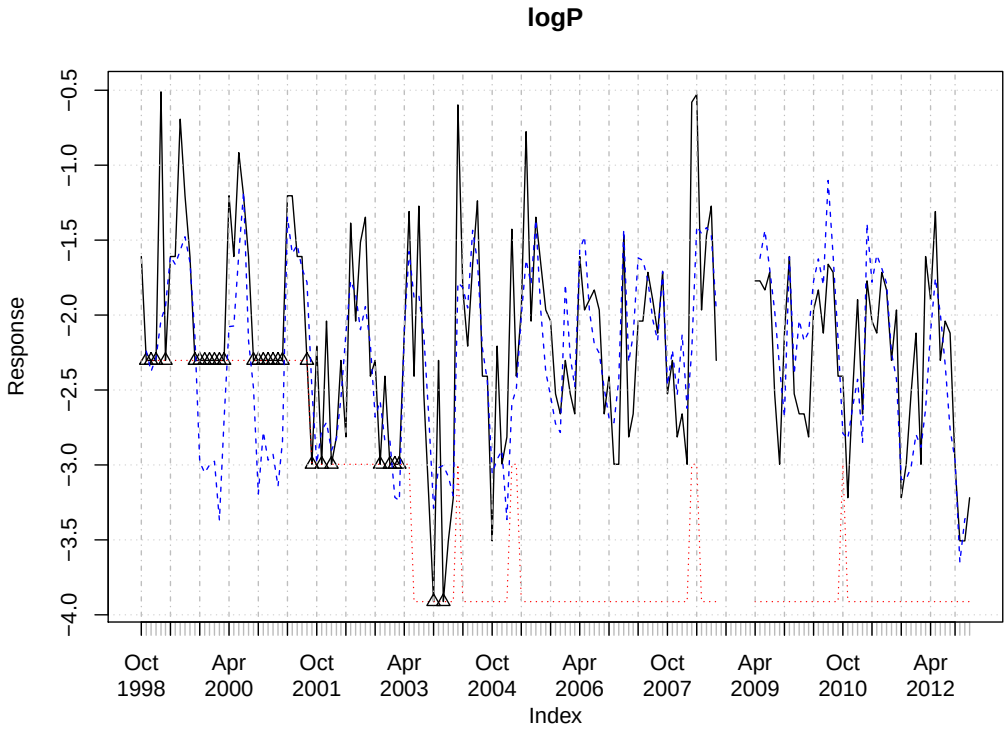


Figure 6: Time plot of the raw data and fitted values from the CARX model. The observed responses are connected as a solid black line, and the lower censoring limits drawn as a red dotted line with censored observations marked by triangles pointing up. The fitted values are connected as a blue dashed line. Outliers (if detected) are marked with a dashed red vertical line.

by calling `carxSim()` with the default setting, hence the true AR order is equal to 2, and for each simulated series we selected the best model among the models with the AR order from 1 to 6. Since the uncensored data were available in the simulation, we repeated the model selection with the uncensored observed data, for comparing with the results using the censored data. This simulation study can be reproduced by the following code.

```
> singleTestSelectAROrder <- function(iter)
+ {
+   seed <- 1375911
+   cts <- carxSim(seed = iter*seed)
+   m0 <- carxSelect(list(f1 = as.formula(y~X1+X2-1)), max.ar = 6,
+                    data=cts[,c("y","X1","X2")],detect.outlier = FALSE)
+   m1 <- carxSelect(list(f1 = as.formula(y~X1+X2-1)),max.ar = 6,
+                    data = cts, detect.outlier = FALSE)
+   c(m0$fitted$p, m1$fitted$p)
+ }
> nRep <- 1000
> orders <- parallel::mclapply( 1:nRep, singleTestSelectAROrder,
+           mc.cores = parallel::detectCores() - 1)
> orders <- do.call(rbind, lapply(orders, matrix, ncol = 2, byrow = TRUE))
> freqComDat = count(orders[1,])
> freqCenDat = count(orders[2,])
```

The selected orders are reported in Table 1, which shows that the true order can be recovered with an empirical probability of 52.7%, and the results using censored and complete data are comparable.

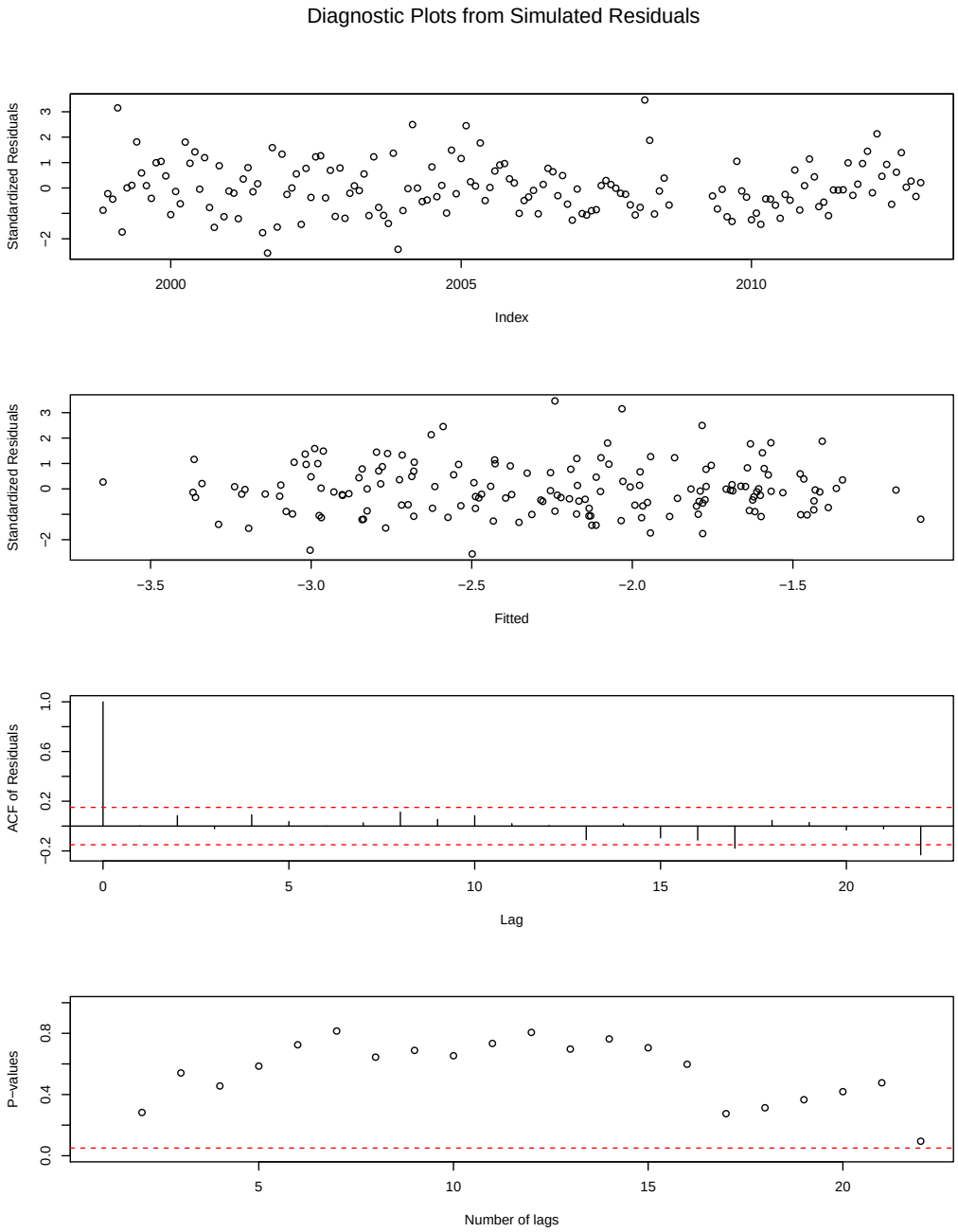


Figure 7: Model diagnostic plots. The plots from top to bottom correspond respectively to the time series plots of standadized simulated residuals, residuals versus fitted values, the residual autocorrelation plots, and the Ljung-Box test statistics of the residuals.

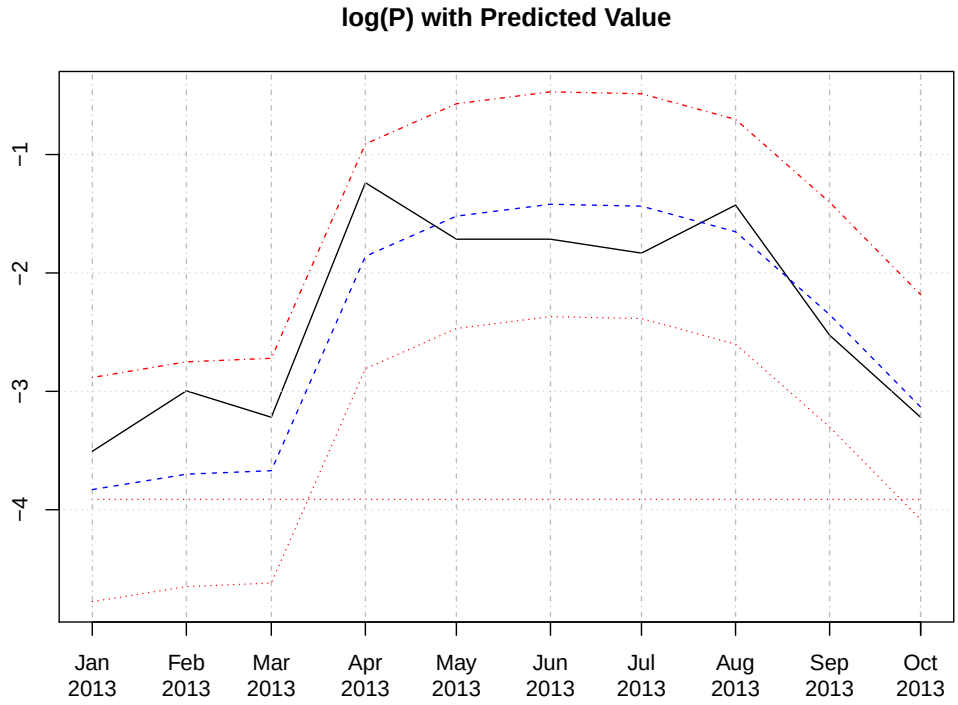


Figure 8: Plot of predictions and observed values. The observed values are drawn by black solid line. The predicted values, lower and upper bound of confidence intervals are drawn by blue dashed, red dotted, and red dash dotted lines respectively.

AR order	1	2	3	4	5	6
Frequency (complete data)	17	626	139	97	69	52
Frequency (censored data)	37	527	160	99	101	76

Table 1: Summary of selected orders. The frequency of selected orders with complete data and censored data are reported.

Performance of model prediction

In this section we report a simulation study about the empirical performance of the model prediction procedure. A series of 210 data was simulated using the default parameters of `carxSim()`, with the first 200 data used to estimate the model, and the last 10 observations used to compare with the predicted values based on the fitted model and the simulated future covariate values. The above procedure was repeated 500 times. The empirical coverage rates of the 95% ℓ-step ahead prediction intervals, $\ell = 1, 2, \ldots, 10$, are summarized in Table 2, which indicates a close match between the empirical and nominal coverage rates. The simulation exercise can be reproduced by the following code.

```
> nRep = 500; nObs = 200; n.ahead=10
> runSimPredCR <- function()
+ {
+   set.seed(0)
+   crMat = matrix(nrow = n.ahead, ncol = nRep)
+   for(iRep in 1:nRep)
+   {
+     sdata = carxSim(nObs = nObs + n.ahead)
+     trainingData = sdata[1:nObs,]
+     testData = sdata[-(1:nObs),]
+     mdl = carx(y ~ X1 + X2 - 1, data = trainingData, p = 2)
+     newxreg = testData[,c('X1','X2')]
+     predVal = predict(mdl, newxreg = newxreg, n.ahead = n.ahead)
```

```
+     crInd = (predVal$ci[,1] <= testData$y) & (predVal$ci[,2] >= testData$y)
+     crMat[,iRep] = crInd
+   }
+   crPred = apply(crMat,1,mean)
+ }
> runSimPredCR()
```

n.ahead	1	2	3	4	5	6	7	8	9	10
Empirical Coverage	0.954	0.932	0.946	0.952	0.946	0.946	0.938	0.946	0.948	0.952

Table 2: Empirical coverage rates of nominally 95% predictive confidence intervals for ℓ-step-ahead prediction, for $\ell = 1, 2, \ldots, 10$.

Conclusion

In summary, we have reviewed the quasi-likelihood method to estimate a censored time series regression model and introduced the **carx** package in which quasi-likelihood estimation is implemented, together with other useful functions for model selection, prediction, diagnostics and outlier detections. We illustrated the **carx** package with two major examples, and shed light on the effectiveness of model selection via minimizing AIC and the prediction accuracy.

Future work includes extending the method for more complex regression noise structure than the AR model, for instance, the more general ARIMA model, and updating the package according to the feedback from the public.

Bibliography

J. D. Cryer and K. S. Chan. *Time Series Analysis: With Applications in R*. Springer-Verlag, New York, 2008. URL https://doi.org/10.1007/978-0-387-75959-3. [p216, 217]

J. Buckley and I. James. Linear regression with censored data. *Biometrika*, 66(3):429–436, 1979. URL https://doi.org/10.1093/biomet/66.3.429. [p213]

K.-S. Chan and B. Ripley. *TSA: Time Series Analysis*, 2012. URL https://CRAN.R-project.org/package=TSA. R package version 1.01. [p222]

A. Genz, F. Bretz, T. Miwa, X. Mi, and T. Hothorn. *mvtnorm: Multivariate Normal and t Distributions*, 2017. URL https://CRAN.R-project.org/package=mvtnorm. R package version 1.0-6. [p216]

C. Gourieroux, A. Monfort, E. Renault, and A. Trognon. Simulated residuals. *Journal of Econometrics*, 34(1):201–252, 1987. URL https://doi.org/10.1016/0304-4076(87)90073-X. [p216]

A. Henningsen. Estimating censored regression models in R using the censreg package. *University of Copenhagen*, 2010. [p213]

A. Henningsen. *censReg: Censored Regression (Tobit) Models*, 2013. URL http://CRAN.R-project.org/package=censReg. R package version 0.5-20. [p213]

J. Huang and A. J. Rossini. Sieve estimation for the proportional-odds failure-time regression model with interval censoring. *Journal of the American Statistical Association*, 92(439):960–967, 1997. ISSN 01621459. URL http://www.jstor.org/stable/2965559. [p214]

C. Kleiber and A. Zeileis. *Applied Econometrics with R*. Springer-Verlag, New York, 2008. URL http://CRAN.R-project.org/package=AER. ISBN 978-0-387-77316-2. [p213]

S. Konishi and G. Kitagawa. *Information Criteria and Statistical Modeling*. Springer-Verlag, 2008. URL https://doi.org/10.1007/978-0-387-71887-3. [p215]

L. Lee. *NADA: Nondetects And Data Analysis for Environmental Data*, 2013. URL http://CRAN.R-project.org/package=NADA. R package version 1.5-6. [p213]

R. D. Libra, C. F. Wolter, and R. J. Langel. *Nitrogen and Phosphorus Budgets for Iowa and Iowa Watersheds*. Iowa Department of Natural Resources, Geological Survey, 2004. [p222]

A. D. Martin, K. M. Quinn, and J. H. Park. MCMCpack: Markov chain monte carlo in R. *Journal of Statistical Software*, 42(9):22, 2011. URL https://doi.org/10.18637/jss.v042.i09. [p213]

A. I. McLeod, N. M. Mohammad, J. Veenstra, and A. El-Shaarawi. *cents: Censored Time Series*, 2014. URL http://CRAN.R-project.org/package=cents. R package version 0.1-41. [p213]

J. W. Park, M. G. Genton, and S. K. Ghosh. Censored time series analysis with autoregressive moving average models. *Canadian Journal of Statistics*, 35(1):151–168, 2007. ISSN 1708-945X. URL https://doi.org/10.1002/cjs.5550350113. [p213, 214]

J. A. Ryan and J. M. Ulrich. *xts: eXtensible Time Series*, 2017. URL https://CRAN.R-project.org/package=xts. R package version 0.10-0. [p214]

K. E. Schilling, K. S. Chan, H. Liu, and Y. K. Zhang. Quantifying the effect of land use land cover change on increasing discharge in the upper mississippi river. *Journal of Hydrology*, 387(3):343–345, 2010. URL https://doi.org/10.1016/j.jhydrol.2010.04.019. [p224]

F. L. Schumacher, V. H. Lachos, and C. E. Galarza. *ARCensReg: Fitting Univariate Censored Linear Regression Model with Autoregressive Errors*, 2016. URL http://CRAN.R-project.org/package=ARCensReg. R package version 2.1. [p213]

G. M. Tallis. The moment generating function of the truncated multi-normal distribution. *Journal of the Royal Statistical Society B*, 23(1):223–229, 1961. [p216]

J. Tobin. Estimation of relationships for limited dependent variables. *Econometrica: journal of the Econometric Society*, pages 24–36, 1958. URL https://doi.org/10.2307/1907382. [p213]

C. Wang and K.-S. Chan. Quasi-likelihood estimation of a censored autoregressive model with exogenous variables. *Journal of the American Statistical Association*, 2017a. URL https://doi.org/10.1080/01621459.2017.1307115. [p213, 214, 215, 216, 219, 221]

C. Wang and K.-S. Chan. *carx: Censored Autoregressive Model with Exogenous Covariates*, 2017b. URL https://CRAN.R-project.org/package=carx. R package version 0.7.1. [p213]

C. Wang, K.-S. Chan, and K. E. Schilling. Total phosphorus concentration trends in 40 iowa rivers, 1999 to 2013. *Journal of Environmental Quality*, 2016. URL https://doi.org/10.2134/jeq2015.07.0365. [p222]

T. W. Yee. *Vector Generalized Linear and Additive Models*. Springer-Verlag, 2015. URL https://doi.org/10.1007/978-1-4939-2818-7. [p213]

S. L. Zeger and R. Brookmeyer. Regression analsis with censored autocorrelated data. *Journal of the American Statistical Association*, 81(395):722–729, 1986. ISSN 01621459. URL https://doi.org/10.1080/01621459.1986.10478328. [p213, 216]

Chao Wang
Department of Statistics and Actuarial Science
The University of Iowa
Iowa City, IA 52242
USA
ORCiD: 0000-0001-8976-3773
chao-wang@uiowa.edu

Kung-Sik Chan
Department of Statistics and Actuarial Science
The University of Iowa
Iowa City, IA 52242
USA
kung-sik-chan@uiowa.edu

liureg: A Comprehensive R Package for the Liu Estimation of Linear Regression Model with Collinear Regressors

by Muhammad Imdadullah, Muhammad Aslam, Saima Altaf

Abstract The Liu regression estimator is now a commonly used alternative to the conventional ordinary least squares estimator that avoids the adverse effects in the situations when there exists a considerable degree of multicollinearity among the regressors. There are only a few software packages available for estimation of the Liu regression coefficients, though with limited methods to estimate the Liu biasing parameter without addressing testing procedures. Our **liureg** package can be used to estimate the Liu regression coefficients utilizing a range of different existing biasing parameters, to test these coefficients with more than 15 Liu related statistics, and to present different graphical displays of these statistics.

Introduction

For data collected either from a designed experiment or from an observational study, the ordinary least square (OLS) method does not provide precise estimates of the effect of any explanatory variable (regressor) when regressors are interdependent (collinear with each other). Consider a multiple linear regression (MLR) model,

$$y = X\beta + \varepsilon,$$

where y is an $n \times 1$ vector of observations on dependent variable, X is known design matrix of order $n \times p$, β is a $p \times 1$ vector of unknown parameters, and ε is an $n \times 1$ vector of random errors with mean zero and variance $\sigma^2 I_n$, where I_n is an identity matrix of order n.

The OLS estimator (OLSE) of β is given by

$$\hat{\beta} = (X'X)^{-1}X'y,$$

which depends on the characteristics of the matrix $X'X$. If $X'X$ is ill-conditioned (near dependencies among various regressors of $X'X$ exist) or $det(X'X) \approx 0$, then the OLS estimates are sensitive to a number of errors, such as non-significant or imprecise regression coefficients (Kmenta, 1980) with wrong sign and non-uniform eigenvalues spectrum. Moreover, the OLS method, for example, can yield a high variance of estimates, large standard errors, and wide confidence intervals.

Researchers may be tempted to eliminate regressor(s) causing problems by consciously removing regressor from the model or by using some screening method such as stepwise and best subset regression etc. However, these methods may destroy the usefulness of the model by removing relevant regressor(s) from the model. To control variance and instability of the OLS estimates, one may regularize the coefficients, with some regularization methods such as the ridge regression (RR), Lasso regression and Liu regression (LR) methods etc., as alternative to the OLS. Computationally, the RR ($\hat{\beta}_r = (X'X + kI)^{-1}X'y$) suppresses the effects of collinearity and reduces the apparent magnitude of the correlation among regressors in order to obtain more stable estimates of the coefficients than the OLS estimates and it also improves the accuracy of prediction (see Hoerl and Kennard, 1970; Montgomery and Peck, 1982; Myers, 1986; Rawlings et al., 1998; Seber and Lee, 2003; Tripp, 1983, etc.). However, the ridge coefficient is a complicated function of k when some popular methods (such as given in Golub et al. (1979), Mallows (1973) and McLeod and Xu (2017) etc.) are used for (optimal) selection of k. Different applications can yield values for k which are too small to correct the problem of the ill-conditioned product, $X'X$. In such cases, the RR may still be unstable. Similarly, the choice of k belongs to the researcher, there being no consensus regarding how to select optimal k. As such, other innovative methods were needed to deal with collinear data. Liu (1993) proposed another biased estimator to mitigate the collinearity effect on regressors. They also discussed some of the properties and methods for suitable selection of biasing parameter used in LR. For further detail, see Section "Liu regression estimator."

We have developed the **liureg** (Imdadullah and Aslam, 2017) package to provide the functionality of Liu related computations. The package provides the most complete suite of tools for the LR available in R. Table 1 provides a comparison with other alternatives. For package development and R documentation, we followed Wickham (2015); Leisch (2008); Team (2015). The **ridge** package by Cule and De Iorio (2012), **lmridge** by Imdadullah and Aslam (2016a) and `lm.ridge` from the **MASS** by Venables and Ripley (2002) also provided guidance in coding.

	lrmest (1)	ltsbase (2)	liureg
Standardization of regressors			
	✓	✓	✓
Estimation and testing of Liu coefficient			
Estimation	✓	✓	✓
Testing	✓		✓
SE of coeff.	✓		✓
Liu related statistics			
R^2	✓		✓
Adj-R^2			✓
Variance			✓
Bias^2			✓
MSE			✓
F-test			✓
σ^2			✓
C_L			✓
Effective df			✓
Hat matrix			✓
Var-Cov matrix			✓
VIF			✓
Residuals		✓	✓
Fitted values		✓	✓
Predict values			✓
Liu model selection			
GCV			✓
AIC&BIC			✓
PRESS			✓
Liu related graphs			
Liu trace			✓
Bias, Var, MSE		✓	✓
AIC, BIC			✓

Table 1: Comparison of Liu related R packages (1 Dissanayake and Wijekoon, 2016; 2 Kan et al., 2013)

In the available literature, there are only two R packages capable of estimating and/or testing of the Liu coefficients. The R packages mentioned in Table 1 are compared with our **liureg** package. The **lrmest** package (Dissanayake and Wijekoon, 2016) computes different estimates such as the OLS, ordinary ridge regression (ORR), Liu estimator (LE), LE type-1, 2, 3, adjusted Liu estimator (ALTE), and their type-1, 2, 3 etc. Moreover, **lrmest** provides scalar mean square error (MSE), prediction residual error sum of squares (PRESS) values of some of the estimators. The testing of ridge coefficient is performed only on scalar `k`, however, for a vector of `d`, the function `liu()` of **lrmest** package returns only MSE along with value of the biasing parameter used. The **ltsbase** package (Kan et al., 2013) computes ridge and Liu estimates based on the least trimmed squares (LTS) method. The MSE value from four regression models can be compared graphically if the argument `plot=TRUE` is passed to the `ltsbase()` function. There are three main functions, (i) `ltsbase()` computes the minimum MSE values for six methods: OLS, ridge, ridge based on LTS, LTS, Liu, and Liu based on LTS method for sequences of biasing parameters ranging from 0 to 1, (ii) the `ltsbaseDefault()` function returns the fitted values and residuals of the model having minimum MSE, and (iii) the `ltsbaseSummary()` function returns the regression coefficients and the biasing parameter for the best MSE among the four regression models.

It is important to note that the **ltsbase** package displays these statistics for models having minimum MSE (bias and variance are not displayed in their output), while our package, **liureg**, computes these and all other statistics not only for scalar but also for vector biasing parameter.

This paper outlines the collinearity detection methods available in the existing literature and uses the **mctest** (Imdadullah and Aslam, 2016b) package through an illustrative example. To overcome the issues of the collinearity effect on regressors a thorough introduction to Liu regression, properties of the Liu estimator, different methods for the selecting values of d, and testing of the Liu coefficients is presented. Finally, estimation of the Liu coefficients, methods of selecting a biasing parameter, testing of the Liu coefficients, and different Liu related statistics are implemented in R within the **liureg** package.

Collinearity detection

Diagnosing collinearity is important to many researchers. It consists of two related but separate elements: (1) detecting the existence of collinear relationship among regressors and (2) assessing the extent to which this relationship has degraded the parameter estimates. There are many diagnostic measures used for detection of collinearity in the existing literature provided by various authors (Belsley et al., 1980; Curto and Pinto, 2011; Farrar and Glauber, 1967; Fox and Weisberg, 2011; Gunst and Mason, 1977; Klein, 1962; Koutsoyiannis, 1977; Kovács et al., 2005; Marquardt, 1970; Theil, 1971). These diagnostic methods assist in determining whether and where some corrective action is necessary (Belsley et al., 1980). Widely used, and the most suggested diagnostics, are the value of pair-wise correlations, the variance inflation factor (VIF)/ tolerance (TOL) (Marquardt, 1970), the eigenvalues and eigenvectors (Kendall, 1957), the CN & CI (Belsley et al., 1980; Chatterjee and Hadi, 2006; Maddala, 1988), Leamer's method (Greene, 2002), Klein's rule (Klein, 1962), the tests proposed by Farrar and Glauber (Farrar and Glauber, 1967), the Red indicator (Kovács et al., 2005), the corrected VIF (Curto and Pinto, 2011), and Theil's measures (Theil, 1971), (see also Imdadullah et al. (2016)). All of these diagnostic measures are implemented in a the R package **mctest** (Imdadullah and Aslam, 2016b). Below, we use the Hald dataset (Hald, 1952), for testing collinearity among regressors. We then use the **liureg** package to compute the Liu regression coefficients for different Liu related statistics and methods of selection of Liu biasing parameter is performed. For optimal choice of biasing parameter,a graphical representation of the Liu coefficients is considered, along with a bias variance trade-off plot. In additino, model selection criteria is also performed. The Hald data are about heat generated during setting of 13 cement mixtures of 4 basic ingredients and used by Hoerl et al. (1975). Each ingredient percentage appears to be rounded down to a full integer. The data set is included in both the **mctest** and **liureg** packages.

Collinearity detection: An example

```
R > data(Hald)
R > x <- Hald[, -1]
R > y <- Hald[,  1]
R > mctest (x, y)

Call:
omcdiag(x = x, y = y, Inter = TRUE, detr = detr, red = red, conf = conf,
        theil = theil, cn = cn)

Overall Multicollinearity Diagnostics

MC Results detection
Determinant |X'X|:         0.0011         1
Farrar Chi-Square:        59.8700         1
Red Indicator:             0.5414         1
Sum of Lambda Inverse:   622.3006         1
Theil's Method:            0.9981         1
Condition Number:        249.5783         1

1 --> COLLINEARITY is detected
0 --> COLLINEARITY is not detected by the test

==================================
Eigenvalues with INTERCEPT
                 Intercept     X1     X2      X3       X4
Eigenvalues:        4.1197 0.5539 0.2887  0.0376   0.0001
Condition Indices:  1.0000 2.7272 3.7775 10.4621 249.5783
```

The results from all overall collinearity diagnostic measures indicate the existence of collinearity among regressor(s). These results do not tell which regressor(s) are reasons of collinearity. The individual collinearity diagnostic measures can be obtained though:

```
> mctest(x = x, y, all = TRUE, type = "i")

Call:
imcdiag(x = x, y = y, method = method, corr = FALSE, vif = vif,
      tol = tol, conf = conf, cvif = cvif, leamer = leamer, all = all)
```

```
All Individual Multicollinearity Diagnostics in 0 or 1

   VIF TOL Wi Fi Leamer CVIF Klein
X1   1   1  1  1      0    0     0
X2   1   1  1  1      1    0     1
X3   1   1  1  1      0    0     0
X4   1   1  1  1      1    0     1

1 --> COLLINEARITY is detected
0 --> COLLINEARITY in not detected by the test

X1 , X2 , X3 , X4 , coefficient(s) are non-significant may be due to multicollinearity

R-square of y on all x: 0.9824

* use method argument to check which regressors may be the reason of collinearity
```

The results from most of the individual collinearity diagnostics suggest that all of the regressors are the reason for collinearity among regressors. The last line of the `imcdiag()` function's output suggests that method argument should be used to check which regressors may be the reason of collinearity among different regressors. This finding suggest that one should use regularization method such as LR.

Liu regression estimator

To deal with multicollinear data, Liu (1993) formulated a new class of biased estimators that has combined benefits of ORR by Hoerl and Kennard (1970) and the Stein type estimator Stein (1956), $\hat{\beta}_S = c\hat{\beta}$, where c is parameter $0 < c < 1$ to avoid their disadvantages. The Liu estimator (LE) can be defined as,

$$\begin{aligned} \hat{\beta}_d &= (X'X + I_p)^{-1}(X'y + d\hat{\beta}_{ols}), \\ &= (X'X + I_p)^{-1}(X'X + dI_p)\hat{\beta}_{ols}, \\ &= F_d\hat{\beta}_{ols}, \end{aligned} \tag{1}$$

where d is the Liu parameter also known as the biasing (tuning or shrinkage) parameter and lies between 0 and 1 (i.e., $0 \leq d \leq 1$), I_p is the identity matrix of order $p \times p$, and $\hat{\beta}$ is OLSE.

and other statistical areas, the LE has produced a number of new techniques and ideas, see for example Akdeniz and Kaçiranlar (2001); Hubert and Wijekoon (2006); Jahufer and Chen (2009, 2011, 2012); Kaçiranlar et al. (1999); Kaçiranlar and Sakalhoğlu (2001); Torigoe and Ujiie (2006).

However, Liu (2011) and Druilhet and Mom (2008) have made statements that the biasing parameter d may lie outside the range given by Liu (1993), that is, it may be less than 0 or greater than 1. The LE is a linear transformation of the OLSE, $\hat{\beta}_d = \hat{\beta}_{ols}$.

the main interest of LE lies in the suitable selection of d for which MSE is minimum and that the efficiency of estimators improves, as compared to other values of d. The $\hat{\beta}_d$ is named as the LE by Akdeniz and Kaçiranlar (1995) and Gruber (1998). Liu (1993), in applications to econometrics and engineering, provided some important methods for the selection of d and also provided numerical examples using an iterative minimum MSE method to get the smallest possible value to overcome the problem of collinearity in an effective manner.

Reparameterization

The design matrix $X_{n\times p}$ and response variable $y_{n\times 1}$ should be standardized, scaled or centered first such that information matrix $X'X$ is in the correlation form and vector $X'y$ is in the form of the correlation among regressors and the response variable. Consider the regression model, $y = \beta_0 1 + \widetilde{X}\beta_1 + \varepsilon$, where $\widetilde{X}$ is centered and $1 = c(1, 1, \cdots, 1)'$. The value for β_0 can be estimated by using $\overline{y}$. Let $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_p \geq 0$, be the ordered eigenvalues of the matrix $\widetilde{X}'\widetilde{X}$ and $q_1, q_2, \cdots, q_p$ be the eigenvectors corresponding to their eigenvalues, such that $Q = (q_1, q_2, \cdots, q_p)$ is an orthogonal matrix of $\widetilde{X}'\widetilde{X}$ and

$$\Lambda = \begin{pmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_p \end{pmatrix},$$

therefore, the model can be rewritten in canonical form as $y = \beta_0 1 + Z\alpha + \varepsilon$, where $Z = \widetilde{X}Q$ and $\alpha = Q'\beta_1$. Note that, $\Lambda = Z'Z = Q'\widetilde{X}'\widetilde{X}Q$. The estimate of α is $\hat{\alpha} = \Lambda^{-1}Z'y$. Similarly, Eq. 1 can be written in canonical form as,

$$\hat{\alpha}_d = (\Lambda + I_p)^{-1}(Z'y + d\hat{\alpha}).$$

The corresponding estimates of $\hat{\beta}_1$ and $\hat{\beta}_d$ can be obtained by following the relations $\hat{\beta}_1 = Q\hat{\alpha}$ and $\hat{\beta}_d = Q\hat{\alpha}_d$, respectively. For simplification of notations, $\widetilde{X}$ and $\hat{\alpha}$ will be represented as X and β, respectively.

The fitted values of the LE can be found using Eq. 1,

$$\begin{aligned} \hat{y}_d &= X\hat{\beta}_d, \\ &= X(X'X + I_p)^{-1}(X'y + d)\hat{\beta}, \\ &= H_d\, y, \end{aligned}$$

where, H_d is LE the matrix (Liu, 1993; Walker and Birch, 1988). It is worthwhile to note that H_d is not idempotent because it is not a projection matrix, therefore it is called quasi-projection matrix.

As $\hat{\beta}_d$ is computed on centered variables, they need to be converted back to the original scale:

$$\hat{\beta} = \left(\frac{\hat{\beta}_{dj}}{S_{xj}}\right),$$

where S_{xj} is the scaling method of regressors.

The intercept term for the LE ($\hat{\beta}_{0d}$) can be estimated using the following relation:

$$\begin{aligned} \hat{\beta}_{0d} &= \overline{y} - (\hat{\beta}_{1d}, \cdots, \hat{\beta}_{pd})\overline{x}'_j \\ &= \overline{y} - \sum_{j=1}^{p} \overline{x}_j \hat{\beta}_{jd}. \end{aligned} \tag{2}$$

Properties of the Liu estimator

Like the linear RR, the Liu regression is also the most popular method among biased methods, because of its relation to OLS. Its statistical properties have been studied by Akdeniz and Kaçiranlar (1995, 2001), Arslan and Billor (2000), Kaçiranlar and Sakalhoğlu (2001), Kaçiranlar et al. (1999) and Sakalhoğlu et al. (2001) among many others. Due to comprehensive properties of the LE, researchers have been attracted towards this area of research.

For $d = 1, \hat{\beta}_d = \beta_{ols}$. In which case, LE is the shrinkage estimator, though biased, but has lower MSE than OLS. That is, $MSE(\hat{\beta}_d) < MSE(\hat{\beta}_{ols})$ (see Sakalhoğlu et al., 2001, etc.).

Let X_j denote the jth column of $X(j = 1, 2, \cdots, p)$, where $X_j = (x_{1j}, x_{2j}, \cdots, x_{nj})'$. As already discussed, the regressors are centered, thus, the intercept will be zero and can thereby be removed from the model. However, it can be estimated from relation given in Eq. 2. Table 2, lists the Liu properties that are implemented in our **liureg** package.

Theoretically and practically, LR is used to propose new methods for the choice of the biasing parameter d to investigate the properties of LR, since the biasing parameter plays a key role while the optimal choice of d is the main issue in this context. In the literature, many methods for the selection of an appropriate biasing parameter d have been studied by Akdeniz and Özkale (2005), Arslan and Billor (2000), Akdeniz et al. (2006), Özkale and Kaçiranlar (2007), and Liu (1993).

Methods of selecting values of d

The existing methods to select biasing parameter in the LR may not fully address the problem of ill-conditioning when there exists severe multicollinearity, while the appropriate selection of biasing parameter d also remains a problem of interest. The parameter d should be selected when there are improvements in the estimates (have stable estimates) or prediction is improved.

The optimal value of d is one which gives minimum MSE. There is one optimal d for any problem

Sr.#	Property	Formula
1)	Linear transformation	The LE is a linear transformation of the OLSE ($\hat{\beta}_d = F_d\hat{\beta}$)
2)	Wide range d	Wide range of d have smaller MSE than the OLS
3)	Optimal d	An optimal d always exists that gives minimum MSE
4)	Mean	$E(\hat{\beta}_d) = F_d\beta$, where $F_d = (X'X + I_p)^{-1}(X'X + dI_p)$
5)	Bias	$Bias = Q'(F_d - I_p)\beta$
6)	Var-Cov matrix	$Cov(\hat{\beta}_d) = \sigma^2 F_d(X'X)^{-1}F_d'$
7)	MSE	$MSE(\hat{\beta}_d) = \sigma^2 F_d(X'X)^{-1}F_d + (F_d - I_p)\beta\beta'(F_d - I_p)'$ $= \sigma^2 \sum_{j=1}^{p} \frac{(\lambda_j + d)^2}{\lambda_j(\lambda_j + 1)^2} + (d-1)^2 \sum_{j=1}^{p} \frac{\beta^2}{(\lambda + 1)^2}$
8)	Effective DF (EDF)	$EDF = trace[XF_d(X'X)^{-1}X']$
9)	Larger regression coeff.	$\hat{\beta}_d'\hat{\beta}_d \geq \hat{\beta}_{ols}'\hat{\beta}_{ols}$
10)	Inflated RSS	$\sum(y - X\hat{\beta}_d)^2$

Table 2: Properties of Liu estimator.

by the analogy with the estimate of k in RR, a wide range of d ($-\infty < d < 1$) can give smaller MSE as compared to that of the OLS. For collinear data, a small change in d varies the LR coefficients rapidly. Therefore, a disciplined way of selecting the shrinkage parameter is required that minimizes the MSE. The biasing parameter d depends on the true regression coefficients (β) and the variance of the residuals σ^2, unfortunately these are unknown, but they can be estimated from the sample data.

We classified estimation methods as (i) Subjective or (ii) Objective

Subjective methods

In these methods, the selection of d is subjective or of judgmental nature and provides graphical evidence of the effect of collinearity on the regression coefficient estimates and also accounts for variation by the LE as compared to the OLSE. In these methods, the reasonable choice of d is done using the Liu trace and the plotting of bias, variance, and MSE. Like ridge trace, the Liu trace is also a graphical representation of the regression coefficients, $\hat{\beta}_d$, as a function of d over the interval $(-\infty, \infty)$. Similarly, the plotting of bias, variance, and MSE from the LE may also be helpful in selecting an appropriate value of d. At the cost of bias, optimal d can be selected at which MSE is minimum. All these graphs can be used for selection of optimal (but judgmental) value of d from the horizontal axis to assess the effect of collinearity on each of the coefficients. These graphical representations do not provide a unique solution, rather they render a vaguely defined class of acceptable solutions. However, these traces are still useful graphical representations to check for some optimal d.

Objective methods

Objective methods, to some extent, are similar to judgmental methods for selection of biasing parameter d, but they require some calculations to obtain these biasing parameters. Table 3 lists widely used methods to estimate the biasing parameter d already available in the existing literature. Table 3 also lists other statistics that can be used for the selection of the biasing parameter d.

Testing of the Liu coefficients

Testing of the Liu coefficients is performed by following Aslam (2014) and Halawa and El-Bassiouni (2000). For testing $H_0 : \beta_{dj} = 0$ against $\beta_{dj} \neq 0$, the non-exact t-statistics defined by Halawa and El-Bassiouni (2000) are,

$$T_{dj} = \frac{\hat{\beta}_{dj}}{SE(\hat{\beta}_{dj})},$$

where $\hat{\beta}_{dj}$ is the jth Liu coefficient estimate and $SE(\hat{\beta}_{dj})$ is an estimate of standard error, which is the square root of the jth diagonal element of the covariance matrix of LE (see property # 6 in Table 2).

The statistics T_{dj} are assumed to follow Student's t distribution with $(n - p)$ df (Halawa and El-Bassiouni, 2000). Hastie and Tibshirani (1990) and Cule and De Iorio (2012) suggest using the df from $(n - trace(H_d))$. For large sample size, the asymptotic distribution of this statistic is normal (Halawa and El-Bassiouni, 2000).

Sr.#	Formula	Reference
1)	$d_{opt} = \frac{\sum_{j=1}^{p}\left[\frac{\alpha_j^2-\sigma^2}{(\lambda_j+1)^2}\right]}{\sum_{j=1}^{p}\left[\frac{\sigma^2+\lambda_j\alpha_j^2}{\lambda_j(\lambda_j+1)^2}\right]}$	Liu (1993)
2)	$\hat{d} = 1 - \hat{\sigma}^2\left[\frac{\sum_{j=1}^{p}\frac{1}{\lambda_j(\lambda_j+1)}}{\sum_{j=1}^{p}\frac{\hat{\alpha}_j^2}{(\lambda_j+1)^2}}\right]$	Liu (1993)
3)	$\hat{d}_{imp} = \frac{\sum_{i=1}^{n}\frac{\tilde{e}}{1-g_{ii}}\left(\frac{\tilde{e}_i}{1-h_{1-ii}} - \frac{\hat{e}_i}{1-h_{ii}}\right)}{\sum_{i=1}^{n}\left(\frac{\tilde{e}}{1-g_{ii}} - \frac{\hat{e}_i}{1-h_{ii}}\right)^2}$, where, $\hat{e} = y_i - x_i'(X'X - x_ix_i')^{-1}(X'y - x_iy_i)$, $\tilde{e} = y_i - x_i'(X'X + I_p - x_ix_i')^{-1}(X'y - x_iy_i)$, $G = X(X'X + I_p)^{-1}X'$, and $H \cong X(X'X)^{-1}X'$	Liu (2011)
4)	$PRESS_d = \sum_{i=1}^{n}(\hat{e}_{d(i)})^2$, where $\hat{e}_{d_{(i)}} = \frac{\hat{e}_{e_i}}{1-h_{1-ii}} - \frac{\hat{e}_i}{(1-h_{1-ii})(1-h_{ii})}(h_{1-ii} - \widetilde{h}_{d-ii})$, $\hat{e}_{d_i} = y_i - \hat{y}_{d_i}$, $\widetilde{H}_{d-ii}$ diagonal elements from Liu hat matrix, $h_{ii} = x_i'(X'X)^{-1}x_i$, and $h_{1-ii} = x_i'(X'X + I)^{-1}x_i$	Özkale and Kaçiranlar (2007)
5)	$C_L = \frac{SSR_d}{\hat{\sigma}^2} + 2\,trace(\widetilde{H}_d) - (n-2)$, where, $\widetilde{H}_d$ is hat matrix of LE	Mallows (1973)
6)	$GCV = \frac{SSR_d}{(n-[1+trace(\widetilde{H}_d)])^2}$	Liu (1993)
7)	$AIC = n\,log(RSS) + 2df$, $BIC = n\,log(RSS) + df\,log(n)$, where $df = trace(H_d)$	

Table 3: Different available methods to estimate d.

For testing overall significance of vector of LE ($\hat{\beta}_d$) with $E(\hat{\beta}_d) = F_d\,\beta$ and $Cov(\hat{\beta}_d)$, the F-statistic is,

$$F = \frac{1}{p}(\hat{\beta}_d - F_d\beta)'(Cov(\hat{\beta}_d))^{-1}(\hat{\beta}_d - F_d\beta)$$

The standard error of $\hat{\beta}_d$ is computed by considering the variance of the estimator, given in Eq. 2, and then taking the square root of this variance, that is:

$$S.E(\hat{\beta}_{0d}) = \sqrt{Var(\overline{y}) + \overline{X}_j^2\,diag[Cov(\hat{\beta}_d)]} \tag{3}$$

The R package liureg

Our R package **liureg** contains functions related to fitting of the LR model and provides a simple way of obtaining the estimates of LR coefficients, testing the Liu coefficients, and the computation of different Liu related statistics, which prove helpful for selection of optimal biasing parameter d. The package computes different Liu related measures available for the selection of biasing parameter d, and computes value of different biasing parameters proposed by some researchers in the literature.

The `liureg` objects contain a set of standard methods such as `print()`, `summary()`, `plot()`, and `predict()`. Therefore, inferences can be made easily using the `summary` method for assessing the estimates of regression coefficients, their standard errors, t-values and their respective p-values. The default function `liu` which calls `liuest()` to perform required computations and estimation for given

values of non-stochastic biasing parameter d. The syntax of default function is,

```
liu(formula,data,scaling=("centered","sc","scaled"),d,...)
```

The four arguments of `liu()` function are described in Table 4.

Argument	Description
`formula`	Symbolic representation for LR model of the form, response $\sim$ predictors.
`data`	Contains the variables that have to be used in LR model.
`d`	The biasing parameter, may be a scalar or vector. If a d value is not provided, $d = 1$ will be used as the default value, i.e., the OLS results will be produced.
`scaling`	The methods for scaling of predictors. The `centered` option, centers the predictors, suggested by Liu (1993), and uses the default scaling option; the `sc` option scales the predictors in correlation form as described in Belsley (1991); Draper and Smith (1998); and the `scaled` option standardizes the predictors having zero mean and unit variance.

Table 4: Description of `liu()` function arguments.

The `liu()` function returns an object of class `"liu"`. The functions `summary()`, `dest()`, and `lstats()` etc., are used to compute and print a summary of the LR results, list of biasing parameter by Liu (1993, 2011) and Liu related statistics such as estimated squared bias, R^2 and variance etc., after bias is introduced in regression model. An object of class `"liu"` is a list, the components of which are described in Table 5.

Object	Description
coef	A named vector of fitted Liu coefficients.
lfit	Matrix of Liu fitted values for each biasing parameter d.
mf	Actual data used.
xm	A vector of means of design matrix X.
y	The centered response variable.
xscale	The scales used to standardize the predictors.
xs	The scaled matrix of the predictors.
scaling	The method of scaling used to standardize the predictors.
d	The LR biasing parameter(s).
Inter	Whether an intercept is included in the model or not.
call	The matched call.
terms	The `terms` object used.

Table 5: Components of the `"liu"` class.

Table 6 lists the functions and methods available in **liureg** package.

The Liu package implementation in R

The use of **liureg** is explained through examples using the Hald dataset.

```
> library(liureg)
> mod <- liu(y ~ X1 + X2 + X3 + X4, data = as.data.frame(Hald),
+  scaling = "centered", d = seq(0, 1, 0.01) )
```

The output of linear LR from `liu()` function is assigned to an object `mod`. The first argument of the function is `formula`, which is used to specify the required LR model for the data provided as second argument. The `print` method for `mod`, an object of class `"liu"`, will display the de-scaled coefficients. The output (de-scaled coefficients) from the above command is only for a few selected biasing parameter values.

```
Call:
liu.default(formula = y ~ ., data = as.data.frame(Hald), d = c(0,
0.01, 0.49, 0.5, 0.9, 1))
```

Functions	Description
Liu coefficient estimation and testing	
`liuest()`	The main model fitting function for implementation of LR models in R.
`coef()`	Display de-scaled Liu coefficients.
`liu()`	Generic function and default method that calls `liuest()` and returns an object of S3 class `"liu"` with different set of methods to standard generics. It has a print method for display of Liu de-scaled coefficients.
`summary()`	Standard LR output (coefficient estimates, scaled coefficient estimates, standard errors, *t*-value and *p*-values); returns an object of class `"summary.liu"` containing the relative summary statistics. Has a print method.
Residuals, fitted values and prediction	
`predict()`	Produces predicted value(s) by evaluating `liuest()` in the frame `newdata`.
`fitted()`	Displays Liu fitted values for observed data.
`residuals()`	Displays Liu residuals values.
`press()`	Generic function that computes prediction residuals error sum of squares (PRESS) for Liu coefficients.
Methods to estimate d	
`dest()`	Displays various *d* (biasing parameter) values from different authors available in literature and have a print method.
Liu statistics	
`vcov()`	Displays associated Var-Cov matrix with matching Liu parameter *d* values.
`hatl()`	Generic function that displays hat matrix from LR.
`infoliu()`	Generic function that compute information criteria AIC and BIC.
`lstats()`	Generic function that displays different statistics of LR such as MSE, squared bias, R^2 etc. Has a print method.
Liu plots	
`plot()`	Liu coefficient trace plot against biasing parameter *d*.
`plot.biasliu()`	Bias, variance, and MSE plot as a function of *d*.
`plot.infoliu()`	Plot of AIC and BIC against *d*.

Table 6: Functions and methods in **liureg** package.

```
       Intercept      X1      X2       X3       X4
d=0     75.01755 1.41348 0.38190 -0.03582 -0.27032
d=0.01  74.89142 1.41486 0.38318 -0.03445 -0.26905
d=0.49  68.83758 1.48092 0.44475  0.03167 -0.20845
d=0.5   68.71146 1.48229 0.44603  0.03304 -0.20719
d=0.9   63.66659 1.53734 0.49734  0.08814 -0.15669
d=1     62.40537 1.55110 0.51017  0.10191 -0.14406
```

To obtain Liu scaled coefficients `mod$coef` can be used:

```
> mod$coef
```

```
           d=0      d=0.01       d=0.49       d=0.5       d=0.9        d=1
X1  1.41348287  1.41485907  1.48091656  1.48229276  1.53734067  1.5511026
X2  0.38189878  0.38318147  0.44475049  0.44603318  0.49734070  0.5101676
X3 -0.03582438 -0.03444704  0.03166517  0.03304251  0.08813603  0.1019094
X4 -0.27031652 -0.26905396 -0.20845133 -0.20718877 -0.15668658 -0.1440610
```

Objects of class `"liu"` contain components such as `lfit`, `d`, and `coef` etc. For a fitted Liu model, the generic method `summary` is used to investigate the Liu coefficients. The parameter estimates of the Liu model are summarized using a matrix of 5 columns, namely *estimates*, *estimates(Sc)*, *StdErr (Sc)*, *t-values (Sc)*, and *P(>|t|)*. The following results are shown only for `d=-1.47218` which produces a minimum MSE as compared to others values specified in the argument.

```
> summary(mod)
```

```
Call:
liu.default(formula = y ~ ., data = as.data.frame(Hald), d = -1.47218)

Coefficients for Liu parameter d= -1.47218
          Estimate Estimate (Sc) StdErr (Sc) t-val (Sc) Pr(>|t|)
Intercept  93.5849       93.5849     15.6226      5.990 2.09e-09 ***
X1          1.2109        1.2109      0.2711      4.466 7.97e-06 ***
X2          0.1931        0.1931      0.2595      0.744   0.4568
X3         -0.2386       -0.2386      0.2671     -0.893   0.3717
X4         -0.4562       -0.4562      0.2507     -1.820   0.0688 .
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Liu Summary
           R2     adj-R2 F     AIC   BIC   MSE
d=-1.47218 0.9819 0.8372 127.8 23.95 59.18 0.7047
```

The `summary()` function also displays Liu related R^2, adjusted-R^2, F-test, AIC, BIC, and minimum MSE at certain d given in `liu()`.

The `dest()` function, which works with Liu fitted models, computes different biasing parameters developed by researchers, see Table 3. The list of different d values (5 in number) may help in deciding the amount of bias needs to be introduced in LR. The biasing parameters by Liu (1993, 2011) include d_{CL}, d_{mm}, d_{opt}, d_{ILE}, and GCV for the appropriate selection of d.

```
> dest(mod)

Liu biasing parameter d
           d values
dmm        -5.91524
dcl        -5.66240
dopt       -1.47218
dILE       -0.83461
min GCV at  1.00000
```

The `lstats()` function can be used to compute different statistics for a given Liu biasing parameter specified in a call to `liu`. The Liu statistics are MSE, squared bias, F-statistics, Liu variance, degrees of freedom (df) by Hastie and Tibshirani (1990), and R^2 etc. Following are results using `lstats()` for some $d = -1.47218, -0.06, 0, 0.1, 0.5, 1$.

```
> lstats(mod)

Liu Regression Statistics:

               EDF Sigma2     CL    VAR Bias^2    MSE        F     R2 adj-R2
d=-1.47218 9.4135 5.2173 5.0880 0.2750 0.4297 0.7047 127.8388 0.9819 0.8372
d=-0.06    9.0760 5.2989 5.5077 1.0195 0.0790 1.0985 125.8693 0.9823 0.8406
d=0        9.0677 5.3010 5.5315 1.0625 0.0703 1.1328 125.8194 0.9823 0.8407
d=0.1      9.0548 5.3043 5.5722 1.1362 0.0569 1.1931 125.7427 0.9823 0.8408
d=0.5      9.0169 5.3139 5.7488 1.4561 0.0176 1.4737 125.5157 0.9824 0.8412
d=1        9.0000 5.3182 6.0000 1.9119 0.0000 1.9119 125.4141 0.9824 0.8414

minimum MSE occurred at d= -1.47218
```

The `lstats()` also displays the value of d which produces minimum MSE among all provided values of d as argument in `liu()` function.

The residuals, fitted values from the LR, and predicted values of the response variable y can be computed using the fuctions `residuals()`, `fitted()`, and `predict()`, respectively. To obtain the Var-Cov and Hat matrices, the functions `vcov()` and `hatl()` can be used. The df are computed by following Hastie and Tibshirani (1990). The results for Var-Cov and diagonal elements of the hat matrix from `vcov()` and `hatl()` functions are given below for $d = -1.47218$.

```
> vcov(liu(y ~ ., as.data.frame(Hald), d = -1.47218))

$`d=-1.47218`
```

```
           X1         X2         X3         X4
X1 0.07351333 0.04805778 0.06567391 0.04874902
X2 0.04805778 0.06732869 0.05192626 0.06412284
X3 0.06567391 0.05192626 0.07134433 0.05149914
X4 0.04874902 0.06412284 0.05149914 0.06284562

> diag(hatl(liu(y ~ ., as.data.frame(Hald), d = -1.47218)))
         1          2          3          4          5          6          7
0.43522319 0.22023015 0.21341231 0.18535953 0.27191765 0.04296839 0.28798591
         8          9         10         11         12         13
0.30622895 0.15028900 0.59103231 0.30392765 0.14087610 0.18778716
```

Following are possible uses of some functions to compute different Liu related statistics. For a detailed description of these functions/commands, see the **liureg** package documentation.

```
> hatl(mod)
> halt(mod)[[1]]
> diag(hatl(mod)[[1]])
> vcov(mod)
> residual(mod)
> fitted(mod)
> predict(mod)
> lstats(mod)$lEDF
> lstats(mod)$var
```

For given values of X, such as for first five rows of X matrix, the predicted values for some $d = -1.47218, -0.06, 0, 0.1, 0.5, 1$ will be computed by `predict()`:

```
> predict(mod, newdata = as.data.frame(Hald[1 : 5, -1]))

  d=-1.47218   d=-0.06       d=0     d=0.1     d=0.5       d=1
1   78.27798  78.40208  78.40736  78.41615  78.45130  78.49524
2   73.09404  72.91968  72.91227  72.89992  72.85053  72.78880
3  106.68373 106.27656 106.25926 106.23043 106.11510 105.97094
4   89.54007  89.41842  89.41325  89.40463  89.37017  89.32710
5   95.61470  95.63443  95.63527  95.63667  95.64226  95.64924
```

The model selection criteria's of AIC and BIC can be computed using `infoliu()` function for each value of d used in argument of `liu()`. For some $d = -1.47218, -0.06, 0.5, 1$, the AIC and BIC values are:

```
> infoliu(liu(y ~ ., as.data.frame(Hald), d = c(-1.47218, -0.06, 0.5, 1)))

                AIC      BIC
d=-1.47218 23.95378 59.18349
d=-0.06    24.43818 59.88178
d=0.5      24.69007 60.21849
d=1        24.94429 60.54843
```

The effect of multicollinearity on the coefficient estimates can be identified by using different graphical displays such as the Liu trace (see Figure 1); the plotting of bias, variance, and MSE against d (see Figure 2); and plotting the information criteria against df (Figure 3). These graphical displays are (subjective) methods for selection of the optimal biasing parameter d.

```
> mod <- liu(y ~ ., as.data.frame(Hald), d = seq(-5, 5, .001) )
> plot(mod)
> plot.biasliu(mod)
> plot.infoliu(mod)
```

Summary

The **liureg** package provides the most complete suite of tools for LR available in R, comparable to those available as listed in Table 1. We have implemented functions to compute the Liu coefficients, the testing of these coefficients, the computation of different Liu related statistics and the computation of the biasing parameter for different existing methods by various authors (see Table 3). We have

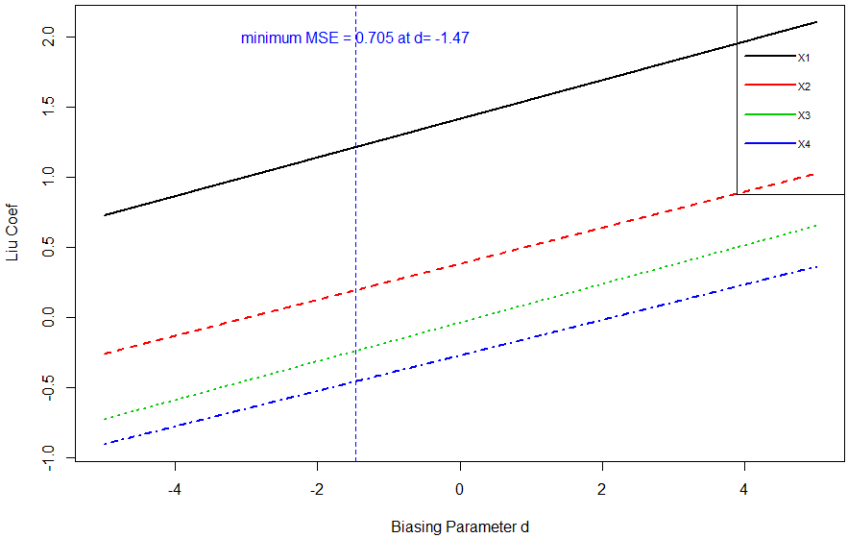


Figure 1: Liu trace: Liu coefficient against biasing parameter d.

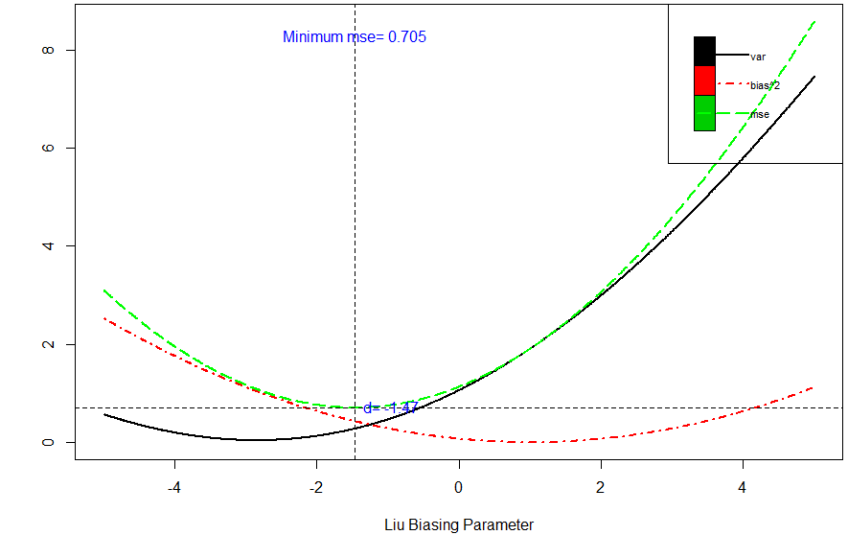


Figure 2: Bias, variance trade-off.

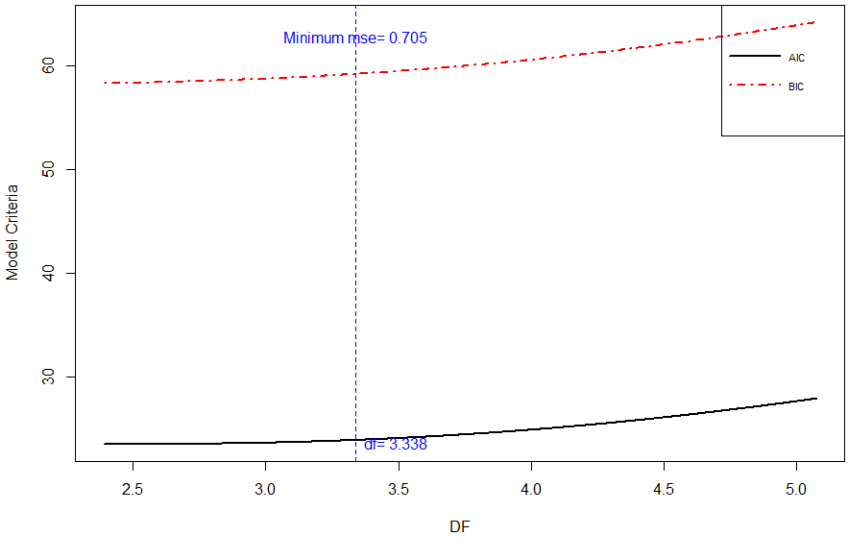


Figure 3: Information Criteria against df.

greatly increased the Liu related statistics and different graphical methods for the selection of the biasing parameter d through the **liureg** package in R.

Up to now, a complete suite of tools for LR was not available for an open source or paid version of statistical software packages, resulting in reduced awareness and use of developed Liu related statistics. The package **liureg** provides a complete open source suite of tools for the computation of Liu coefficients estimation, testing, and computation of different statistics. We believe the availability of these tools will lead to an increased utilization and better Liu related practices.

Bibliography

F. Akdeniz and S. Kaçiranlar. On the Almost Unbiased Generalized Liu Estimators and Unbiased Estimation of the Bias and MSE. *Communications in Statistics-Theory and Methods*, 24(7):1789–1797, 1995. URL http://doi.org/10.1080/03610929508831585. [p235, 236]

F. Akdeniz and S. Kaçiranlar. More on the New Biased Estimator in Linear Regression. *Sankhyā: The Indian Journal of Statistics, Series B (1960-2002)*, 63(3):321–325, 2001. [p235, 236]

F. Akdeniz and R. M. Özkale. The Distribution of Stochastic Shrinkage Biasing Parameter of the Liu Type Estimator. *Applied Mathematics and Computation*, 163(1):29–38, 2005. URL https://doi.org/10.1016/j.amc.2004.03.026. [p236]

F. Akdeniz, G. P. H. Styan, and H. J. Werner. The General Expression for the Moments of the Stochastics Shrinkage Parameters of the Liu Type Estimator. *Communications in Statistics-Theory and Methods*, 35 (3):423–437, 2006. URL http://doi.org/10.1080/03610920500476572. [p236]

O. Arslan and N. Billor. Robust Liu Estimator for Regression Based on an M-Estimator. *Journal of Applied Statistics*, 27(1):39–47, 2000. URL http://doi.org/10.1080/02664760021817. [p236]

M. Aslam. Using Heteroscedasticity-Consistent Standard Errors for the Linear Regression Model with Correlated Regressors. *Communications in Statistics-Simulation and Computation*, 43(10):2353–2373, 2014. URL http://doi.org/10.1080/03610918.2012.750354. [p237]

D. A. Belsley. A Guide to using the Collinearity Diagnostics. *Computer Science in Economics and Management*, 4(1):33–50, 1991. URL https://doi.org/10.1007/BF00426854. [p239]

D. A. Belsley, E. Kuh, and R. E. Welsch. *Diagnostics: Identifying Influential Data and Sources of Collinearity*. John Wiley & Sons, New York, 1980. chap. 3. [p234]

S. Chatterjee and A. S. Hadi. *Regression Analysis by Example*. John Wiley & Sons, 4th edition, 2006. [p234]

E. Cule and M. De Iorio. A Semi-Automatic Method to Guide the Choice of Ridge Regression. *Annals of Applied Statistics*, arxiv:1205.0686v1, 2012. URL https://arxiv.org/abs/1205.0686v1. [p232, 237]

J. D. Curto and J. C. Pinto. The Corrected VIF (CVIF). *Journal of Applied Statistics*, 38(7):1499–1507, 2011. URL http://doi.org/10.1080/02664763.2010.505956. [p234]

A. Dissanayake and P. Wijekoon. *lrmest: Different Types of Estimators to Deal with Multicollinearity*, 2016. URL https://CRAN.R-project.org/package=lrmest. R package version 3.0. [p233]

N. R. Draper and H. Smith. *Applied Regression Analysis*. John Wiley & Sons, New York, 2nd edition, 1998. [p239]

P. Druilhet and A. Mom. Shrinkage Structure in Biased Regression. *Journal of Multivariate Analysis*, 99 (2):232–244, 2008. URL https://doi.org/10.1016/j.jmva.2006.06.011. [p235]

D. E. Farrar and R. R. Glauber. Multicollinearity in Regression Analysis: The Problem Revisted. *The Review of Economics and Statistics*, 49(1):92–107, 1967. URL http://doi.org/10.2307/1937887. [p234]

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. [p234]

G. Golub, G. Wahba, and C. Heath. Generalized Cross Validation as a Method for Choosing a Good Ridge Parameter. *Technometrics*, 21(2):215–223, 1979. URL http://doi.org/10.2307/1268518. [p232]

W. H. Greene. *Econometric Analysis*. Prentice Hall, New Jersey, 5th edition, 2002. [p234]

M. Gruber. *Improving Efficiency by Shrinkage: The James-Stein and Ridge Regression Estimator*. Marcel Dekker, Inc., New York, 1998. [p235]

R. F. Gunst and R. L. Mason. Advantages of Examining Multicollinearities in Regression Analysis. *Biometrics*, 33(1):249–260, 1977. URL http://doi.org/10.2307/2529320. [p234]

A. M. Halawa and M. Y. El-Bassiouni. Tests of Regression Coefficients Under Ridge Regression Models. *Journal of Statistical-Computation and Simulation*, 65(1–4):341–356, 2000. URL http://doi.org/10.1080/00949650008812006. [p237]

A. Hald. *Statistical Theory with Engineering Applications*. John Wiley & Sons, New York, 1952. [p234]

T. Hastie and R. Tibshirani. *Generalized Additive Models*. Chapman & Hall, 1990. [p237, 241]

A. Hoerl and R. W. Kennard. Ridge Regression: Biased Estimation of Nonorthogonal Problems. *Technometrics*, 12(1):55–67, 1970. URL http://doi.org/10.1080/00401706.1970.10488634. [p232, 235]

A. E. Hoerl, R. W. Kennard, and K. F. Baldwin. Ridge Regression: Some Simulations. *Communications in Statistics*, 4(2):105–123, 1975. URL http://doi.org/10.1080/03610927508827232. [p234]

M. Hubert and P. Wijekoon. Improvement of the Liu Estimator in Linear Regression Model. *Journal of Statistical Papers*, 47(3):471–479, 2006. URL https://doi.org/10.1007/s00362-006-0300-4. [p235]

M. Imdadullah and D. M. Aslam. *lmridge: Linear Ridge Regression with Ridge Penalty and Ridge Statistics*, 2016a. URL https://CRAN.R-project.org/package=lmridge. R package version 1.0. [p232]

M. Imdadullah and D. M. Aslam. *liureg: Liu Regression with Liu Biasing Parameters and Statistics*, 2017. URL https://CRAN.R-project.org/package=liureg. R package version 1.1.1. [p232]

M. Imdadullah and M. Aslam. *mctest: Multicollinearity Diagnostic Measures*, 2016b. URL https://CRAN.R-project.org/package=mctest. R package version 1.1. [p233, 234]

M. Imdadullah, M. Aslam, and S. Altaf. mctest: An R Package for Detection of Collinearity Among Regressors. *The R Journal*, online published paper, 2016. URL https://journal.r-project.org/archive/accepted/imdadullah-aslam-altaf.pdf. [p234]

A. Jahufer and J. Chen. Assessing Global Influential Observations in Modified Ridge Regression. *Statistics and Probability Letters*, 79(4):513–518, 2009. URL https://doi.org/10.1016/j.spl.2008.09.019. [p235]

A. Jahufer and J. Chen. Measuring Local Influential Observations in Modified Ridge Regression. *Journal of Data Science*, 9(3):359–372, 2011. [p235]

A. Jahufer and J. Chen. Identifying Local Influential Observations in Liu Estimator. *Journal of Metrika*, 75(3):425–438, 2012. URL https://doi.org/10.1007/s00184-010-0334-4. [p235]

S. Kaçiranlar and S. Sakalhoğlu. Combining the LIU Estimator and the Principal Component Regression Estimator. *Communications in Statistics-Theory and Methods*, 30(12):2699–2706, 2001. URL http://doi.org/10.1081/STA-100108454. [p235, 236]

S. Kaçiranlar, S. Sakalhoğlu, F. Akdeniz, G. Styan, and H. Werner. A new Biased Estimator in Linear Regression and a Detailed Analysis of the Widely-Analysed Dataset on Portland Cement. *Sankhyā: The Indian Journal of Statistics, Series B*, 61(B3):443–459, 1999. [p235, 236]

B. Kan, O. Alpu, and B. Yazici. Robust Ridge and Liu Estimator for Regression Based on the LTS Estimator. *Journal of Applied Statistics*, 40(3):644–665, 2013. URL http://doi.org/10.1080/02664763.2012.750285. [p233]

M. G. Kendall. *A Course in Multivariate Analysis*. Griffin, London, 1957. pp. 70–75. [p234]

L. R. Klein. *An Introduction to Econometrics*. Prentice-Hall, Englewood, Cliffs, N. J., 1962. pp. 101. [p234]

J. Kmenta. *Elements of Econometrics*. Macmillan Publishing Company, New York, 2nd edition, 1980. pp. 431. [p232]

A. Koutsoyiannis. *Theory of Econometrics*. Macmillan Education Limited, 1977. [p234]

P. Kovács, T. Petres, and Tóth. A new Measure of Multicollinearity in Linear Regression Models. *International Statistical Review / Revue Internationale de Statistique*, 73(3):405–412, 2005. URL http://doi.org/10.1111/j.1751-5823.2005.tb00156.x. [p234]

F. Leisch. *Creating R Packages: A Tutorial*. Compstat 2008-Proceedings in Computational Statistics, Physica Verlage, Heidelberg, Germay, 2008, 2008. URL ftp://cran.r-project.org/pub/R/doc/contrib/Leisch-CreatingPackages.pdf. [p232]

K. Liu. A new Class of Biased Estimate in Linear Regression. *Communications in Statistics-Theory and Methods*, 22(2):393–402, 1993. URL http://doi.org/10.1080/03610929308831027. [p232, 235, 236, 238, 239, 241]

X.-Q. Liu. Improved Liu Estimator in a Linear Regression Model. *Journal of Statistical Planning and Inference*, 141(1):189–196, 2011. URL https://doi.org/10.1016/j.jspi.2010.05.030. [p235, 238, 239, 241]

G. S. Maddala. *Introduction to Econometrics*. Macmillan, New York, 1988. [p234]

C. L. Mallows. Some Comments on Cp. *Technometrics*, 15(4):661–675, 1973. URL http://doi.org/10.2307/1267380. [p232, 238]

D. W. Marquardt. Generalized Inverses, Ridge Regression, Biased Linear Estimation, and Nonlinear Estimation. *Technometrics*, 12(3):591–612, 1970. URL http://doi.org/10.2307/1267205. [p234]

A. McLeod and C. Xu. *bestglm: Best Subset GLM*, 2017. URL https://CRAN.R-project.org/package=bestglm. R package version 0.36. [p232]

D. C. Montgomery and E. A. Peck. *Introduction to Linear Regression Analysis*. John Wiley & Sons, New York, 1982. [p232]

R. H. Myers. *Classical and Modern Regression with Application*. PWS-KENT Publishing Company, 2 edition, 1986. [p232]

R. M. Özkale and S. Kaçiranlar. A Prediction-Oriented Criterion for Choosing the Biasing Parameter in Liu Estimation. *Communications in Statistics-Theory and Methods*, 36(10):1889–1903, 2007. URL http://doi.org/10.1080/03610920601126522. [p236, 238]

J. O. Rawlings, S. G. Pantula, and D. A. Dickey. *Applied Regression Analysis: A Research Tool*. Springer-Verlag, New York, 2nd edition, 1998. [p232]

S. Sakalhoğlu, S. Kaçiranlar, and F. Akdeniz. Mean Squared Error Comparisons of Some Biased Regression Estimators. *Communications in Statistics-Theory and Methods*, 30(2):347–361, 2001. URL http://doi.org/10.1081/STA-100002036. [p236]

G. A. F. Seber and A. J. Lee. *Linear Regression Analysis*. John Wiley & Sons, New Jersey, 2 edition, 2003. [p232]

C. Stein. Inadmissibility of Usual Estimator for the Mean of a Multivariate Normal Distribution. In *Proc. Third Berkeley Symp. Mathemat. Statist, Probab.*, pages 197–206, Berkeley, 1956. University of California Press. [p235]

R. C. Team. *Writing R Extensions*. R Foundation for Statistical Computing, 2015. Version R 3.2.3. [p232]

H. Theil. *Principles of Econometrics*. John Wiley & Sons, New York, 1971. [p234]

N. Torigoe and K. Ujiie. On the Restricted Liu Estimator in the Gauss-Markov Model. *Communications in Statistics–Theory and Methods*, 35(9):1713–1722, 2006. URL http://doi.org/10.1080/03610920600683754. [p235]

R. E. Tripp. *Non-stochastic Ridge Regression and Effective Rank of the Regressors Matrix*. Ph.d. thesis, Department of Statistic, Virginia Polytechnic Institute and State University., 1983. [p232]

W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S*. Springer-Verlag, New York, fourth edition, 2002. URL http://www.stats.ox.ac.uk/pub/MASS4. ISBN 0-387-95457-0. [p232]

E. Walker and J. B. Birch. Influence Measures in Ridge Regression. *Technometrics*, 30(2):221–227, 1988. URL http://doi.org/10.2307/1270168. [p236]

H. Wickham. *R Packages: Organize, Test, Document, and Share Your Code*. O'Reilly Media, 2015. [p232]

Muhammad Imdadullah
Ph.D scholar (Statistics),
Department of Statistics
Bahauddin Zakariya University, Multan, Pakistan
ORCiD ID: 0000-0002-1315-491X
mimdadasad@gmail.com

Muhammad Aslam
Associate Professor,
Department of Statistics
Bahauddin Zakariya University, Multan, Pakistan
aslamasadi@bzu.edu.pk

Saima Altaf
Assistant Professor,
Department of Statistics
Bahaudding Zakariya University, Multan, Pakistan
drsaimaaltaf27@gmail.com

A Tidy Data Model for Natural Language Processing using cleanNLP

by Taylor Arnold

Abstract Recent advances in natural language processing have produced libraries that extract low-level features from a collection of raw texts. These features, known as annotations, are usually stored internally in hierarchical, tree-based data structures. This paper proposes a data model to represent annotations as a collection of normalized relational data tables optimized for exploratory data analysis and predictive modeling. The R package **cleanNLP**, which calls one of two state of the art NLP libraries (CoreNLP or spaCy), is presented as an implementation of this data model. It takes raw text as an input and returns a list of normalized tables. Specific annotations provided include tokenization, part of speech tagging, named entity recognition, sentiment analysis, dependency parsing, coreference resolution, and word embeddings. The package currently supports input text in English, German, French, and Spanish.

Introduction

There has been an ongoing trend towards converting raw data into a collection of normalized tables prior to conducting further analyses. This paradigm, recently popularized by Hadley Wickham under the term "tidy data" (Wickham, 2014), draws on concepts from database and visualization theory to provide a welcomed theoretical basis for data analysis. There are also many pragmatic benefits to putting data into a set of normalized tables prior to beginning an exploratory analysis or building inferential models. When working with normalized data most modeling, data manipulation, and visualization tasks can be described using a small collection of functions. This makes code more readable, less-error prone, and allows for better code reuse. As many of these simple functions reduce to basic database operations, this style of coding can simplify the task of integrating statistical models into a production codebase. Also, normalized tables can be stored unambiguously as delimited plain text flat files, allowing for interoperability between programming languages and users.

As both a cause and result of the popularity of this approach, a number of software packages have been developed to help construct and manipulate collections of normalized data tables. In R, well-known examples include **dplyr** (Wickham and Francois, 2016), **ggplot2** (Wickham, 2009), **magrittr** (Bache and Wickham, 2014), **broom** (Robinson, 2017), **janitor** (Firke, 2016), and **tidyr** (Wickham, 2017). On the Python side, much of this functionality is included within the pandas (McKinney et al., 2010) and sklearn (Pedregosa et al., 2011) modules.

While cleaning messy data is often a time-consuming task, deciding on a specific normalized schema for representing a set of inputs is in most cases relatively straightforward. Outside of potentially removing outliers, missing data, and bad inputs, the process of tidying data is generally a lossless procedure. At a high-level, data tidying is often simply a reorganization of the raw inputs. However, if we are working with unstructured data such as collections of text, images, or sound, converting into a normalized tabular format is significantly more involved. The process of tidying in these cases becomes synonymous with featurization, whereby structured outputs are algorithmically extracted from a raw input. For example, from an audio music file we might extract features such as the overall length, beats per minute, and quantiles of the music's loudness.

The featurization of raw text, known in natural language processing as *text annotation*, includes tasks such as tokenization (splitting text into words), part of speech tagging, and named entity recognition. Recent advancements in neural networks and heavy investment from both industry and academia have produced fast and highly accurate annotation libraries such as Stanford's CoreNLP (Manning et al., 2014), spaCy (Honnibal and Johnson, 2015), Apache's OpenNLP (Baldridge, 2005), and Google's SyntaxNet (Petrov, 2016). All of these, however, internally represent annotations using collections of complex, hierarchical, object-oriented classes. While these structures are ideal for annotation, they are not optimal for exploratory and predictive modeling.

In this paper, we present a method for uniting the cutting edge advancements in natural language processing with the popular normalized data paradigm. Specifically, we give a data schema representing the output of an NLP annotation pipeline as a collection of normalized tables. Alongside this specification, we present the R package **cleanNLP** that implements this specification over three distinct back ends. The package contains:

- custom Java code, called by **rJava** (Urbanek, 2016), that annotates raw text using the CoreNLP library;

- a custom Python script, called by **reticulate** (Allaire et al., 2017), that annotates raw text using the spaCy library;
- a simple, system dependency free, annotation engine using the package **tokenizers** (Mullen, 2016).

The package **cleanNLP** also includes tools for converting from the normalized data model into (sparse) data matrices appropriate for exploratory and predictive modeling. Together, these contributions simplify the process of doing exploratory data analysis over a corpus of text.

There are several existing R packages that have some similar or complementary features to those in **cleanNLP**. The R package **tidytext** (Silge and Robinson, 2016) also offers the ability to convert raw rext into a data frame. It is quite similar to the functionality of **cleanNLP** when using the **tokenizers** back end, with the addition of basic sentiment analysis and part of speech tagging for English through the use of word lists. With all annotations occurring at the token level, results are given as a single table rather than a normalized schema between many tables as in **cleanNLP**, which simplifies its application for new users. As such, **tidytext** works well for applications that do not need more advanced annotators such as named entities, dependencies, and coreferences. Given the overlap in general approaches, it should be relatively straightforward for users to transition from **tidytext** to **cleanNLP** when they find the need for these annotation tasks. There are two existing R packages that also call functions in the CoreNLP library. The package **StanfordCoreNLP** (Hornik, 2016c), available only through the datacube website at Vienna University, integrates into the **NLP** framework. A similar, standalone approach is offered by **coreNLP** (Arnold and Tilton, 2016). Both of these packages run the annotation pipeline over a corpus of text, call the java class `edu.stanford.nlp.pipeline.XMLOutputter`, and then parse the output using the **XML** package. This approach is not ideal as parsing the output XML file is computationally time-consuming. It is also error prone because there is no published format specifying the output of the XML.[1] There is also the package **spacyr** (Benoit and Matsuo, 2017), which was published after **cleanNLP**, that offers another way of calling the spaCy library from R. Internally, **spacyr** works similarly to the spaCy back end in **cleanNLP** by calling the Python library and extracting information into R data types. However, **spacyr** returns results as a single denormalized data frame and (perhaps in part as a result of having no easy way of storing them in the one-table output) does not support the word embeddings feature of the spaCy library.

The package has been designed to integrate into workflows that utilize the many other packages for text processing available in R, such as those found in the CRAN Taskview *NaturalLanguageProcessing*. For example, users may use the framework provided by **tm** (Feinerer et al., 2008) to manage external corpora or the classes within **NLP** (Hornik, 2016a) to run alternative parsers that can be converted into a tidy framework by way of the `from_CoNLL` function. The Apache OpenNLP annotation pipeline, available via **openNLP** (Hornik, 2016b), for instance, provides several languages not yet supported by spaCy or the CoreNLP pipeline. Packages that focus on the analysis and modeling of text data can usually be used directly with the output from **cleanNLP**; these include **lda** (Chang, 2015), **lsa** (Wild, 2015), and **topicmodels** (Grün and Hornik, 2011). Similarly, general-purpose database back-ends such as **sqliter** (Freitas, 2014) can be used to store the tidy data tables; predictive modeling functions may be used to do predictive analytics over generated term-frequency matrices.

In the following section we illustrate the usage of the R package across all three back ends. Next, we give a detailed description and justification of our data model. Along the way, we give a high-level introduction to the ideas behind the underlying NLP annotators. We finish by illustrating a longer example of using the package to study a corpus of historical speeches made by Presidents of the United States.

Basic usage of cleanNLP

Before describing the data model for text annotations, it is useful to understand the basic workflow provided by the R package **cleanNLP**. We start by writing the opening lines of Douglas Adams' *Life, the Universe and Everything* to a temporary file.

```
> txt <- c("The regular early morning yell of horror was the sound of Authur",
+          "Dent waking up and suddenly remembering where he was. It wasn't",
+          "just that the cave was cold, it wasn't just that it was damp and",
+          "smelly. It was the fact that the cave was in the middle of",
```

[1] This author, who is also the maintainer of **coreNLP**, has witnessed this first-hand by way of the persistent bug reports centering around the formatting of the XML output in strange edge cases or over new versions of the CoreNLP library. The **coreNLP** will still be maintained for users looking explicitly to access methods from the Stanford Library, whereas **cleanNLP** is being developed to provide a simpler interface that is consistent across various back ends.

```
+          "Islington and there wasn't a bus due for two million years.")
> writeLines(txt, tf <- tempfile())
```

The package **cleanNLP** can be installed directly from CRAN, with binaries available for all major operating systems. In order to annotate raw text, an NLP back end must first be initalized. Once this is done, annotation is done by calling the function `annotate` with a vector of path(s) to the input documents. We start with an example using the tokenizers back end.

```
> library(cleanNLP)
> init_tokenizers()
> anno <- run_annotators(tf)
```

The result of the annotation is a named list of six data frames and one matrix. We can see the elements of the object by printing out their names.

```
> names(anno)
[1] "coreference" "dependency"  "document"    "entity"      "sentence"
[6] "token"       "vector"
```

The individual tables can be referenced with the generic R accessor functions (such as `` `[[` ``), however the preferred method is to call the relevant **cleanNLP** functions of the form `get_TABLENAME()`. For example, the tokens table for this example can be accessed with the `get_token` function.

```
> get_token(anno)
# A tibble: 61 x 8
      id   sid   tid    word lemma  upos   pos   cid
   <int> <int> <int>   <chr> <chr> <chr> <chr> <int>
1      1     0     1     The  <NA>  <NA>  <NA>    NA
2      1     0     2 regular  <NA>  <NA>  <NA>    NA
3      1     0     3   early  <NA>  <NA>  <NA>    NA
4      1     0     4 morning  <NA>  <NA>  <NA>    NA
5      1     0     5    yell  <NA>  <NA>  <NA>    NA
6      1     0     6      of  <NA>  <NA>  <NA>    NA
7      1     0     7  horror  <NA>  <NA>  <NA>    NA
8      1     0     8     was  <NA>  <NA>  <NA>    NA
9      1     0     9     the  <NA>  <NA>  <NA>    NA
10     1     0    10   sound  <NA>  <NA>  <NA>    NA
# ... with 51 more rows
```

The `get` functions are preferable because they provide useful options for modifying the output before returning it. Notice that the annotation process here has split out each word in the input into its own row. There are also several columns of ids and columns filled with missing values. The specific schema of the tables will be the focus of discussion in the following section.

The tokenizers back end requires no external dependencies, however it does not support any of the advanced annotation tasks that illustrate the utility of the **cleanNLP** package. This explains why most of the columns in the example are missing. It is included primarily for testing and demonstration purposes in cases where the other back ends cannot be installed. The spaCy back end uses the Python library by the same name for the purpose of extracting text annotations. Users must install Python and the library externally (detailed instructions are provided in the package documentation). Once installed, the only modification required by the R code is to adjust which `init_` function is being called.

```
> init_spaCy()
> anno <- run_annotators(tf)
> get_token(anno)
# A tibble: 68 x 8
      id   sid   tid    word   lemma  upos   pos   cid
   <int> <int> <int>   <chr>   <chr> <chr> <chr> <int>
1      1     1     1     The     the   DET    DT     0
2      1     1     2 regular regular   ADJ    JJ     4
3      1     1     3   early   early   ADJ    JJ    12
4      1     1     4 morning morning  NOUN    NN    18
5      1     1     5    yell    yell  NOUN    NN    26
6      1     1     6      of      of   ADP    IN    31
7      1     1     7  horror  horror  NOUN    NN    34
8      1     1     8     was      be  VERB   VBD    41
9      1     1     9     the     the   DET    DT    45
```

table name	record	primary key	foreign keys
document	document	id	·
token	word / punctuation	id, sid, tid	cid
dependencies	token pairs	id, sid, tid, tid_target	·
entity	set of tokens	id, sid, tid, tid_end	·
coreference	mentions	id, rid, mid	sid, tid, tid_end, tid_head
sentence	sentence	id, sid	·
vector	word embedding	id, sid, tid	·

Table 1: Tables in the data model and their (composite) primary and foreign keys. All keys are given by non-negative integers. Namely, `id` indexes the documents, `sid` the sentences within a document, and `tid` the tokens within a sentence. The `cid` gives character offsets into the raw input text. Keys `rid` and `mid` are specifically constructed by the coreference annotator.

```
10     1     1    10   sound   sound  NOUN    NN    49
# ... with 58 more rows
```

The output is in the exact some format but now all of the token columns are filled in with useful information such as the lemmatized form of each word and part of speech codes. Similar details are also filled into the other fields.

The third and final back end currently available uses the Java library coreNLP. Users must install Java version 1.8 or higher and link it to R using the **rJava**. The coreNLP models, which are over 1 GB, can then be either manually downloaded or grabbed using the helper function `download_coreNLP()`. Once installed, the back end works just as with the other back ends.

```
> init_coreNLP()
> anno <- run_annotators(tf)
> get_token(anno)
# A tibble: 68 x 8
      id   sid   tid    word   lemma  upos   pos   cid
   <int> <int> <int>   <chr>   <chr> <chr> <chr> <int>
1      1     1     1     The     the   DET    DT     0
2      1     1     2 regular regular   ADJ    JJ     4
3      1     1     3   early   early   ADJ    JJ    12
4      1     1     4 morning morning  NOUN    NN    18
5      1     1     5    yell    yell  VERB    VB    26
6      1     1     6      of      of   ADP    IN    31
7      1     1     7  horror  horror  NOUN    NN    34
8      1     1     8     was      be  VERB   VBD    41
9      1     1     9     the     the   DET    DT    45
10     1     1    10   sound   sound  NOUN    NN    49
# ... with 58 more rows
```

The token output here is similar, but not exactly the same, as that produced by the spaCy annotation engine. The only distinction in the first ten rows is whether the word *yell* is categorized as a noun (spaCy) or a verb (coreNLP). While yell can be either part of speech, in context the spaCy interpretation is correct.

As seen in the code-snippets here, the philosophy behind the design of the **cleanNLP** package is to make it as easy as possible to get raw text turned into data frames. All of the functions introduced here have optional parameters that change the way the back ends are run or how the annotations are returned. This includes which annotators to run and selecting the desired language model to use. Complete documentation is available within the R help pages.

A data model for the NLP pipeline

An annotation object is simply a named list with each item containing a data frame. These frames should be thought of as tables living inside of a single database, with keys linking each table to one another. All tables are in the second normal form of Codd (1990). For the most part they also satisfy the third normal form, or, equivalently, the formal tidy data model of Wickham (2014). The limited departures from this more stringent requirement are justified below wherever they exist. In every case the cause is a transitive dependency that would require a complex range join to reconstruct.

Several standards have previously been proposed for representing textual annotations. These

	get_document()
id	integer. Id of the source document.
time	date time. The time at which the parser was run on the text.
version	character. Version of the NLP library used to parse the text.
language	character. Language of the text, in ISO 639-1 format.
uri	character. Description of the raw text location.

Table 2: Schema for the document table. The id field serves as a primary key, and other meta data fields may be appended that give domain-specific information about each document.

include the linguistic Annotation Framework (Ide and Romary, 2001), NLP Interchange Format (Hellmann et al., 2012), and CoNLL-X (Buchholz and Marsi, 2006). The function `from_CoNLL` is included as a helper function in **cleanNLP** to convert from CoNLL formats into the **cleanNLP** data model. All of these, however, are concerned with representing annotations for interoperability between systems. Our goal is instead to create a data model well-suited to direct analysis, and therefore requires a new approach.

In this section each table is presented and justifications for its existence and form are given. Individual tables may be pulled out with access functions of the form `get_*`. Example tables are pulled from the dataset `obama`, which is included with the **cleanNLP** package. This gives the annotation object obtained from the text of the annual speeches Barack Obama made to Congress. These annual addresses, known as *The State of the Union*, are mandated by the US Constitution and have been given by every president since George Washington.

Documents

The documents table contains one row per document in the annotation object. What exactly constitutes a document is up to the user. It might include something as granular as a paragraph or as coarse as an entire novel. For many applications, particularly stylometry, it may be useful to simultaneously work with several hierarchical levels: sections, chapters, and an entire body of work. The solution in these cases is to define a document as the smallest unit of measurement, denoting the higher-level structures as metadata. For example, when working with a corpus of texts where each book is broken into chapters, we would make each document an individual chapter. A metadata field would be assigned to each chapter indicating which book it is a part of.

The primary key for the document table is a document id, stored as an integer index. By design, there should be no extrinsic meaning placed on this key. Other tables use it to map to one another and to the document table, but any metadata *about* the document is contained only in the document table rather than being forced into the document key. In other words, the temptation to use keys such as "Obama2016" is avoided because, while these look nice, trying to make use of them to extract document-level metadata is error prone and ultimately more verbose than making use of a join with the document table.

The minimal fields required by the document table are given in Table 2. These are all filled in automatically by the annotation function. Any number of additional corpora-specific metadata, such as the aforementioned section and chapter designations, may be attached as well by giving it as an option to the `meta` parameter of `run_annotators`. The document table for the example corpus is:

```
> get_document(obama)
# A tibble: 8 x 5
     id                time version language      uri
  <int>              <dttm>   <chr>    <chr>    <chr>
1     1 2017-05-21 09:27:55   1.8.2       en 2009.txt
2     2 2017-05-21 09:28:00   1.8.2       en 2010.txt
3     3 2017-05-21 09:28:05   1.8.2       en 2011.txt
4     4 2017-05-21 09:28:10   1.8.2       en 2012.txt
5     5 2017-05-21 09:28:14   1.8.2       en 2013.txt
6     6 2017-05-21 09:28:18   1.8.2       en 2014.txt
7     7 2017-05-21 09:28:22   1.8.2       en 2015.txt
8     8 2017-05-21 09:28:26   1.8.2       en 2016.txt
```

It may seem that common fields such as year and author should be added to the formal specification but the perceived advantage is minimal. It would still be necessary for users to manually add the content of these fields at some point as any other metadata is not unambiguously extractable from the raw text.

	get_token()
id	integer. Id of the source document.
sid	integer. Sentence id, starting from 0.
tid	integer. Token id, with the root of the sentence starting at 0.
word	character. Raw word in the input text.
lemma	character. Lemmatized form the token.
upos	character. Universal part of speech code.
pos	character. Language-specific part of speech code; uses the Penn Treebank codes.
cid	integer. Character offset at the start of the word in the original document.

Table 3: Schema for the token table. The fields id, sid, and tid serve as a composite key for each token. A row also exist for the root of each sentence.

Tokens

The token table contains one row for each unique token, usually a word or punctuation mark, in any document in the corpus. Any annotator that produces an output for each token has its results displayed here. These include the lemmatizer and the part of the speech tagger (Toutanova and Manning, 2000). Table 3 shows the required columns contained in the token table. Given the annotators selected during the pipeline initialization, some of these columns may contain only missing data. A composite key exists by taking together the document id, sentence id, and token id. There is also a foreign key, `cid`, giving the character offset back into the original source document. An example of the table looks like this:

```
> get_token(obama, include_root = TRUE)
# A tibble: 65,758 x 8
      id   sid   tid      word     lemma  upos   pos   cid
   <int> <int> <int>     <chr>     <chr> <chr> <chr> <int>
1      1     1     0      ROOT      ROOT  <NA>  <NA>    NA
2      1     1     1     Madam     madam PROPN   NNP     0
3      1     1     2   Speaker   speaker PROPN   NNP     6
4      1     1     3         ,         , PUNCT     ,    13
5      1     1     4       Mr.       mr. PROPN   NNP    15
6      1     1     5      Vice      vice PROPN   NNP    19
7      1     1     6 President president PROPN   NNP    24
8      1     1     7         ,         , PUNCT     ,    33
9      1     1     8   Members   members PROPN  NNPS    35
10     1     1     9        of        of   ADP    IN    43
# ... with 65,748 more rows
```

A phantom token "ROOT" is included at the start of each sentence (it always has `tid` equal to 0) if the option `include_root` is set to TRUE (it is FALSE by default). This is useful so that joins from the dependency table, which contains references to the sentence root, into the token table have no missing values.

The field `upos` contains the universal part of speech code, a language-agnostic classification, for the token. It could be argued that in order to maintain database normalization one should simply look up the universal part of speech code by finding the language code in the document table and joining a table mapping the Penn Treebank codes to the universal codes. This has not been done for several reasons. First, universal parts of speech are very useful for exploratory data analysis as they contain tags much more familiar to non-specialists such as "NOUN" (noun) and "CONJ" (conjunction). Asking users to apply a three table join just to access them seems overly cumbersome. Secondly, it is possible for users to use other parsers or annotation engines. These may not include granular part of speech codes and it would be difficult to figure out how to represent these if there were not a dedicated universal part of speech field.

Dependencies

Dependencies give the grammatical relationship between pairs of tokens within a sentence (Green et al., 2011; Rafferty and Manning, 2008). As they are at the level of token pairs, they must be represented as a new table. All included fields are described in Table 4. Only one dependency should exist for any pair of tokens; the document id, sentence id, and source and target token ids together serve as a composite key. As dependencies exist only within a sentence, the sentence id does not need to be

	get_dependency()
id	integer. Id of the source document.
sid	integer. Sentence id of the source token.
tid	integer. Id of the source token.
sid_target	integer. Sentence id of the target token.
tid_target	integer. Id of the target token.
relation	character. Language-agnostic universal dependency type.
relation_full	character. Language specific universal dependency type.
word	character. The source word in the raw text.
lemma	character. Lemmatized form of the source word.
word_target	character. The target word in the raw text.
lemma_target	character. Lemmatized form of the target word.

Table 4: Schema for the dependency table. The final four variables are only provided when the option get_token is set to TRUE. The first five fields together create a composite key for the table.

defined separately for the source and target. Dependencies take significantly longer to calculate than the lemmatization and part of speech tagging tasks.

The get_dependency function has an option (set to FALSE by default) to auto join the dependency to the target and source words and lemmas from the token table. This is a common task and involves non-trivial calls to the left_join function making it worthwhile to include as an option. For example, the following code replicates the behavior of get_dependency when set to return words and lemmas:

```
dep <- get_dependency(obama) %>%
  left_join(select(get_token(obama, include_root = TRUE),
                   id, sid, tid, word, lemma),
            by = c("id", "sid", "tid")) %>%
  left_join(select(get_token(obama, include_root = TRUE),
                   id, sid, tid_target = tid,
                   word_target = word, lemma_target = lemma),
            by = c("id", "sid", "tid_target"))
```

The output, equivalently using a call to get_dependency, is given by:

```
> get_dependency(obama, get_token = TRUE)
# A tibble: 62,781 x 10
      id   sid   tid tid_target relation relation_full      word     lemma
   <int> <int> <int>      <int>    <chr>         <chr>     <chr>     <chr>
1      1     1     2          1 compound          <NA>   Speaker   speaker
2      1     1     0          2     ROOT          <NA>      ROOT      ROOT
3      1     1     2          3    punct          <NA>   Speaker   speaker
4      1     1     6          4 compound          <NA> President president
5      1     1     6          5 compound          <NA> President president
6      1     1     2          6    appos          <NA>   Speaker   speaker
7      1     1     6          7    punct          <NA> President president
8      1     1     6          8    appos          <NA> President president
9      1     1     8          9     prep          <NA>   Members   members
10     1     1     9         10     pobj          <NA>        of        of
   word_target lemma_target
         <chr>        <chr>
1       Madam        madam
2     Speaker      speaker
3           ,            ,
4         Mr.          mr.
5        Vice         vice
6   President    president
7           ,            ,
8     Members      members
9          of           of
10   Congress     congress
# ... with 62,771 more rows
```

The word "ROOT" shows up in the first row, which would have been NA had sentence roots not been

	get_entity()
id	integer. Id of the source document.
sid	integer. Sentence id of the entity mention.
tid	integer. Token id at the start of the entity mention.
tid_end	integer. Token id at the end of the entity mention.
entity_type	character. Type of entity.
entity	character. Raw words of the named entity in the text.
entity_normalized	character. Normalized version of the entity.

Table 5: Schema for the entity table. The first three fields serve as a composite key.

explicitly included in the token table.

Our parser produces universal dependencies (De Marneffe et al., 2014), which have a language-agnostic set of relationship types with language-specific subsets pertaining to specific grammatical relationships with a particular language. For the same reasons that both the part of speech codes and universal part of speech codes are included, each of these relationship types have been added to the dependency table.

Named entities

Named entity recognition is the task of finding entities that can be defined by proper names, categorizing them, and standardizing their formats (Finkel et al., 2005). The XML output of the Stanford CoreNLP pipeline places named entity information directly into their version of the token table. Doing this repeats information over every token in an entity and gives no canonical way of extracting the entirety of a single entity mention. We instead have a separate entity table, as is demanded by the normalized database structure, and record each entity mention in its own row. The full set of fields are given in Table 5, with the combination of document id, sentence id, and token id serving as a composite key.

An example of the named entity table is given by:

```
> get_entity(obama)
# A tibble: 3,035 x 6
      id   sid   tid tid_end entity_type              entity
   <int> <int> <int>   <int>       <chr>               <chr>
1      1     1     1       2      PERSON       Madam Speaker
2      1     1     8      10         ORG Members of Congress
3      1     1    12      14         ORG      the First Lady
4      1     1    16      18         GPE   the United States
5      1     1    30      30        TIME             tonight
6      1     1    43      44       EVENT            Chamber,
7      1     2     6       6        NORP           Americans
8      1     4    24      25        DATE           every day
9      1     8    23      23        TIME             tonight
10     1     8    27      27        NORP            American
# ... with 3,025 more rows
```

The categories available in the field `entity_type` are dependent on the specific back end used. When using the coreNLP back end, the entities 'MONEY', 'ORDINAL' 'PERCENT', 'DATE' and 'TIME' also have a normalized form. Entities for the spaCy backend offer more granular distinctions, with a full list contained in the help page for the function `get_entity`. As with the coreference table, a complete representation of the entity is given as a character string due to the difficulty in reconstructing this after the fact from the token table, so the character string has been included as an explicit field.

Coreference

Coreferences link sets of tokens that refer to the same underlying person, object, or idea (Recasens et al., 2013; Lee et al., 2013, 2011; Raghunathan et al., 2010). One common example is the linking of a noun in one sentence to a pronoun in the next sentence. The coreference table describes these relationships but is not strictly a table of coreferences. Instead, each row represents a single mention of an expression and gives a reference id indicating all of the other mentions that it also coreferences. Table 6 gives the entire schema of the coreference table. The document, reference, and mention ids

	get_coreference()
id	integer. Id of the source document.
rid	integer. Relation ID.
mid	integer. Mention ID; unique to each coreference within a document.
mention	character. The mention as raw words from the text.
mention_type	character. One of "LIST", "NOMINAL", "PRONOMINAL", or "PROPER".
number	character. One of "PLURAL", "SINGULAR", or "UNKNOWN".
gender	character. One of "FEMALE", "MALE", "NEUTRAL", or "UNKNOWN".
animacy	character. One of "ANIMATE", "INANIMATE", or "UNKNOWN".
sid	integer. Sentence id of the coreference.
tid	integer. Token id at the start of the coreference.
tid_end	integer. Token id at the start of the coreference.
tid_head	integer. Token id of the head of the coreference.

Table 6: Schema for the coreference table. Each row is best thought of as a coreference mention, rather than the coreference itself.

serve as a composite key for the table. Links back into the token table for the start, end and head of the mention are given as well; these are pushed to the right of the table as they should be considered foreign keys within this table.

An example helps to explain exactly what the coreference table represents:

```
> get_coreference(obama)
# A tibble: 6,982 x 12
      id   rid   mid                      mention mention_type   number  gender
   <int> <int> <int>                        <chr>        <chr>    <chr>   <chr>
1      1  2049     7           the United States       PROPER SINGULAR NEUTRAL
2      1  2049    77 the United States of America       PROPER SINGULAR NEUTRAL
3      1  2049   102                      America       PROPER SINGULAR NEUTRAL
4      1  2049   315                      America       PROPER SINGULAR NEUTRAL
5      1  2049   742                    America 's       PROPER SINGULAR NEUTRAL
6      1  2049   782                      America       PROPER SINGULAR NEUTRAL
7      1  2049   939                      America       PROPER SINGULAR NEUTRAL
8      1  2049   991                      America       PROPER SINGULAR NEUTRAL
9      1  2049  1003                      America       PROPER SINGULAR NEUTRAL
10     1  2049  1045                      America       PROPER SINGULAR NEUTRAL
     animacy   sid   tid tid_end tid_head
       <chr> <dbl> <int>   <int>    <int>
1  INANIMATE     1    16      18       18
2  INANIMATE     8    41      45       43
3  INANIMATE    12     6       6        6
4  INANIMATE    40    12      12       12
5  INANIMATE   103     8       9        8
6  INANIMATE   109     8       8        8
7  INANIMATE   132     5       5        5
8  INANIMATE   138    27      27       27
9  INANIMATE   140    41      41       41
10 INANIMATE   147     4       4        4
# ... with 6,972 more rows
```

Here, these are all mentions of the same underlying entity: The United States of America. There is a special relationship between the reference id `rid` and the mention id `mid`. The coreference annotator selects a specific mention for each reference that gets treated as the canonical mention for the entire class. The mention id for this mention becomes the reference id for the class. This relationship provides a way of identifying the canonical mention within a reference class and a way of treating the coreference table as pairs of mentions rather than individual mentions joined by a given key.

The text of the mention itself is included within the table. This was done because as the mention may span several tokens it would otherwise be very difficult to extract this information from the token table. It is also possible, though not supported in the current CoreNLP pipeline, that a mention could consist of a set of non-contiguous tokens, making this field impossible to otherwise reconstruct.

	get_sentence()
id	integer. Id of the source document.
sid	integer. Sentence id.
sentiment	integer. Predicted sentiment; 0 (very negative) to 4 (very positive).

Table 7: Schema for the setence table. The document and sentence ids serve as a composite key.

Sentence level annoations

The sentiment tagger provided by the CoreNLP pipeline predicts whether a sentence is very negative (0), negative (1), neutral (2), positive (3), or very positive (4) (Socher et al., 2013). There is no native sentiment model currently supported by spaCy. The sentiment output is placed in a separate table because it returns information exclusively at the sentence level, unlike any of the other parsers. The schema, described in Table 7, has the document and sentence ids serving as composite keys, with the only other field being an integer sentiment code. An example of the output can be seen in:

```
> get_sentence(obama)
# A tibble: 2,988 x 3
      id   sid sentiment
   <int> <dbl>     <int>
1      1     1         1
2      1     2         3
3      1     3         1
4      1     4         1
5      1     5         2
6      1     6         1
7      1     7         3
8      1     8         1
9      1     9         1
10     1    10         1
# ... with 2,978 more rows
```

The underlying sentiment model is a neural network. While at the moment few annotators exist at the sentence level, there is currently active research in modeling features that would eventually fit well into this table such as indicators of mood (Gaikwad and Joshi, 2016), levels of sarcasm (Schifanella et al., 2016) or a characterization of the sentence's "style" (Kabbara and Cheung, 2016).

Word vectors

Our final table in the data model stores the relatively new concept of a word vector. Also known as word embeddings, these vectors are deterministic maps from the set of all available words into a high-dimensional, real valued vector space. Words with similar meanings or themes will tend to be clustered together in this high-dimensional space. For example, we would expect apple and pear to be very close to one another, with vegetables such as carrots, broccoli, and asparagus only slightly farther away. The embeddings can often be used as input features when building models on top of textual data. For a more detailed description of these embeddings, see the papers on either of the most well-known examples: GloVe (Pennington et al., 2014) and word2vec (Mikolov et al., 2013). Only the spaCy back end to **cleanNLP** currently supports word vectors; these are turned off by default because they take a significantly large amount of space to store. The embedding model uses the fasttext embeddings (Bojanowski et al., 2016), a modification of the GloVe embeddings, which map words into a 300-dimensional space. To compute the embeddings, set the `vector_flag` parameter of `init_spaCy` to `TRUE` prior to running the annotation.

Word vectors are stored in a separate table from the tokens table out of convenience rather than as a necessity of preserving the data model's normalized schema. Due to its size and the fact that the individual components of the word embedding have no intrinsic meaning, this table is stored as a matrix. We can see that there is exactly one row in the word embeddings for every non-ROOT token in the token table (note that the word embeddings for the `obama` dataset are not included with the package as they are too large to be uploaded to CRAN).

```
> dim(get_token(obama))
[1] 62781     8
> dim(get_vector(obama))
[1] 62781   303
```

The first three columns hold the keys `id`, `sid`, and `tid`, respectively. If no embedding is computed, the function `get_vector` returns an empty matrix.

Using cleanNLP to study State of the Union addresses

The President of the United States is constitutionally obligated to provide a report known as the *State of the Union*. The report summarizes the current challenges facing the country and the president's upcoming legislative agenda. While historically the State of the Union was often a written document, in recent decades it has always taken the form of an oral address to a joint session of the United States Congress. In this final section the utility of the package is illustrated by showing how it can be used to study a corpus consisting of every such address made by a United States president through 2016 (Peters, 2016). It highlights some of the major benefits of the tidy data model as it applies to the study of textual data, though by no means attempts to give an exhaustive coverage of all the available tables and approaches. The examples make heavy use of the table verbs provided by **dplyr**, the piping notation of **magrittr** and **ggplot2** graphics. These are used because they best illustrate the advantages of the tidy data model that has been built in **cleanNLP** for representing corpus annotations. Relevant functions are prepended with `cleanNLP::` in the following analysis in order to be clear which functions are supplied by the **cleanNLP** package.

Loading and parsing the data

The full text of all the State of the Union addresses through 2016 are available in the R package **sotu** (Arnold, 2017), available on CRAN. The package also contains meta-data concerning each speech that we will add to the document table while annotating the corpus. The code to run this annotation is given by:

```
> library(sotu)
> library(cleanNLP)
>
> data(sotu_text)
> data(sotu_meta)
> init_spaCy()
> sotu <- cleanNLP::run_annotators(sotu_text, as_strings = TRUE,
+                                  meta = sotu_meta)
```

The annotation object, which we will use in the example in the following analysis, is stored in the object `sotu`.

Exploratory analysis

Simple summary statistics are easily computed off of the token table. To see the distribution of sentence length, the token table is grouped by the document and sentence id and the number of rows within each group are computed. The percentiles of these counts give a quick summary of the distribution.

```
> library(ggplot2)
> library(dplyr)
> cleanNLP::get_token(sotu) %>%
+   count(id, sid) %$%
+   quantile(n, seq(0,1,0.1))
  0%  10%  20%  30%  40%  50%  60%  70%  80%  90% 100%
   1   11   16   19   23   27   31   37   44   58  681
```

The median sentence has 28 tokens, whereas at least one has over 600 (this is due to a bulleted list in one of the written addresses being treated as a single sentence) To see the most frequently used nouns in the dataset, the token table is filtered on the universal part of speech field, grouped by lemma, and the number of rows in each group are once again calculated. Sorting the output and selecting the top 42 nouns, yields a high level summary of the topics of interest within this corpus.

```
> cleanNLP::get_token(sotu) %>%
+   filter(upos == "NOUN") %>%
+   count(lemma) %>%
+   top_n(n = 42, n) %>%
+   arrange(desc(n)) %>%
+   use_series(lemma)
```

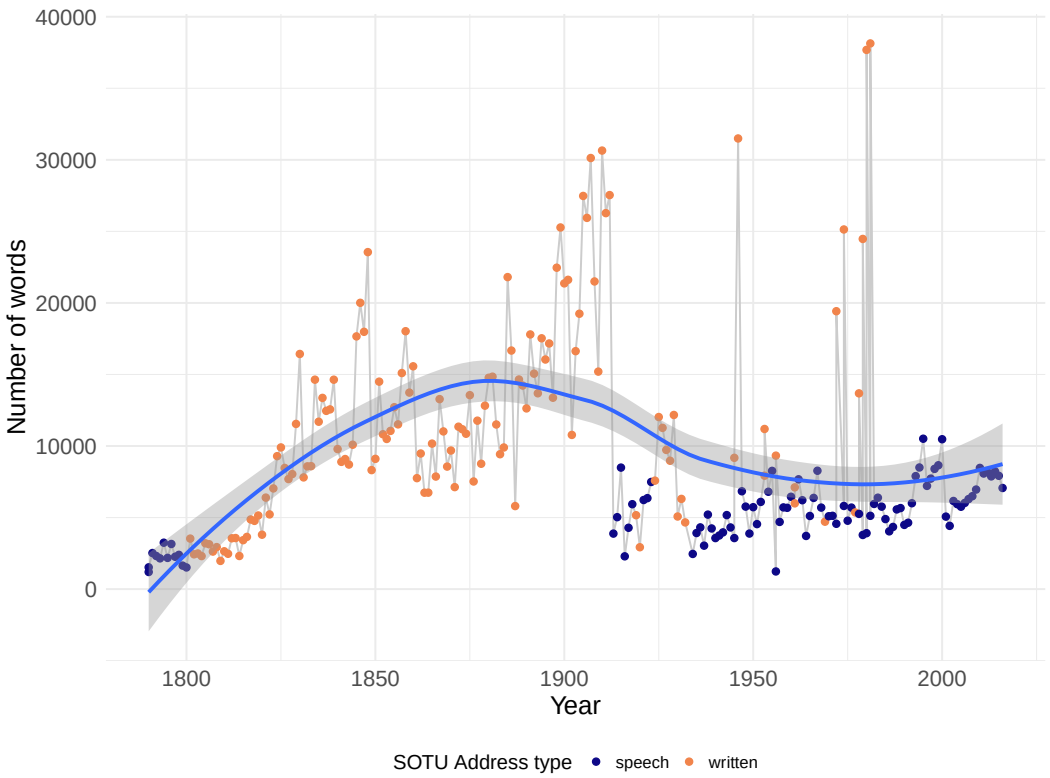


Figure 1: Length of each State of the Union address, in total number of tokens. Color shows whether the address was given as a speech or delivered as a written document.

```
 [1] "year"          "country"       "people"        "government"
 [5] "law"           "time"          "nation"        "who"
 [9] "power"         "interest"      "world"         "war"
[13] "citizen"       "service"       "duty"          "part"
[17] "system"        "peace"         "right"         "man"
[21] "program"       "policy"        "work"          "act"
[25] "state"         "condition"     "subject"       "legislation"
[29] "force"         "effort"        "treaty"        "purpose"
[33] "what"          "land"          "business"      "action"
[37] "measure"       "tax"           "way"           "question"
[41] "relation"      "consideration"
```

The result is generally as would be expected from a corpus of government speeches, with references to proper nouns representing various organizations within the government and non-proper nouns indicating general topics of interest such as “tax”, “law”, and “peace”.

The length in tokens of each address is calculated similarly by grouping and summarizing at the document id level. The results can be joined with the document table to get the year of the speech and then piped in a **ggplot2** command to illustrate how the length of the State of the Union has changed over time.

```
> cleanNLP::get_token(sotu) %>%
+   count(id) %>%
+   left_join(cleanNLP::get_document(sotu)) %>%
+   ggplot(aes(year, n)) +
+     geom_line(color = grey(0.8)) +
+     geom_point(aes(color = sotu_type)) +
+     geom_smooth()
```

Here, color is used to represent whether the address was given as an oral address or a written document. The output in Figure 1 shows that their are certainly time trends to the address length, with the form of the address (written versus spoken) also having a large effect on document length.

Finding the most used entities from the entity table over the time period of the corpus yields an alternative way to see the underlying topics. A slightly modified version of the code snippet used

to find the top nouns in the dataset can be used to find the top entities. The `get_token` function is replaced by `get_entity` and the table is filtered on `entity_type` rather than the universal part of speech code.

```
> cleanNLP::get_entity(sotu) %>%
+   filter(entity_type == "GPE") %>%
+   count(entity) %>%
+   top_n(n = 26, n) %>%
+   arrange(desc(n)) %>%
+   use_series(entity)
 [1] "the United States"        "America"
 [3] "States"                   "Mexico"
 [5] "Great Britain"            "Spain"
 [7] "Washington"               "China"
 [9] "Executive"                "France"
[11] "Cuba"                     "Japan"
[13] "Texas"                    "Russia"
[15] "The United States"        "Germany"
[17] "United States"            "California"
[19] "Nicaragua"                "the Soviet Union"
[21] "Mississippi"              "Iraq"
[23] "Alaska"                   "U.S."
[25] "Philippines"              "Panama"
[27] "the District of Columbia"
```

The ability to redo analyses from a slightly different perspective is a direct consequence of the tidy data model supplied by **cleanNLP**. The top locations include some obvious and some less obvious instances. Those sovereign nations included such as Great Britain, Mexico, Germany, and Japan seem as expected given either the United State's close ties or periods of war with them. The top states include the most populous regions (New York, California, and Texas) but also smaller states (Kansas, Oregon, Mississippi), the latter being more surprising.

One of the most straightforward way of extracting a high-level summary of the content of a speech is to extract all direct object object dependencies where the target noun is not a very common word. In order to do this for a particular speech, the dependency table is joined to the document table, a particular document is selected, and relationships of type "dobj" (direct object) are filtered out. The result is then joined to the data set `word_frequency`, which is included with **cleanNLP**, and pairs with a target occurring less than 0.5% of the time are selected to give the final result. Here is an example of this using the first address made by George W. Bush in 2001:

```
> cleanNLP::get_dependency(sotu, get_token = TRUE) %>%
+   left_join(get_document(sotu)) %>%
+   filter(year == 2001, relation == "dobj") %>%
+   select(id = id, start = word, word = lemma_target) %>%
+   left_join(word_frequency) %>%
+   filter(frequency < 0.001) %>%
+   select(id, start, word) %$%
+   sprintf("%s => %s", start, word)
Joining, by = "id"
Joining, by = "word"
 [1] "take => oath"                  "using => statistic"
 [3] "increasing => layoff"          "protects => trillion"
 [5] "makes => welcoming"            "accelerating => cleanup"
 [7] "fight => homelessness"         "helping => neighbor"
 [9] "allowing => taxpayer"          "provide => mentor"
[11] "fight => illiteracy"           "promotes => compassion"
[13] "asked => ashcroft"             "end => profiling"
[15] "pay => trillion"               "throw => dart"
[17] "restores => fairness"          "promoting => internationalism"
[19] "makes => downpayment"          "discard => relic"
[21] "confronting => shortage"       "directed => cheney"
[23] "sound => footing"              "divided => conscience"
[25] "done => servant"
```

Most of these phrases correspond with the "compassionate conservatism" that George W. Bush ran under in the preceding 2000 election. Applying the same analysis to the 2002 State of the Union, which came under the shadow of the September 11th terrorist attacks, shows a drastic shift in focus.

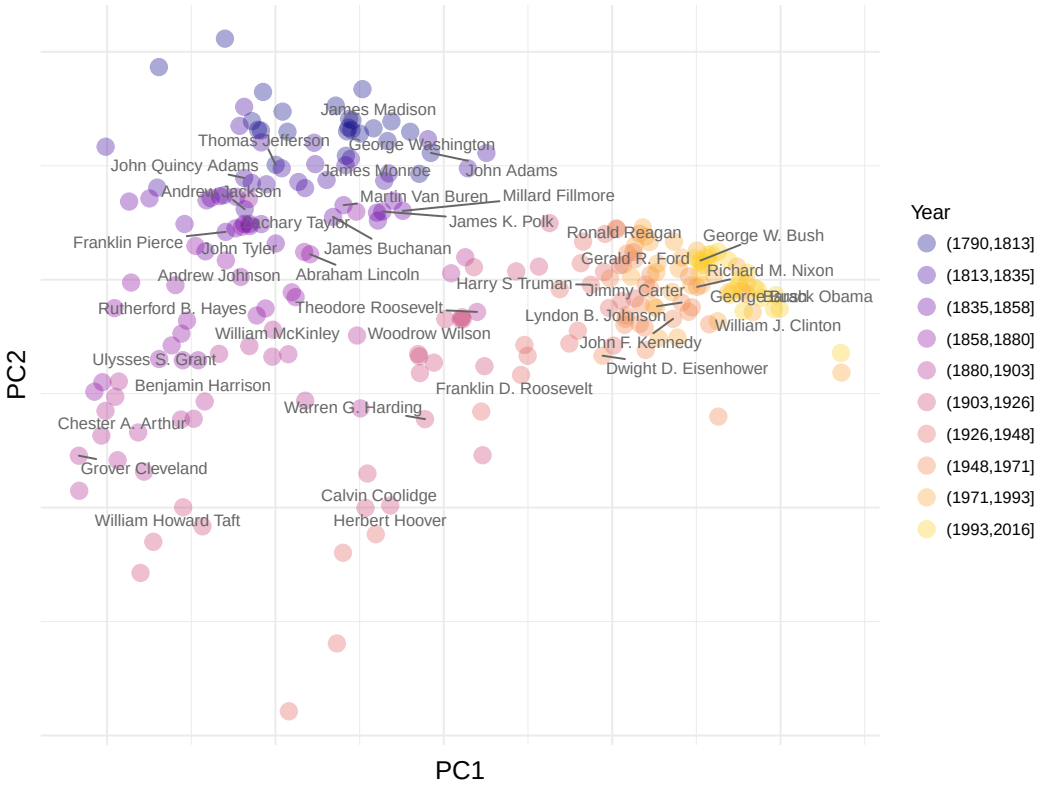


Figure 2: State of the Union Speeches, highlighting each President's first address, plotted using the first two principal components of the term frequency matrix of non-proper nouns.

```
> cleanNLP::get_dependency(sotu, get_token = TRUE) %>%
+   left_join(get_document(sotu)) %>%
+   filter(year == 2002, relation == "dobj") %>%
+   select(id = id, start = word, word = lemma_target) %>%
+   left_join(word_frequency) %>%
+   filter(frequency < 0.0005) %>%
+   select(id, start, word) %$%
+   sprintf("%s => %s", start, word)
Joining, by = "id"
Joining, by = "word"
 [1] "urged => follower"        "called => troop"
 [3] "brought => sorrow"        "owe => micheal"
 [5] "ticking => timebomb"      "have => troop"
 [7] "hold => hostage"          "eliminate => parasite"
 [9] "flaunt => hostility"      "develop => anthrax"
[11] "put => troop"             "increased => vigilance"
[13] "fight => anthrax"         "thank => attendant"
[15] "defeat => recession"      "want => paycheck"
[17] "set => posturing"         "enact => safeguard"
[19] "embracing => ethic"       "owns => aspiration"
[21] "containing => resentment" "erasing => rivalry"
[23] "embrace => tyranny"
```

Here the topics have almost entirely shifted to counter-terrorism and national security efforts.

Models

The `get_tfidf` function provided by **cleanNLP** converts a token table into a sparse matrix representing the term-frequency inverse document frequency matrix (or any intermediate part of that calculation). This is particularly useful when building models from a textual corpus. The `tidy_pca`, also included with the package, takes a matrix and returns a data frame containing the desired number of principal components. Dimension reduction involves piping the token table for a corpus into the `get_tfidif`

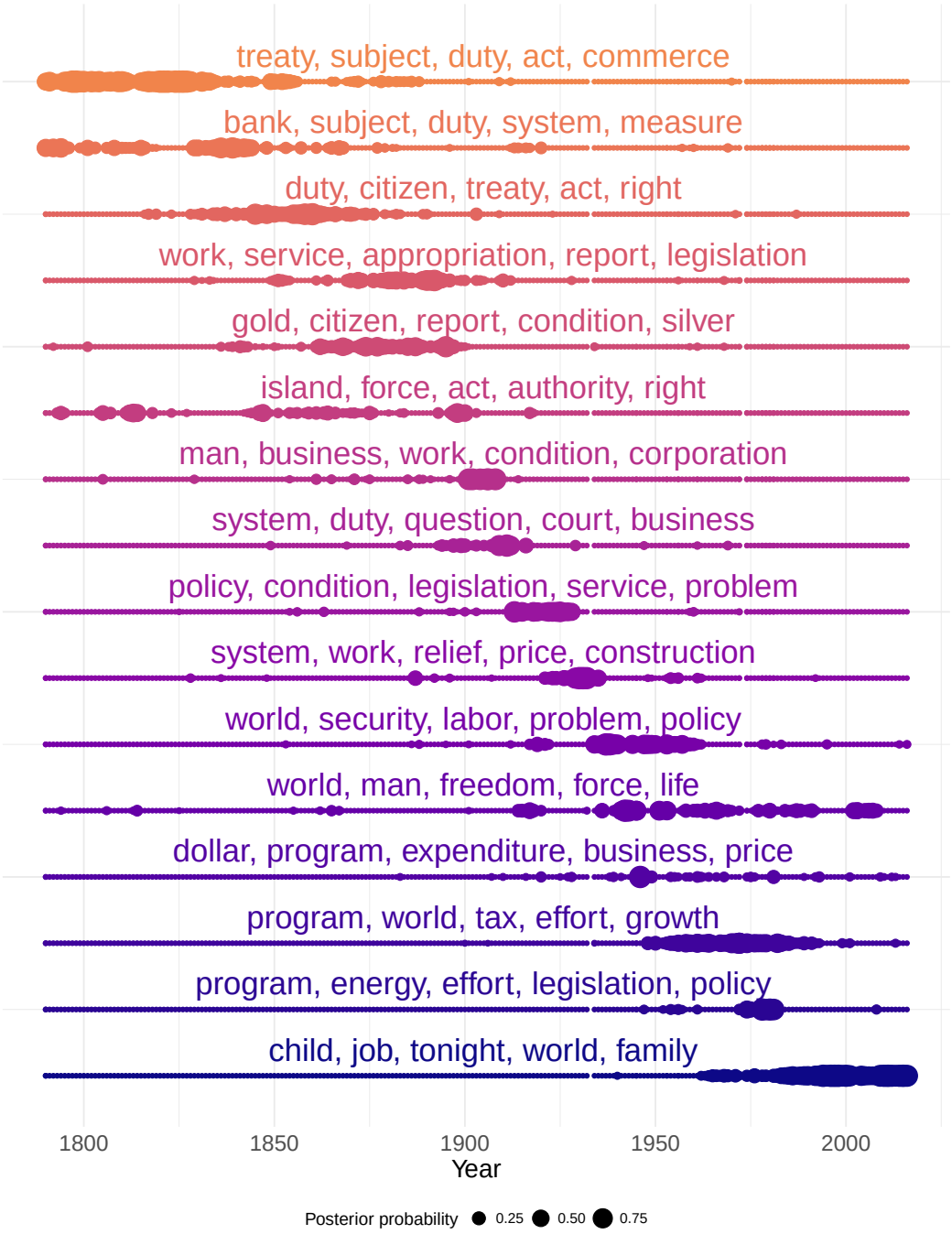


Figure 3: Distribution of topic model posterior probabilities over time on the State of the Union corpus. The top five words associated with each topic are displayed, with topics sorted by the median year of all documents placed into the respective topic using the maximum posterior probabilities.

function and passing the results to `tidy_pca`.

```
> pca <- cleanNLP::get_token(sotu) %>%
+   filter(pos %in% c("NN", "NNS")) %>%
+   cleanNLP::get_tfidf(min_df = 0.05, max_df = 0.95,
+                       type = "tfidf", tf_weight = "dnorm") %$%
+   cleanNLP::tidy_pca(tfidf, get_document(sotu))
```

In this example only non-proper nouns have been included in order to minimize the stylistic attributes of the speeches in order to focus more on their content. A scatter plot of the speeches using these components is shown in Figure 2. There is a definitive temporal pattern to the documents, with the 20th century addresses forming a distinct cluster on the right side of the plot.

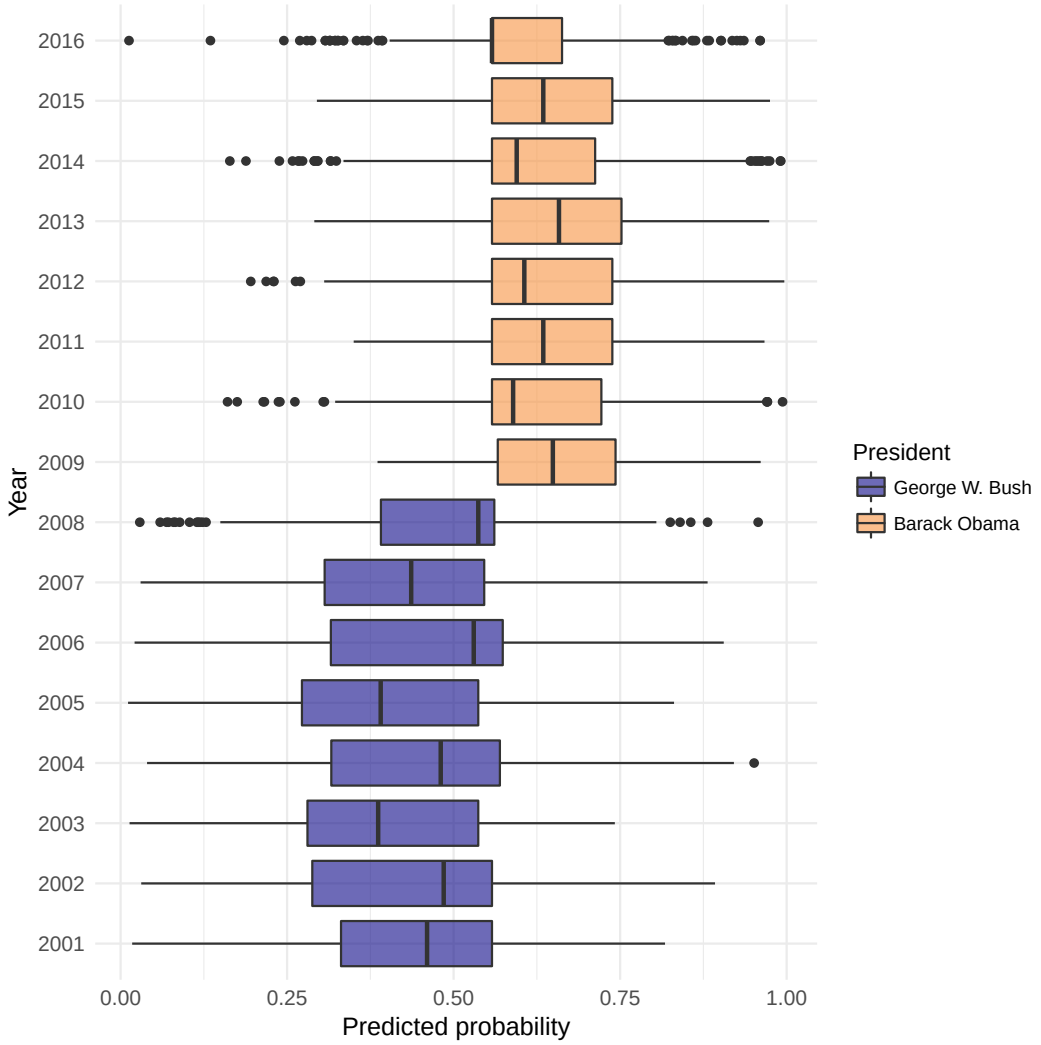


Figure 4: Boxplot of predicted probabilities, at the sentence level, for all 16 State of the Union addresses by Presidents George W. Bush and Barack Obama. The probability represents the extent to which the model believe the sentence was spoken by President Obama. Odd years were used for training and even years for testing. Cross-validation on the training set was used, with the one standard error rule, to set the lambda tuning parameter.

Topic models are a collection of statistical models for describing abstract themes within a textual corpus. Each theme is characterized by a collection of words that commonly co-occur; for example, the words 'crop', 'dairy', 'tractor', and 'hectare', might define a *farming* theme. One of the most popular topic models is latent Dirichlet allocation (LDA), a Bayesian model where each topic is described by a probability distribution over a vocabulary of words. Each document is then characterized by a probability distribution over the available topics. For a formal description, see Blei et al. (2003) and Pritchard et al. (2000), the original papers outlining LDA. To fit LDA on a corpus of text parsed by the **cleanNLP** package, the output of `get_tfidf` can be piped directly to the `LDA` function in the package **topicmodels**. The topic model function requires raw counts, so the type variable in `get_tfidf` is set to "tf".

```
> library(topicmodels)
> tm <- cleanNLP::get_token(sotu) %>%
+   filter(pos %in% c("NN", "NNS")) %>%
+   cleanNLP::get_tfidf(min_df = 0.05, max_df = 0.95,
+                       type = "tf", tf_weight = "raw") %
+   LDA(tf, k = 16, control = list(verbose = 1))
```

The topics, ordered by approximate time period, are visualized in Figure 3. We describe each topic by giving the five most important words Most topics exist for a few decades and then largely disappear, though some persist over non-contiguous periods of the presidency. The "program, energy, effort,

legislation, policy" topic, for example, appears during the 1950s and crops up again during the energy crisis of the 1970s. The "world, man, freedom, force, life" topic peaks during both World Wars, but is absent during the 1920s and early 1930s.

Finally, the **cleanNLP** data model is also convenient for building predictive models. The State of the Union corpus does not lend itself to an obviously applicable prediction problem. A classifier that distinguishes speeches made by George W. Bush and Barrack Obama will be constructed here for the purpose of illustration. As a first step, a term-frequency matrix is extracted using the same technique as was used with the topic modeling function. However, here the frequency is computed for each sentence in the corpus rather than the document as a whole. The ability to do this seamlessly with a single additional `mutate` function defining a new id illustrates the flexibility of the `get_tfidf` function.

```
> df <- get_token(sotu) %>%
+   left_join(get_document(sotu)) %>%
+   filter(year > 2000) %>%
+   mutate(new_id = paste(id, sid, sep = "-")) %>%
+   filter(pos %in% c("NN", "NNS"))
Joining, by = "id"
> mat <- get_tfidf(df, min_df = 0, max_df = 1, type = "tf",
+                  tf_weight = "raw", doc_var = "new_id")
```

It will be nessisary to define a response variable y indicating whether this is a speech made by President Obama as well as a training flag indicating which speeches were made in odd numbered years. This is done via a separate table join and a pair of mutations.

```
> meta <- data_frame(new_id = mat$id) %>%
+   left_join(df[!duplicated(df$new_id),]) %>%
+   mutate(y = as.numeric(president == "Barack Obama")) %>%
+   mutate(train = year %in% seq(2001,2016, by = 2))
Joining, by = "new_id"
```

The output may now be used as input to the elastic net function provided by the **glmnet** package. The response is set to the binomial family given the binary nature of the response and training is done on only those speeches occurring in odd-numbered years. Cross-validation is used in order to select the best value of the model's tuning parameter.

```
> library(glmnet)
> model <- cv.glmnet(mat$tf[meta$train,], meta$y[meta$train], family = "binomial")
```

A boxplot of the predicted classes for each address is given in Figure 4. The algorithm does a very good job of separating the speeches. Looking at the odd years versus even years (the training and testing sets, respectively) indicates that the model has not been over-fit.

One benefit of the penalized linear regression model is that it is possible to interpret the coefficients in a meaningful way. Here are the non-zero elements of the regression vector, coded as whether the have a positive (more Obama) or negative (more Bush) sign:

```
> beta <- coef(model, s = model[["lambda"]][11])[-1]
> sprintf("%s (%d)", mat$vocab, sign(beta))[beta != 0]
 [1] "job (1)"          "business (1)"     "citizen (-1)"
 [4] "terrorist (-1)"   "government (-1)"  "freedom (-1)"
 [7] "home (1)"         "college (1)"      "weapon (-1)"
[10] "deficit (1)"      "company (1)"      "peace (-1)"
[13] "enemy (-1)"       "terror (-1)"      "income (-1)"
[16] "drug (-1)"        "kid (1)"          "regime (-1)"
[19] "class (1)"        "crisis (1)"       "industry (1)"
[22] "need (-1)"        "fact (1)"         "relief (-1)"
[25] "bank (1)"         "liberty (-1)"     "society (-1)"
[28] "duty (-1)"        "folk (1)"         "account (-1)"
[31] "compassion (-1)"  "environment (-1)" "inspector (-1)"
```

These generally seem as expected given the main policy topics of focus under each administration. During most of the Bush presidency, as mentioned before, the focus was on national security and foreign policy. Obama, on the other hand, inherited the recession of 2008 and was more focused on the overall economic policy.

Conclusions

In this paper a normalized data model for representing text annotations has been presented and rationalized. We have also demonstrated how the R package **cleanNLP** implements this data model using various, configurable back ends. Our focus has been to illustrate why this general approach and specific implementation is both powerful and easy to integrate into existing data pipelines. It is expected that some users will utilize the entirety of the underlying annotation pipelines, internal R structures, and helper functions. Others may use the package as a convenient wrapper around either the CoreNLP or spaCy libraries. In either extreme, or anywhere in between, our approach provides powerful tools for applying exploratory, graphical, and model-based techniques to textual data sources.

The **cleanNLP** package continues to be actively developed. In particular, we hope to include new sentence-level annotations as they are integrated into the spaCy and CoreNLP libraries. While major releases are available on CRAN, new features are added periodically on the development branch located at: https://github.com/statsmaths/cleanNLP. Bug reports, feature and collaboration requests can all be made using the GitHub issues page.

Bibliography

J. Allaire, K. Ushey, Y. Tang, and D. Eddelbuettel. *reticulate: R Interface to Python*, 2017. URL https://github.com/rstudio/reticulate. [p249]

T. Arnold and L. Tilton. *coreNLP: Wrappers Around Stanford CoreNLP Tools*, 2016. R package version 0.4-2. [p249]

T. B. Arnold. *sotu: United States Presidential State of the Union Addresses*, 2017. URL https://CRAN.R-project.org/package=sotu. R package version 1.0.2. [p258]

S. M. Bache and H. Wickham. *magrittr: A Forward-Pipe Operator for R*, 2014. URL https://CRAN.R-project.org/package=magrittr. R package version 1.5. [p248]

J. Baldridge. The openNLP project. *URL: http://opennlp. apache. org/index. html,(accessed 2 February 2012)*, 2005. [p248]

K. Benoit and A. Matsuo. *spacyr: R Wrapper to the spaCy NLP Library*, 2017. URL http://github.com/kbenoit/spacyr. R package version 0.9.0. [p249]

D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3 (Jan):993–1022, 2003. [p263]

P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*, 2016. [p257]

S. Buchholz and E. Marsi. Conll-x shared task on multilingual dependency parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*, pages 149–164. Association for Computational Linguistics, 2006. [p252]

J. Chang. *lda: Collapsed Gibbs Sampling Methods for Topic Models*, 2015. URL https://CRAN.R-project.org/package=lda. R package version 1.4.2. [p249]

E. F. Codd. *The relational model for database management: version 2*. Addison-Wesley Longman Publishing Co., Inc., 1990. [p251]

M.-C. De Marneffe, T. Dozat, N. Silveira, K. Haverinen, F. Ginter, J. Nivre, and C. D. Manning. Universal Stanford dependencies: A cross-linguistic typology. In *LREC*, volume 14, pages 4585–92, 2014. [p255]

I. Feinerer, K. Hornik, and D. Meyer. Text mining infrastructure in R. *Journal of statistical software*, 25(5): 1–54, 2008. URL https://doi.org/10.18637/jss.v025.i05. [p249]

J. R. Finkel, T. Grenager, and C. Manning. Incorporating non-local information into information extraction systems by Gibbs sampling. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 363–370. Association for Computational Linguistics, 2005. [p255]

S. Firke. *janitor: Simple Tools for Examining and Cleaning Dirty Data*, 2016. URL https://CRAN.R-project.org/package=janitor. R package version 0.2.1. [p248]

W. Freitas. *sqliter: Connection wrapper to SQLite databases*, 2014. URL https://CRAN.R-project.org/package=sqliter. R package version 0.1.0. [p249]

G. Gaikwad and D. J. Joshi. Multiclass mood classification on twitter using lexicon dictionary and machine learning algorithms. In *Inventive Computation Technologies (ICICT), International Conference on*, volume 1, pages 1–6. IEEE, 2016. [p257]

S. Green, M.-C. De Marneffe, J. Bauer, and C. D. Manning. Multiword expression identification with tree substitution grammars: A parsing tour de force with french. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 725–735. Association for Computational Linguistics, 2011. [p253]

B. Grün and K. Hornik. topicmodels: An R package for fitting topic models. *Journal of Statistical Software*, 40(13):1–30, 2011. URL https://doi.org/10.18637/jss.v040.i13. [p249]

S. Hellmann, J. Lehmann, and S. Auer. Nif: An ontology-based and linked-data-aware NLP interchange format. *Working Draft*, 2012. [p252]

M. Honnibal and M. Johnson. An improved non-monotonic transition system for dependency parsing. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1373–1378, Lisbon, Portugal, September 2015. Association for Computational Linguistics. URL https://aclweb.org/anthology/D/D15/D15-1162. [p248]

K. Hornik. *NLP: Natural Language Processing Infrastructure*, 2016a. URL https://CRAN.R-project.org/package=NLP. R package version 0.1-9. [p249]

K. Hornik. *openNLP: Apache OpenNLP Tools Interface*, 2016b. URL https://CRAN.R-project.org/package=openNLP. R package version 0.2-6. [p249]

K. Hornik. *StanfordCoreNLP*, 2016c. URL http://datacube.wu.ac.at/src/contrib/. R package version 0.1-1. [p249]

N. Ide and L. Romary. A common framework for syntactic annotation. In *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*, pages 306–313. Association for Computational Linguistics, 2001. [p252]

J. Kabbara and J. C. K. Cheung. Stylistic transfer in natural language generation systems using recurrent neural networks. *EMNLP 2016*, page 43, 2016. [p257]

H. Lee, Y. Peirsman, A. Chang, N. Chambers, M. Surdeanu, and D. Jurafsky. Stanford's multi-pass sieve coreference resolution system at the CoNLL-2011 shared task. In *Proceedings of the Fifteenth Conference on Computational Natural Language Learning: Shared Task*, pages 28–34. Association for Computational Linguistics, 2011. [p255]

H. Lee, A. Chang, Y. Peirsman, N. Chambers, M. Surdeanu, and D. Jurafsky. Deterministic coreference resolution based on entity-centric, precision-ranked rules. *Computational Linguistics*, 39(4):885–916, 2013. URL https://doi.org/10.1162/coli_a_00152. [p255]

C. D. Manning, M. Surdeanu, J. Bauer, J. R. Finkel, S. Bethard, and D. McClosky. The Stanford CoreNLP natural language processing toolkit. In *ACL (System Demonstrations)*, pages 55–60, 2014. [p248]

W. McKinney et al. Data structures for statistical computing in Python. In *Proceedings of the 9th Python in Science Conference*, volume 445, pages 51–56. van der Voort S, Millman J, 2010. [p248]

T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013. [p257]

L. Mullen. *tokenizers: A Consistent Interface to Tokenize Natural Language Text*, 2016. URL https://CRAN.R-project.org/package=tokenizers. R package version 0.1.4. [p249]

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(Oct):2825–2830, 2011. [p248]

J. Pennington, R. Socher, and C. D. Manning. GloVe: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–1543, 2014. URL https://doi.org/10.3115/v1/D14-1162. [p257]

G. Peters. *State of the Union Addresses and Messages*, 2016. URL http://www.presidency.ucsb.edu/sou.php. [p258]

S. Petrov. Announcing SyntaxNet: The world's most accurate parser goes open source. *Google Research Blog, May*, 12:2016, 2016. [p248]

J. K. Pritchard, M. Stephens, and P. Donnelly. Inference of population structure using multilocus genotype data. *Genetics*, 155(2):945–959, 2000. [p263]

A. N. Rafferty and C. D. Manning. Parsing three German treebanks: Lexicalized and unlexicalized baselines. In *Proceedings of the Workshop on Parsing German*, pages 40–46. Association for Computational Linguistics, 2008. [p253]

K. Raghunathan, H. Lee, S. Rangarajan, N. Chambers, M. Surdeanu, D. Jurafsky, and C. Manning. A multi-pass sieve for coreference resolution. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 492–501. Association for Computational Linguistics, 2010. [p255]

M. Recasens, M.-C. de Marneffe, and C. Potts. The life and death of discourse entities: Identifying singleton mentions. In *HLT-NAACL*, pages 627–633, 2013. [p255]

D. Robinson. *broom: Convert Statistical Analysis Objects into Tidy Data Frames*, 2017. URL https://CRAN.R-project.org/package=broom. R package version 0.4.2. [p248]

R. Schifanella, P. de Juan, J. Tetreault, and L. Cao. Detecting sarcasm in multimodal social platforms. In *Proceedings of the 2016 ACM on Multimedia Conference*, pages 1136–1145. ACM, 2016. [p257]

J. Silge and D. Robinson. tidytext: Text mining and analysis using tidy data principles in R. *The Journal of Open Source Software*, 1(3), 2016. URL https://doi.org/10.21105/joss.00037. [p249]

R. Socher, A. Perelygin, J. Y. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the conference on empirical methods in natural language processing (EMNLP)*, volume 1631, page 1642. Citeseer, 2013. [p257]

K. Toutanova and C. D. Manning. Enriching the knowledge sources used in a maximum entropy part-of-speech tagger. In *Proceedings of the 2000 Joint SIGDAT conference on Empirical methods in natural language processing and very large corpora: held in conjunction with the 38th Annual Meeting of the Association for Computational Linguistics-Volume 13*, pages 63–70. Association for Computational Linguistics, 2000. [p253]

S. Urbanek. *rJava: Low-Level R to Java Interface*, 2016. URL https://CRAN.R-project.org/package=rJava. R package version 0.9-8. [p248]

H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2009. ISBN 978-0-387-98140-6. URL https://doi.org/10.1007/978-0-387-98141-3. [p248]

H. Wickham. Tidy data. *Journal of Statistical Software*, 59(i10), 2014. URL https://doi.org/10.18637/jss.v059.i10. [p248, 251]

H. Wickham. *tidyr: Easily Tidy Data with 'spread()' and 'gather()' Functions*, 2017. URL https://CRAN.R-project.org/package=tidyr. R package version 0.6.1. [p248]

H. Wickham and R. Francois. *dplyr: A Grammar of Data Manipulation*, 2016. URL https://CRAN.R-project.org/package=dplyr. R package version 0.5.0. [p248]

F. Wild. *lsa: Latent Semantic Analysis*, 2015. URL https://CRAN.R-project.org/package=lsa. R package version 0.73.1. [p249]

Taylor Arnold
Department of Mathematics and Computer Science
University of Richmond
Richmond, VA 23173 USA
tarnold2@richmond.edu

mle.tools: An R Package for Maximum Likelihood Bias Correction

by Josmar Mazucheli, André Felipe B. Menezes and Saralees Nadarajah

Abstract Recently, Mazucheli (2017) uploaded the package **mle.tools** to CRAN. It can be used for bias corrections of maximum likelihood estimates through the methodology proposed by Cox and Snell (1968). The main function of the package, `coxsnell.bc()`, computes the bias corrected maximum likelihood estimates. Although in general, the bias corrected estimators may be expected to have better sampling properties than the uncorrected estimators, analytical expressions from the formula proposed by Cox and Snell (1968) are either tedious or impossible to obtain. The purpose of this paper is twofolded: to introduce the **mle.tools** package, especially the `coxsnell.bc()` function; secondly, to compare, for thirty one continuous distributions, the bias estimates from the `coxsnell.bc()` function and the bias estimates from analytical expressions available in the literature. We also compare, for five distributions, the observed and expected Fisher information. Our numerical experiments show that the functions are efficient to estimate the biases by the Cox-Snell formula and for calculating the observed and expected Fisher information.

Introduction

Since it was proposed by Fisher in a series of papers from 1912 to 1934, the maximum likelihood method for parameter estimation has been employed to several issues in statistical inference, because of its many appealing properties. For instance, the maximum likelihood estimators, hereafter referred to as MLEs, are asymptotically unbiased, efficient, consistent, invariant under parameter transformation and asymptotically normally distributed (Edwards, 1992; Lehmann, 1999). Most properties that make the MLEs attractive depend on the sample size, hence such properties as unbiasedness, may not be valid for small samples or even moderate samples (Kay, 1995). Indeed, the maximum likelihood method produces biased estimators, i.e., expected values of MLEs differ from the real true parameter values providing systematic errors. In particular, these estimators typically have biases of order $\mathcal{O}\left(n^{-1}\right)$, thus these errors reduce as sample size increases (Cordeiro and Cribari-Neto, 2014).

Applying the corrective Cox-Snell methodology, many researchers have developed nearly unbiased estimators for the parameters of several probability distributions. Interested readers can refer to Cordeiro et al. (1997), Cribari-Neto and Vasconcellos (2002), Saha and Paul (2005), Lemonte et al. (2007), Giles and Feng (2009) Lagos-Álvarez et al. (2011), Lemonte (2011), Giles (2012b), Giles (2012a), Schwartz et al. (2013), Giles et al. (2013), Teimouri and Nadarajah (2013), Xiao and Giles (2014), Zhang and Liu (2015), Teimouri and Nadarajah (2016), Reath (2016), Giles et al. (2016), Schwartz and Giles (2016), Wang and Wang (2017), Mazucheli and Dey (2017) and references cited therein.

In general, the Cox-Snell methodology is efficient for bias corrections. However, obtaining analytical expressions for some probability distributions, mainly for those indexed by more than two parameters, can be notoriously cumbersome or impossible. Stočsić and Cordeiro (2009) presented Maple and Mathematica scripts that may be used to calculate closed form analytic expressions for bias corrections using the Cox-Snell formula. They tested the scripts for 20 two-parameter continuous probability distributions, and the results were compared with those published in earlier works. In the same direction, researchers from the University of Illinois, at Urbana-Champaign, have developed a Mathematica program, entitled "CSCK MLE Bias Calculation" (Johnson et al., 2012b) that enables the user to calculate the analytic Cox-Snell MLE bias vectors for various probability distributions with up to four unknown parameters. It is important to mention that both, Maple (Maple, 2017) and Mathematica (Wolfram Research, Inc., 2010), are commercial softwares.

In this paper, our objective is to introduce a new contributed R (R Core Team, 2016) package, namely **mle.tools** that computes the expected/observed Fisher information and the bias corrected estimates by the methodology proposed by Cox and Snell (1968). The theoretical background of the methodology is presented in Section Overview of the Cox-Snell methodology. Details about the **mle.tools** package are described in Section The **mle.tools** package details. Closed form solutions of bias corrections are collected from the literature for a large number of distributions and compared to the output from the `coxsnell.bc()` function, see Section Comparative study. In Section Additional Applications, we compare various estimates of Fisher's information, considering a real application from the literature. Finally, Section Concluding Remarks contains some concluding remarks and directions for future research.

Overview of the Cox-Snell methodology

Let $X_1, \ldots, X_n$ be n independent random variables with probability density function $f(x_i \mid \boldsymbol{\theta})$ depending on a p-dimensional parameter vector $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_p)$. Without loss of generality, let $l = l(\boldsymbol{\theta} \mid \boldsymbol{x})$ be the log-likelihood function for the unknown p-dimensional parameter vector $\boldsymbol{\theta}$ given a sample of n observations. We shall assume some regularity conditions on the behavior of $l(\boldsymbol{\theta} \mid \boldsymbol{x})$ (Cox and Hinkley, 1979).

The joint cumulants of the derivatives of l are given by:

$$\kappa_{ij} = \mathbb{E}\left[\frac{\partial^2 l}{\partial \boldsymbol{\theta}_i \, \partial \boldsymbol{\theta}_j}\right], \tag{1}$$

$$\kappa_{ijl} = \mathbb{E}\left[\frac{\partial^3 l}{\partial \boldsymbol{\theta}_i \, \partial \boldsymbol{\theta}_j \, \partial \boldsymbol{\theta}_l}\right], \tag{2}$$

$$\kappa_{ij,l} = \mathbb{E}\left[\left(\frac{\partial^2 l}{\partial \boldsymbol{\theta}_i \, \partial \boldsymbol{\theta}_j}\right)\left(\frac{\partial l}{\partial \boldsymbol{\theta}_l}\right)\right], \tag{3}$$

$$\kappa_{ij}^{(l)} = \frac{\partial \kappa_{ij}}{\partial \boldsymbol{\theta}_l} \tag{4}$$

for $i, j, l = 1, \ldots, p$.

The bias expression of the sth element of $\widehat{\boldsymbol{\theta}}$, the MLEs of $\boldsymbol{\theta}$, when the sample data are independent, but not necessarily identically distributed, was proposed by Cox and Snell (1968):

$$\mathcal{B}\left(\widehat{\theta}_s\right) = \sum_{i=1}^{p}\sum_{j=1}^{p}\sum_{l=1}^{p} \kappa^{si}\,\kappa^{jl}\left[0.5\kappa_{ijl} + \kappa_{ij,l}\right] + \mathcal{O}\left(n^{-2}\right), \tag{5}$$

where $s = 1, \ldots, p$ and κ^{ij} is the (i, j)th element of the inverse of the negative of the expected Fisher information.

Thereafter, Cordeiro and Klein (1994) noticed that equation (5) holds even if the data are non-independent, and it can be re-expressed as:

$$\mathcal{B}\left(\widehat{\theta}_s\right) = \sum_{i=1}^{p} \kappa^{si} \sum_{j=1}^{p}\sum_{l=1}^{p}\left[\kappa_{ij}^{(l)} - 0.5\kappa_{ijl}\right]\kappa^{jl} + \mathcal{O}\left(n^{-2}\right). \tag{6}$$

Defining $a_{ij}^{(l)} = \kappa_{ij}^{(l)} - 0.5\kappa_{ijl}$, $A^{(l)} = \left\{a_{ij}^{(l)}\right\}$ and $K = \left[-\kappa_{ij}\right]$, the expected Fisher information matrix for $i, j, l = 1, \ldots, n$, the bias expression for $\widehat{\boldsymbol{\theta}}$ in matrix notation is:

$$\mathcal{B}\left(\widehat{\boldsymbol{\theta}}\right) = K^{-1} A \operatorname{vec}\left(K^{-1}\right) + \mathcal{O}\left(n^{-2}\right), \tag{7}$$

where $\operatorname{vec}\left(K^{-1}\right)$ is the vector obtained by stacking the columns of K^{-1} and $A = \left\{A^1 \mid \cdots \mid A^p\right\}$.

Finally, the bias corrected MLE for θ_s can be obtained as:

$$\widetilde{\theta}_s = \widehat{\theta}_s - \widehat{\mathcal{B}}\left(\widehat{\theta}_s\right). \tag{8}$$

Alternatively, using matrix notation the bias corrected MLEs can be expressed as Cordeiro and Klein (1994):

$$\widetilde{\boldsymbol{\theta}} = \widehat{\boldsymbol{\theta}} - \widehat{K}^{-1}\widehat{A}\operatorname{vec}\left(\widehat{K}^{-1}\right), \tag{9}$$

where $\widehat{K} = K\big|_{\boldsymbol{\theta}=\widehat{\boldsymbol{\theta}}}$ and $\widehat{A} = A\big|_{\boldsymbol{\theta}=\widehat{\boldsymbol{\theta}}}$.

The mle.tools package details

The current version of the **mle.tools** package, uploaded to CRAN in February, 2017, has implemented three functions — `observed.varcov()`, `expected.varcov()` and `coxsnell.bc()` — which are of great interest in data analysis based on MLEs. These functions calculate, respectively, the observed Fisher information, the expected Fisher information and the bias corrected MLEs using the bias formula in (5). The above mentioned functions can be applied to any probability density function whose terms

are available in the derivatives table of the `D()` function (see "deriv.c" source code for further details). Integrals, when required, are computed numerically via the `integrate()` function. Below are some mathematical details of how the returned values from the three functions are calculated.

Let $X_1, \ldots, X_n$ be independent and identical random variables with probability density function $f(x_i \mid \boldsymbol{\theta})$ depending on a p-dimensional parameter vector $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_p)$. The (j,k)th element of the observed, H_{jk}, and expected, I_{jk}, Fisher information are calculated, respectively, as

$$H_{jk} = -\sum_{i=1}^{n} \frac{\partial^2}{\partial\theta_j \partial\theta_k} \log f(x_i \mid \boldsymbol{\theta}) \Bigg|_{\boldsymbol{\theta}=\widehat{\boldsymbol{\theta}}}$$

and

$$I_{jk} = -n \times E\left(\frac{\partial^2}{\partial\theta_j \partial\theta_k} \log f(x \mid \boldsymbol{\theta})\right) = -n \times \int_{\mathcal{X}} \frac{\partial^2}{\partial\theta_j \partial\theta_k} \log f(x \mid \boldsymbol{\theta}) \times f(x \mid \boldsymbol{\theta})\, \mathrm{d}x \Bigg|_{\boldsymbol{\theta}=\widehat{\boldsymbol{\theta}}},$$

where $j,k = 1, \ldots, p$, $\widehat{\boldsymbol{\theta}}$ is the MLE of $\boldsymbol{\theta}$ and $\mathcal{X}$ denotes the support of the random variable X.

The `observed.varcov()` function is as follows:

```
function (logdensity, X, parms, mle)
```

where `logdensity` is an R expression of the log of the probability density function, `X` is a numeric vector containing the observations, `parms` is a character vector of the parameter name(s) specified in the logdensity expression and `mle` is a numeric vector of the parameter estimate(s). This function returns a list with two components (i) `mle`: the inputed MLEs and (ii) `varcov`: the observed variance-covariance evaluated at the inputed MLE argument. The elements of the Hessian matrix are calculated analytically.

The functions `expected.varcov()` and `coxsnell.bc()` have the same arguments and are as follows:

```
function (density, logdensity, n, parms, mle, lower = "-Inf", upper = "Inf", ...)
```

where `density` and `logdensity` are R expressions of the probability density function and its logarithm, respectively, `n` is a numeric scalar of the sample size, `parms` is a character vector of the parameter names(s) specified in the density and log-density expressions, `mle` is a numeric vector of the parameter estimates, `lower` is the lower integration limit (`-Inf` is the default), `upper` is the upper integration limit (`Inf` is the default) and `...` are additional arguments passed to the `integrate()` function. The `expected.varcov()` function returns a list with two components:

`$mle` the inputed MLEs and

`$varcov` the expected covariance evaluated at the inputed MLEs.

The `coxsnell.bc()` function returns a list with five components:

`$mle` the inputed MLEs,

`$varcov` the expected variance-covariance evaluated at the inputed MLEs,

`$mle.bc` the bias corrected MLEs,

`$varcov.bc` the expected variance-covariance evaluated at the bias corrected MLEs

`$bias` the bias estimate(s).

Furthermore, the bias corrected MLE of θ_s, $s = 1, \ldots, p$ denoted by $\widetilde{\theta}_s$ is calculated as $\widetilde{\theta}_s = \widehat{\theta}_s - \widehat{\mathcal{B}}\left(\widehat{\theta}_s\right)$, where $\widehat{\theta}_s$ is the MLE of θ_s and

$$\widehat{\mathcal{B}}\left(\widehat{\theta}_s\right) = \sum_{j=1}^{p}\sum_{k=1}^{p}\sum_{l=1}^{p} \kappa^{sj}\kappa^{kl}\left[0.5\kappa_{jkl} + \kappa_{jk,l}\right] \Bigg|_{\boldsymbol{\theta}=\widehat{\boldsymbol{\theta}}},$$

where κ^{jk} is the (j,k)th element of the inverse of the negative of the expected Fisher information,

$$\kappa_{jkl} = n \int_{\mathcal{X}} \frac{\partial^3}{\partial\theta_j \partial\theta_k \partial\theta_l} \log f(x \mid \boldsymbol{\theta})\, f(x \mid \boldsymbol{\theta})\, \mathrm{d}x \Bigg|_{\boldsymbol{\theta}=\widehat{\boldsymbol{\theta}}},$$

$$\kappa_{jk,l} = n \int_{\mathcal{X}} \frac{\partial^2}{\partial\theta_j \partial\theta_k} \log f(x \mid \boldsymbol{\theta})\, \frac{\partial}{\theta_l} \log f(x \mid \boldsymbol{\theta})\, f(x \mid \boldsymbol{\theta})\, \mathrm{d}x \Bigg|_{\boldsymbol{\theta}=\widehat{\boldsymbol{\theta}}}$$

and $\mathcal{X}$ denotes the support of the random variable X.

It is important to emphasize that first, second and third-order partial log-density derivatives are analytically calculated via the `D()` function, while integrals are computed numerically, using the `integrate()` function. Furthermore, if numerical integration fails and/or the expected/observed information is singular, an error message is returned.

Comparative study

In order to evaluate the robustness of the `coxsnell.bc()` function, we compare, through real applications, the estimated biases obtained from the package and from the analytical expressions for a total of thirty one continuous probability distributions. The analytical expressions for each distribution, named as `distname.bc()`, can be found in the supplementary file "analyticalBC.R". For example, the entry `lindley.bc(n,mle)` evaluates the bias estimates locally at `n` and `mle` values.

In the sequel, the probability density function, the analytical Cox-Snell expressions and the bias estimates are provided for: Lindley, inverse Lindley, inverse Exponential, Shanker, inverse Shanker, Topp-Leone, Lévy, Rayleigh, inverse Rayleigh, Half-Logistic, Half-Cauchy, Half-Normal, Normal, inverse Gaussian, Log-Normal, Log-Logistic, Gamma, inverse Gamma, Lomax, weighted Lindley, generalized Rayleigh, Weibull, inverse Weibull, generalized Half-Normal, inverse generalized Half-Normal, Marshall-Olkin extended Exponential, Beta, Kumaraswamy, inverse Beta, Birnbaum-Saunders and generalized Pareto distributions.

It is noteworthy that analytical bias corrected expressions are not reported in the literature for the Lindley, Shanker, inverse Shanker, Lévy, inverse Rayleigh, half-Cauchy, inverse Weibull, inverse generalized half-normal and Marshall-Olkin extended exponential distributions.

According to all the results presented below, we observe concordance between the bias estimates given by the `coxsnell.bc()` function and the analytical expression(s) for 28 out the 31 distributions. The distributions which did not agree with the `coxsnell.bc()` function were the beta, Kumaraswamy and inverse beta distributions. Perhaps there are typos either in our typing or in the analytical expressions reported by Cordeiro et al. (1997), Lemonte (2011) and Stočsić and Cordeiro (2009). Having this view, we recalculated the analytical expressions for the biases. For the beta and inverse beta distributions, our recalculated analytical expressions agree with the results returned by the `coxsnell.bc()` function, so there are actually typos in the expression of Cordeiro et al. (1997) and Stočsić and Cordeiro (2009). For the Kumaraswamy, we could not evaluate the analytical expression given by the author but we compare the results from `coxsnell.bc()` function with a numerical evaluation in Maple (Maple, 2017) and the results are exactly equals.

1. One-parameter Lindley distribution with scale parameter θ

$$f(x \mid \theta) = \frac{\theta^2}{1+\theta}(1+x)\exp(-\theta x), \quad x > 0.$$

- Bias expression (not previously reported in the literature):

$$\mathcal{B}\left(\widehat{\theta}\right) = \frac{\left(\theta^3 + 6\,\theta^2 + 6\,\theta + 2\right)(\theta+1)\,\theta}{n\left(\theta^2 + 4\,\theta + 2\right)^2}. \tag{10}$$

Using the data set from Ghitany et al. (2008) we have $n = 100$, $\widehat{\theta} = 0.1866$ and $\widehat{se}\left(\widehat{\theta}\right) = 0.0133$. Evaluating the analytical expression (10) and the `coxsnell.bc()` function, we have, respectively,

```
lindley.bc(n = 100, mle = 0.1866)
##     theta
## 0.0009546
pdf <- quote(theta^2 / (theta + 1) * (1 + x) * exp(-theta * x))
lpdf <- quote(2 * log(theta) - log(1 + theta) - theta * x)
coxsnell.bc(density = pdf, logdensity = lpdf, n = 100,
            parms = c("theta"), mle = 0.1866, lower = 0)$bias
##     theta
## 0.0009546
```

2. Inverse Lindley distribution with scale parameter θ

$$f(x \mid \theta) = \frac{\theta^2}{1+\theta}\left(\frac{1+x}{x^3}\right)\exp\left(-\frac{\theta}{x}\right), \quad x > 0.$$

• Bias expression (Wang, 2015):

$$\mathcal{B}\left(\widehat{\theta}\right) = \frac{(\theta+1)\,\theta\,\left(\theta^3+6\,\theta^2+6\,\theta+2\right)}{n\,\left(\theta^2+4\,\theta+2\right)^2}. \tag{11}$$

Using the data set from Sharma et al. (2015) we have $n = 58$, $\widehat{\theta} = 60.0016$ and $\widehat{se}\left(\widehat{\theta}\right) = 7.7535$. Evaluating the analytical expression (11) and the `coxsnell.bc()` function, we have, respectively,

```
invlindley.bc(n = 58, mle =  60.0016)
## theta
## 1.017
pdf <- quote(theta^2 / (theta + 1) * ((1 + x) / x^3) *
             exp(-theta / x))
lpdf <- quote(2 * log(theta) - log(1 + theta) - theta / x)
coxsnell.bc(density = pdf, logdensity = lpdf, n = 58,
            parms = c("theta"), mle =  60.0016, lower = 0)$bias
## theta
## 1.017
```

3. Inverse exponential distribution with rate parameter θ

$$f(x \mid \theta) = \frac{\theta}{x^2}\,\exp\left(-\frac{\theta}{x}\right), \quad x > 0.$$

• Bias expression (Johnson et al., 2012b):

$$\mathcal{B}\left(\widehat{\theta}\right) = \frac{\theta}{n}. \tag{12}$$

Using the data set from Lawless (2011), we have $n = 30$, $\widehat{\theta} = 11.1786$ and $\widehat{se}\left(\widehat{\theta}\right) = 2.0409$. Evaluating the analytical expression (12) and the `coxsnell.bc()` function, we have, respectively,

```
invexp.bc(n = 30, mle = 11.1786)
##  theta
## 0.3726
pdf <- quote(theta / x^2 * exp(- theta / x))
lpdf <- quote(log(theta) - theta / x)
coxsnell.bc(density = pdf, logdensity = lpdf, n = 30,
            parms = c("theta"), mle = 11.1786, lower = 0)$bias
##  theta
## 0.3726
```

4. Shanker distribution with scale parameter θ

$$f(x \mid \theta) = \frac{\theta^2}{\theta^2+1}\,(\theta+x)\,\exp(-\theta\,x), \quad x > 0.$$

• For bias expression (not previously reported in the literature, see the "analyticalBC.R" file. Using the data set from Shanker (2015), we have $n = 31$, $\widehat{\theta} = 0.0647$ and $\widehat{se}\left(\widehat{\theta}\right) = 0.0082$. Evaluating the analytical expression and the `coxsnell.bc()` function, we have, respectively,

```
shanker.bc(n = 31, mle = 0.0647)
##    theta
## 0.001035
pdf <- quote(theta^2 / (theta^2 + 1) *  (theta + x) *
             exp(-theta * x))
lpdf <- quote(2*log(theta) - log(theta^2 + 1) + log(theta + x) -
              theta * x)
coxsnell.bc(density = pdf, logdensity = lpdf, n = 31,
            parms = c("theta"), mle = 0.0647, lower = 0)$bias
##    theta
## 0.001035
```

5. Inverse Shanker distribution with scale parameter θ

$$f(x \mid \theta) = \frac{\theta^2}{1+\theta^2}\left(\frac{1+\theta\,x}{x^3}\right)\exp\left(-\frac{\theta}{x}\right), \quad x > 0.$$

• Bias expression (not previously reported in the literature):

$$\mathcal{B}\left(\widehat{\theta}\right) = \frac{\theta^3 + 2\,\theta}{n\,(\theta^2 + 1)}. \tag{13}$$

Using the data set from Sharma et al. (2015), we have $n = 58$, $\widehat{\theta} = 59.1412$ and $\widehat{se}\left(\widehat{\theta}\right) = 7.7612$. Evaluating the analytical expression (13) and the `coxsnell.bc()` function, we have, respectively,

```
invshanker.bc(n = 58, mle = 59.1412)
## theta
##  1.02
pdf <- quote(theta^2 / (theta^2 + 1) * (theta * x + 1) /
             x^3 * exp(-theta / x))
lpdf <- quote(log(theta) - 2 * log(x) - theta / x)
coxsnell.bc(density = pdf, logdensity = lpdf, n = 58,
            parms = c("theta"), mle = 59.1412, lower = 0)$bias
## theta
##  1.02
```

6. Topp-Leone distribution with shape parameter ν

$$f(x \mid \nu) = 2\,\nu\,(1-x)\,x^{\nu-1}\,(2-x)^{\nu-1}, \quad 0 < x < 1.$$

• Bias expression (Giles, 2012a):

$$\mathcal{B}\left(\widehat{\nu}\right) = \frac{\nu}{n}. \tag{14}$$

Using the data set from Cordeiro and dos Santos Brito (2012), we have $n = 107$, $\widehat{\nu} = 2.0802$ and $\widehat{se}\left(\widehat{\nu}\right) = 0.2011$. Evaluating the analytical expression (14) and the `coxsnell.bc()` function, we have, respectively,

```
toppleone.bc(n = 107, mle = 2.0802)
##      nu
## 0.01944
pdf <- quote(2 * nu * x^(nu - 1) * (1 - x) * (2 - x)^(nu - 1))
lpdf <- quote(log(nu) + nu * log(x) + log(1 - x) + (nu - 1) *
              log(2 - x))
coxsnell.bc(density = pdf, logdensity = lpdf, n = 107,
            parms = c("nu"), mle = 2.0802, lower = 0, upper = 1)$bias
##      nu
## 0.01944
```

7. One-parameter Lévy distribution with scale parameter σ

$$f(x \mid \sigma) = \sqrt{\frac{\sigma}{2\,\pi}}\,x^{-\frac{3}{2}}\,\exp\left(-\frac{\sigma}{2\,x}\right), \quad x > 0.$$

• Bias expression (not previously reported in the literature):

$$\mathcal{B}\left(\widehat{\sigma}\right) = \frac{2\,\sigma}{n}. \tag{15}$$

Using the data set from Achcar et al. (2013), we have $n = 361$, $\widehat{\sigma} = 4.4461$ and $\widehat{se}\left(\widehat{\sigma}\right) = 0.3309$. Evaluating the analytical expression (15) and the `coxsnell.bc()` function, we have, respectively,

```
levy.bc(n = 361, mle = 4.4460)
##   sigma
## 0.02463
pdf <- quote(sqrt(sigma / (2 * pi)) * exp(-0.5 * sigma / x) /
             x^(3 / 2))
lpdf <- quote(0.5 * log(sigma) - 0.5 * sigma / x - (3 / 2) * log(x))
coxsnell.bc(density = pdf, logdensity = lpdf, n = 361,
            parms = c("sigma"), mle = 4.4460, lower = 0)$bias
##   sigma
## 0.02463
```

8. Rayleigh distribution with scale parameter σ

$$f(x \mid \sigma) = \frac{x}{\sigma^2} \exp\left(-\frac{x^2}{2\sigma^2}\right), \quad x > 0.$$

• Bias expression (Xiao and Giles, 2014):

$$\mathcal{B}(\widehat{\sigma}) = -\frac{\sigma}{8n}. \tag{16}$$

Using the data set from Bader and Priest (1982), we have $n = 69$, $\widehat{\sigma} = 1.2523$ and $\widehat{se}(\widehat{\sigma}) = 0.0754$. Evaluating the analytical expression (16) and the `coxsnell.bc()` function, we have, respectively,

```
rayleigh.bc(n = 69, mle = 1.2522)
##     sigma
## -0.002268
pdf <- quote(x / sigma^2 * exp(- 0.5 * (x / sigma)^2))
lpdf <- quote(- 2 * log(sigma) - 0.5 * x^2 / sigma^2)
coxsnell.bc(density = pdf, logdensity = lpdf, n = 69,
            parms = c("sigma"), mle = 1.2522, lower = 0)$bias
##     sigma
## -0.002268
```

9. Inverse Rayleigh distribution with scale parameter σ

$$f(x \mid \sigma) = \frac{2\sigma^2}{x^3} \exp\left(-\frac{\sigma}{x^2}\right), \quad x > 0.$$

• Bias expression (not previously reported in the literature):

$$\mathcal{B}(\widehat{\sigma}) = \frac{3\sigma}{8n}. \tag{17}$$

Using the data set from Bader and Priest (1982), we have $n = 63$, $\widehat{\sigma} = 2.8876$ and $\widehat{se}(\widehat{\sigma}) = 0.1819$. Evaluating the analytical expression (17) and the `coxsnell.bc()` function, we have, respectively,

```
invrayleigh.bc(n = 63, mle = 2.8876)
##   sigma
## 0.01719
pdf <- quote(2 * sigma^2 / x^3 * exp(-sigma^2 / x^2))
lpdf <- quote(2 * log(sigma) - sigma^2 / x^2)
coxsnell.bc(density = pdf, logdensity = lpdf, n = 63,
            parms = c("sigma"), mle = 2.8876, lower = 0)$bias
##   sigma
## 0.01719
```

10. Half-logistic distribution with scale parameter σ

$$f(x \mid \sigma) = \frac{2 \exp\left(-\frac{x}{\sigma}\right)}{\sigma\left[1 + \exp\left(-\frac{x}{\sigma}\right)\right]^2}, \quad x > 0.$$

• Bias expressions (Giles, 2012b):

$$\mathcal{B}(\widehat{\sigma}) = -\frac{0.05256766607\,\sigma}{n}. \tag{18}$$

Using the data set from Bhaumik et al. (2009), we have $n = 34$, $\widehat{\sigma} = 1.3926$ and $\widehat{se}(\widehat{\sigma}) = 0.2056$. Evaluating the analytical expression (17) and the `coxsnell.bc()` function, we have, respectively,

```
halflogistic.bc(n = 34, mle = 1.3925)
##     sigma
## -0.002153
pdf <- quote((2/sigma) * exp(-x / sigma) / (1 + exp(-x / sigma))^2)
lpdf <- quote(-log(sigma) - x / sigma - 2 * log(1 + exp(-x / sigma)))
coxsnell.bc(density = pdf, logdensity = lpdf, n = 34,
            parms = c("sigma"), mle = 1.3925, lower = 0)$bias
##     sigma
## -0.002153
```

11. Half-Cauchy distribution with scale parameter σ

$$f(x \mid \sigma) = \frac{2}{\pi} \frac{\sigma}{\sigma^2 + x^2}, \quad x > 0.$$

• Bias expression (not previously reported in the literature):

$$\mathcal{B}(\widehat{\sigma}) = -\frac{\sigma}{n}. \tag{19}$$

Using the data set from Alzaatreh et al. (2016), we have $n = 64$, $\widehat{\sigma} = 28.3345$ and $\widehat{se}(\widehat{\sigma}) = 4.4978$. Evaluating the analytical expression (19) and the `coxsnell.bc()` function, we have, respectively,

```
halfcauchy.bc(n = 64, mle = 28.3345)
##  sigma
## 0.4427
pdf <- quote( 2 / pi * sigma / (x^2 + sigma^2))
lpdf <- quote(log(sigma) - log(x^2 + sigma^2))
coxsnell.bc(density = pdf, logdensity = lpdf, n = 64,
            parms = c("sigma"), mle = 28.3345, lower = 0)$bias
##  sigma
## 0.4456
```

12. Half-normal distribution with scale parameter σ

$$f(x \mid \sigma) = \sqrt{\frac{2}{\pi}} \frac{1}{\sigma} \exp\left(-\frac{x^2}{2\sigma^2}\right), \quad x > 0.$$

• Bias expressions (Xiao and Giles, 2014):

$$\mathcal{B}(\widehat{\sigma}) = -\frac{\sigma}{4n}. \tag{20}$$

Using the data set from Raqab et al. (2008), we have $n = 69$, $\widehat{\sigma} = 1.5323$ and $\widehat{se}(\widehat{\sigma}) = 0.1304$. Evaluating the analytical expression (20) and the `coxsnell.bc()` function, we have, respectively,

```
halfnormal.bc(n = 69, mle = 1.5323)
##     sigma
## -0.005552
pdf <- quote(sqrt(2) / (sqrt(pi) * sigma) * exp(-x^2 / (2 * sigma^2)))
lpdf <- quote(-log(sigma) - x^2 / sigma^2 / 2 )
coxsnell.bc(density = pdf, logdensity = lpdf, n = 69,
            parms = c("sigma"), mle = 1.5323, lower = 0)$bias
##     sigma
## -0.005552
```

13. Normal distribution with mean μ and standard deviation σ

$$f(x \mid \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{(x-\mu)^2}{2\sigma^2}\right], \quad x \in (-\infty, \infty).$$

• Bias expressions (Stočsić and Cordeiro, 2009):

$$\mathcal{B}(\widehat{\mu}) = 0 \text{ and } \mathcal{B}(\widehat{\sigma}) = -\frac{3\sigma}{4n}. \tag{21}$$

Using the data set from Kundu (2005), we have $n = 23$, $\widehat{\mu} = 4.1506$, $\widehat{\sigma} = 0.5215$, $\widehat{se}(\widehat{\mu}) = 0.1087$ and $\widehat{se}(\widehat{\sigma}) = 0.0769$. Evaluating the analytical expressions (21) and the `coxsnell.bc()` function, we have, respectively,

```
normal.bc(n = 23, mle = c(4.1506, 0.5215))
##       mu    sigma
##  0.00000 -0.01701
pdf <- quote(1 / (sqrt(2 * pi) * sigma) *
             exp(-0.5 / sigma^2 * (x - mu)^2))
lpdf <- quote(-log(sigma) - 0.5 / sigma^2 * (x - mu)^2)
coxsnell.bc(density = pdf, logdensity = lpdf, n = 23,
            parms = c("mu", "sigma"), mle = c(4.1506, 0.5215))$bias
##         mu      sigma
## -4.071e-13 -1.701e-02
```

14. Inverse Gaussian distribution with mean μ and shape λ

$$f(x \mid \mu, \lambda) = \sqrt{\frac{\lambda}{2\,\pi\,x^3}}\,\exp\left[-\frac{\lambda\,(x-\mu)^2}{2\,x\,\mu^2}\right], \quad x > 0.$$

• Bias expressions (Stočsić and Cordeiro, 2009):

$$\mathcal{B}\left(\widehat{\mu}\right) = 0 \text{ and } \mathcal{B}\left(\widehat{\lambda}\right) = \frac{3\lambda}{n}. \tag{22}$$

Using the data set from Chhikara and Folks (1977), we have $n = 46$, $\widehat{\mu} = 3.6067$, $\widehat{\lambda} = 1.6584$, $\widehat{se}\left(\widehat{\mu}\right) = 0.7843$ and $\widehat{se}\left(\widehat{\lambda}\right) = 0.3458$. Evaluating the analytical expressions (22) and the `coxsnell.bc()` function, we have, respectively,

```
invgaussian.bc(n = 46, mle =  c(3.6065, 1.6589))
##     mu lambda
## 0.0000 0.1082
pdf <- quote(sqrt(lambda / (2 * pi * x^3)) *
             exp(-lambda * (x - mu)^2 / (2 * mu^2 * x)))
lpdf <- quote(0.5 * log(lambda) - lambda * (x - mu)^2 /
              (2 * mu^2 * x))
coxsnell.bc(density = pdf, logdensity = lpdf, n = 46,
            parms = c("mu", "lambda"), mle = c(3.6065, 1.6589),
            lower = 0)$bias
##        mu    lambda
## 3.483e-07 1.082e-01
```

15. Log-normal distribution with location μ and scale σ

$$f(x \mid \mu, \sigma) = \frac{1}{\sqrt{2\,\pi}\,x\,\sigma}\,\exp\left[-\frac{(\log x - \mu)^2}{\sigma^2}\right], \quad x > 0.$$

• Bias expressions (Stočsić and Cordeiro, 2009):

$$\mathcal{B}\left(\widehat{\mu}\right) = 0 \text{ and } \mathcal{B}\left(\widehat{\sigma}\right) = -\frac{3\,\sigma}{4\,n}. \tag{23}$$

Using the data set from Kumagai et al. (1989), we have $n = 30$, $\widehat{\mu} = 2.164$, $\widehat{\sigma} = 1.1765$, $\widehat{se}\left(\widehat{\mu}\right) = 0.2148$ and $\widehat{se}\left(\widehat{\sigma}\right) = 0.1519$. Evaluating the analytical expressions (23) and the `coxsnell.bc()` function, we have, respectively,

```
lognormal.bc(n = 30, mle = c(2.1643, 1.1765))
##       mu    sigma
##  0.00000 -0.02941
pdf <- quote(1 / (sqrt(2 * pi) * x * sigma) *
             exp(-0.5 * (log(x) - mu)^2 / sigma^2))
lpdf <- quote(-log(sigma) - 0.5 * (log(x) - mu)^2 / sigma^2)
coxsnell.bc(density = pdf, logdensity = lpdf, n = 30,
            parms = c("mu", "sigma"), mle = c(2.1643, 1.1765),
            lower = 0)$bias
##         mu      sigma
## -5.952e-09 -2.941e-02
```

16. Log-logistic distribution with shape β and scale α

$$f(x \mid \alpha, \beta) = \frac{(\beta/\alpha)\,(x/\alpha)^{\beta-1}}{\left[1 + (x/\alpha)^{\beta}\right]^2}, \quad x > 0.$$

• For bias expressions, see Reath (2016).

From Reath (2016) we have $n = 19$, $\widehat{\alpha} = 6.2542$, $\widehat{\beta} = 1.1732$, $\widehat{se}\left(\widehat{\alpha}\right) = 2.1352$, $\widehat{se}\left(\widehat{\beta}\right) = 0.2239$, $\widehat{\mathcal{B}}\left(\widehat{\alpha}\right) = 0.3585$ and $\widehat{\mathcal{B}}\left(\widehat{\beta}\right) = 0.0789$. Evaluating the `coxsnell.bc()` function, we have:

```
pdf <- quote((beta / alpha) * (x / alpha)^(beta - 1) /
             (1 + (x / alpha)^beta)^2)
lpdf <- quote(log(beta) - log(alpha) + (beta - 1) * log(x / alpha) -
```

```
              2 * log(1 + (x / alpha)^beta))
coxsnell.bc(density = pdf, logdensity = lpdf, n = 19,
            parms =  c("alpha", "beta"), mle = c(6.2537, 1.1734),
            lower = 0)$bias
##   alpha    beta
## 0.35854 0.07883
```

17. Gamma distribution with shape α and rate λ

$$f(x \mid \alpha, \lambda) = \frac{\lambda^{\alpha}}{\Gamma(\alpha)} x^{\alpha-1} \exp(-\lambda x), \quad x > 0.$$

• Bias expressions (Giles and Feng, 2009):

$$\mathcal{B}(\widehat{\alpha}) = \frac{\alpha \left[\Psi'(\alpha) - \alpha\Psi''(\alpha)\right] - 2}{2n \left[\alpha\Psi'(\alpha) - 1\right]^2} \tag{24}$$

and

$$\mathcal{B}\left(\widehat{\lambda}\right) = \frac{\lambda \left[2\alpha \left(\Psi'(\alpha)\right)^2 - 3\Psi'(\alpha) - \alpha\Psi''(\alpha)\right]}{2n \left[\alpha\Psi'(\alpha) - 1\right]^2}. \tag{25}$$

Using the data set from Delignette-Muller et al. (2008), we have $n = 254$, $\widehat{\alpha} = 4.0083$, $\widehat{\lambda} = 0.0544$, $\widehat{se}(\widehat{\alpha}) = 0.3413$ and $\widehat{se}\left(\widehat{\lambda}\right) = 0.0049$. Evaluating the analytical expressions (24), (25) and the `coxsnell.bc()` function, we have, respectively,

```
gamma.bc(n = 254, mle = c(4.0082, 0.0544))
##     alpha    lambda
## 0.0448278 0.0006618
pdf <- quote((lambda^alpha) / gamma(alpha) * x^(alpha - 1) *
             exp(-lambda *x))
lpdf <- quote(alpha * log(lambda) - lgamma(alpha) + alpha * log(x) -
              lambda * x)
coxsnell.bc(density = pdf, logdensity = lpdf, n = 254,
            parms = c("alpha", "lambda"), mle = c(4.0082, 0.0544),
            lower = 0)$bias
##     alpha    lambda
## 0.0448278 0.0006618
```

18. Inverse gamma distribution with shape α and scale β

$$f(x \mid \alpha, \beta) = \frac{1}{\Gamma(\alpha)\,\beta^{\alpha}} x^{\alpha-1} \exp\left(-\frac{x}{\beta}\right), \quad x > 0.$$

• Bias expressions (Stočsić and Cordeiro, 2009):

$$\mathcal{B}(\widehat{\alpha}) = \frac{-0.5\,\alpha^2\,\Psi''(\alpha) + 0.5\,\Psi'(\alpha)\,\alpha - 1}{n\,\alpha\,\left(\Psi'(\alpha) - 1\right)^2} \tag{26}$$

and

$$\mathcal{B}\left(\widehat{\beta}\right) = \frac{\beta\left(-0.5\,\alpha\,\Psi''(\alpha) - 1.5\,\Psi'(\alpha) + \left(\Psi'(\alpha)\right)^2\alpha\right)}{n\left(\Psi'(\alpha)\,\alpha - 1.0\right)^2}. \tag{27}$$

Using the data set from Kumagai and Matsunaga (1995), we have $n = 31$, $\widehat{\alpha} = 1.0479$, $\widehat{\beta} = 5.491$, $\widehat{se}(\widehat{\alpha}) = 0.2353$ and $\widehat{se}\left(\widehat{\beta}\right) = 1.5648$. Evaluating the analytical expressions (26), (27) and the `coxsnell.bc()` function, we have, respectively,

```
invgamma.bc(n = 31, mle = c(5.4901, 1.0479))
##    beta   alpha
## 0.60849 0.08388
pdf <- quote(beta^alpha / gamma(alpha) * x^(-alpha - 1) *
             exp(-beta / x))
lpdf <- quote(alpha * log(beta) - lgamma(alpha) -
              alpha * log(x) - beta / x)
```

```
coxsnell.bc(density = pdf, logdensity = lpdf, n = 31,
            parms = c("beta", "alpha"), mle = c(5.4901, 1.0479),
            lower = 0)$bias
##    beta   alpha
## 0.60847 0.08388
```

19. Lomax distribution with shape α and scale β

$$f(x \mid \alpha, \beta) = \alpha\,\beta\,(1+\beta\,x)^{-(\alpha+1)}, \quad x > 0.$$

• Bias expressions (Giles et al., 2013):

$$\mathcal{B}\,(\widehat{\alpha}) = \frac{2\,\alpha\;(\alpha+1)\,\left(\alpha^2+\alpha-2\right)}{(\alpha+3)\,n} \tag{28}$$

and

$$\mathcal{B}\left(\widehat{\beta}\right) = -\frac{2\,\beta\;(\alpha+1.6485)\,(\alpha+0.3934)\,(\alpha-1.5419)}{n\,\alpha\;(\alpha+3)}. \tag{29}$$

Using the data set from Tahir et al. (2016), we have $n = 179$, $\widehat{\alpha} = 4.9103$, $\widehat{\beta} = 0.0028$, $\widehat{se}\,(\widehat{\alpha}) = 0.6208$ and $\widehat{se}\left(\widehat{\beta}\right) = 3.4803 \times 10^{-4}$. Evaluating the analytical expressions (28), (29) and the `coxsnell.bc()` function, we have, respectively,

```
lomax.bc(n = 179, mle = c(4.9103, 0.0028))
##      alpha       beta
##  1.281e+00 -9.438e-05
pdf <- quote(alpha * beta / (1 + beta * x)^(alpha + 1))
lpdf <- quote(log(alpha) + log(beta) - (alpha + 1) *
              log(1 + beta * x))
coxsnell.bc(density = pdf, logdensity = lpdf, n = 179,
            parms = c("alpha", "beta"), mle = c(4.9103, 0.0028),
            lower = 0)$bias
##      alpha       beta
##  1.281e+00 -9.439e-05
```

20. Weighted Lindley distribution with shape α and scale θ

$$f(x \mid \alpha, \theta) = \frac{\theta^{\alpha+1}}{(\theta+\alpha)\,\Gamma(\alpha)}\,x^{\alpha-1}\,(1+x)\,\exp(-\theta x), \quad x > 0.$$

• For bias expressions, see (Wang and Wang, 2017):
Using the data set from Ghitany et al. (2013), we have $n = 69$, $\widehat{\alpha} = 22.8889$, $\widehat{\theta} = 9.6246$, $\widehat{se}\,(\widehat{\alpha}) = 3.9507$ and $\widehat{se}\left(\widehat{\theta}\right) = 1.6295$. Evaluating the analytical expressions and the `coxsnell.bc` function, we have, respectively,

```
wlindley.bc(n = 69, mle = c(22.8889, 9.6246))
##  alpha  theta
## 1.0070 0.4167
pdf <- quote(theta^(alpha + 1) / ((theta + alpha) * gamma(alpha)) *
             x^(alpha - 1) * (1 + x) * exp(-theta * x))
lpdf <- quote((alpha + 1) * log(theta) + alpha * log(x) -
              log(theta + alpha) - lgamma(alpha) - theta * x)
coxsnell.bc(density = pdf, logdensity = lpdf, n = 69,
            parms = c("alpha", "theta"), mle = c(22.8889, 9.6246),
            lower = 0)$bias
##  alpha  theta
## 1.0068 0.4166
```

21. Generalized Rayleigh with shape α and scale θ

$$f(x \mid \beta, \mu) = \frac{2\,\theta^{\alpha+1}}{\Gamma(\alpha+1)}\,x^{2\,\alpha+1}\,\exp\left(-\theta\,x^2\right), \quad x > 0.$$

• For bias expressions, see (Xiao and Giles, 2014):

Using the data set from Gomes et al. (2014), we have $n = 384$, $\widehat{\theta} = 0.5195$, $\widehat{\alpha} = 0.0104$, $\widehat{se}\left(\widehat{\theta}\right) = 0.2184$ and $\widehat{se}\left(\widehat{\alpha}\right) = 0.0014$. Evaluating the analytical expressions and the `coxsnell.bc()` function, we have, respectively,

```
generalizedrayleigh.bc(n =  384, mle = c(0.5195, 0.0104))
##     alpha     theta
## 1.035e-02 8.865e-05
pdf <- quote(2 * theta^(alpha + 1) / gamma(alpha + 1) *
             x^(2 * alpha + 1) * exp(-theta * x^2 ))
lpdf <- quote((alpha + 1) * log(theta) - lgamma(alpha + 1) +
              2 * alpha * log(x) - theta * x^2)
coxsnell.bc(density = pdf, logdensity = lpdf,  n = 384,
            parms = c("alpha", "theta"), mle = c(0.5195, 0.0104),
            lower = 0)$bias
##     alpha     theta
## 1.035e-02 8.865e-05
```

22. Weibull distribution with shape β and scale μ

$$f(x \mid \beta, \mu) = \frac{\beta}{\mu^{\beta}} x^{\beta-1} \exp\left(-\frac{x}{\mu}\right)^{\beta}, \quad x > 0.$$

• Bias expressions (the expressions below differs from Stočsić and Cordeiro (2009)):

$$\mathcal{B}\left(\widehat{\mu}\right) = \frac{\mu\left(0.5543324495 - 0.3698145397\,\beta\right)}{n\,\beta^2} \tag{30}$$

and

$$\mathcal{B}\left(\widehat{\beta}\right) = \frac{1.379530692\,\beta}{n}. \tag{31}$$

From Datta and Datta (2013), we have $n = 50$, $\widehat{\mu} = 2.5752$, $\widehat{\beta} = 38.0866$, $\widehat{se}\left(\widehat{\mu}\right) = 0.2299$ and $\widehat{se}\left(\widehat{\beta}\right) = 2.2299$. Evaluating the analytical expression (30), (31) and the `coxsnell.bc()` function, we have, respectively,

```
weibull.bc(n = 50, mle = c(38.0866, 2.5751))
##       mu     beta
## -0.04572  0.07105
pdf <- quote(beta / mu^beta * x^(beta - 1) *
             exp(-(x / mu)^beta))
lpdf <- quote(log(beta) - beta * log(mu) + beta * log(x) -
              (x / mu)^beta)
coxsnell.bc(density = pdf, logdensity = lpdf, n = 50,
            parms = c("mu", "beta"), mle = c(38.0866, 2.5751),
            lower = 0)$bias
##       mu     beta
## -0.04572  0.07105
```

23. Inverse Weibull distribution with shape β and scale μ

$$f(x \mid \beta, \alpha) = \beta\,\mu^{\beta}\,x^{-(\beta+1)} \exp\left[-\left(\frac{\mu}{x}\right)^{\beta}\right], \quad x > 0.$$

• Bias expressions (not previously reported in the literature):

$$\mathcal{B}\left(\widehat{\beta}\right) = \frac{1.379530690\,\beta}{n} \tag{32}$$

and

$$\mathcal{B}\left(\widehat{\mu}\right) = \frac{\mu\,\left(0.3698145391\,\beta + 0.5543324494\right)}{n\beta^2}. \tag{33}$$

Using the data set from Nichols and Padgett (2006), we have $n = 100$, $\widehat{\beta} = 1.769$, $\widehat{\mu} = 1.8917$, $\widehat{se}\left(\widehat{\beta}\right) = 0.1119$ and $\widehat{se}\left(\widehat{\mu}\right) = 0.1138$. Evaluating the analytical expressions (32), (33) and the `coxsnell.bc()` function, we have, respectively,

```
inverseweibull.bc(n = 100, mle = c(1.7690, 1.8916))
##     beta       mu
## 0.024404 0.007305
pdf <- quote(beta * mu^beta * x^(-beta - 1) *
             exp(-(mu / x)^beta))
lpdf <- quote(log(beta) + beta * log(mu) - beta * log(x) -
              (mu / x)^beta)
coxsnell.bc(density = pdf, logdensity = lpdf, n = 100,
            parms = c("beta", "mu"), mle = c(1.7690, 1.8916),
            lower = 0)$bias
##     beta       mu
## 0.024404 0.007305
```

24. Generalized half-normal distribution with shape α and scale θ

$$f(x \mid \alpha, \theta) = \sqrt{\frac{2}{\pi}}\, \frac{\alpha}{\theta^{\alpha}}\, x^{\alpha-1} \exp\left[-\frac{1}{2}\left(\frac{x}{\theta}\right)^{2\alpha}\right].$$

• Bias expressions (Mazucheli and Dey, 2017):

$$\mathcal{B}\left(\widehat{\alpha}\right) = 1.483794456\, \frac{\alpha}{n} \tag{34}$$

and

$$\mathcal{B}\left(\widehat{\theta}\right) = (0.2953497661 - 0.3665611957\, \alpha)\, \frac{\theta}{n\, \alpha^2}. \tag{35}$$

Using the data set from Nadarajah (2008a), we have $n = 119$, $\widehat{\alpha} = 3.8096$, $\widehat{\theta} = 4.9053$, $\widehat{se}\left(\widehat{\alpha}\right) = 0.2758$ and $\widehat{se}\left(\widehat{\theta}\right) = 0.0913$. Evaluating the analytical expressions (34), (35) and the `coxsnell.bc()` function, we have, respectively,

```
genhalfnormal.bc(n = 119, mle = c(3.8095, 4.9053))
##     alpha     theta
##  0.047500 -0.003127
pdf <- quote(sqrt(2 / pi) * alpha / theta^alpha * x^(alpha - 1)*
             exp(- 0.5 * (x / theta)^(2 * alpha) ))
lpdf <- quote(log(alpha) - alpha * log(theta) + alpha * log(x) -
              0.5 * (x / theta)^(2 * alpha))
coxsnell.bc(density = pdf, logdensity = lpdf, n = 119,
            parms = c("alpha", "theta"), mle = c(3.8095, 4.9053),
            lower = 0)$bias
##     alpha     theta
##  0.047500 -0.003127
```

25. Inverse generalized half-normal distribution with shape α and scale θ

$$f(x \mid \alpha, \theta) = \sqrt{\frac{2}{\pi}}\, \left(\frac{\alpha}{x}\right) \left(\frac{1}{\theta\, x}\right)^{\alpha} \exp\left[-\frac{1}{2}\left(\frac{1}{\theta\, x}\right)^{2\alpha}\right], \quad x > 0.$$

• For bias expressions (not previously reported in the literature, see the "analyticalBC.R" file. Using the data set from Nadarajah et al. (2011), we have $n = 20$, $\widehat{\alpha} = 3.0869$, $\widehat{\theta} = 0.6731$, $\widehat{se}\left(\widehat{\alpha}\right) = 0.5534$ and $\widehat{se}\left(\widehat{\theta}\right) = 0.0379$. Evaluating the analytical expressions and the `coxsnell.bc()` function, we have, respectively,

```
invgenhalfnormal.bc(n = 20, mle = c(3.0869, 0.6731))
##     alpha     theta
##  0.229016 -0.002953
pdf <- quote(sqrt(2) * pi^(-0.5) * alpha * x^(-alpha - 1) *
             exp(-0.5 * x^(-2 * alpha) * (1 / theta)^(2 * alpha)) *
             theta^(-alpha))
lpdf <- quote(log(alpha) - alpha  * log(x) - 0.5e0 / (x^alpha)^2*
              theta^(-2 * alpha) - alpha * log(theta))
coxsnell.bc(density = pdf, logdensity = lpdf, n = 20,
            parms = c("alpha", "theta"), mle = c(3.0869, 0.6731),
            lower = 0)$bias
```

```
##     alpha     theta
##  0.229016 -0.002953
```

26. Marshall-Olkin extended exponential distribution with shape α and rate λ

$$f(x \mid \alpha, \lambda) = \frac{\lambda\,\alpha\,\exp(-\lambda\,x)}{[1-(1-\alpha)\,\exp(-\lambda\,x)]^2}, \quad x > 0.$$

• For bias expressions (not previously reported in the literature, see the "analyticalBC.R" file. Using the data set from Linhart and Zucchini (1986), we have $n = 20$, $\widehat{\alpha} = 0.2782$, $\widehat{\lambda} = 0.0078$, $\widehat{se}(\widehat{\alpha}) = 0.2321$ and $\widehat{se}\left(\widehat{\lambda}\right) = 0.0049$. Evaluating the analytical expressions and the `coxsnell.bc()` function, we have, respectively,

```
moeexp.bc(n = 20, mle = c(0.2781, 0.0078))
##    alpha   lambda
## 0.210919 0.003741
pdf <- quote(alpha * lambda * exp(-x * lambda) /
             ((1- (1 - alpha) * exp(- x * lambda)))^2)
lpdf <- quote(log(alpha) + log(lambda) - x * lambda -
              2 * log((1 - (1-alpha) * exp(- x * lambda))))
coxsnell.bc(density = pdf, logdensity = lpdf, n = 20,
            parms = c("alpha", "lambda"), mle = c(0.2781, 0.0078),
            lower = 0)$bias
##   alpha  lambda
## 0.21086 0.00374
```

27. Beta distribution with shapes α and β

$$f(x \mid \alpha, \beta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\,\Gamma(\beta)}\,x^{\alpha-1}\,(1-x)^{\beta-1}, \quad 0 < x < 1.$$

• For bias expressions, see (Cordeiro et al., 1997).
Using the data set from Javanshiri et al. (2015), we have $n = 48$, $\widehat{\alpha} = 5.941$, $\widehat{\beta} = 21.2024$, $\widehat{se}(\widehat{\alpha}) = 1.1812$ and $\widehat{se}\left(\widehat{\beta}\right) = 4.3462$. Evaluating the analytical expressions in Cordeiro et al. (1997), our analytical expressions and the `coxsnell.bc()` function, we have, respectively,

```
beta.gauss.bc(n = 48, mle = c(5.941, 21.2024))
##  alpha   beta
## -4.784 -4.125
beta.bc(n = 48, mle = c(5.941, 21.2024))
##  alpha   beta
## 0.3582 1.3315
pdf <- quote(gamma(alpha + beta) / (gamma(alpha) * gamma(beta)) *
             x^(alpha - 1) * (1 - x)^(beta - 1))
lpdf <- quote(lgamma(alpha + beta) - lgamma(alpha) -
              lgamma(beta) + alpha * log(x) + beta * log(1 - x))
coxsnell.bc(density = pdf, logdensity = lpdf, n = 48,
            parms = c("alpha", "beta"), mle = c(5.941, 21.2024),
            lower = 0, upper =  1)$bias
##  alpha   beta
## 0.3582 1.3315
```

28. Kumaraswamy distribution with shapes α and β

$$f(x \mid \alpha, \beta) = \alpha\,\beta\,x^{\alpha-1}\,(1-x^{\alpha})^{\beta-1}, \quad 0 < x < 1.$$

• For bias expressions, see (Lemonte, 2011).
Using the data set from Wang et al. (2017), we have $n = 20$, $\widehat{\alpha} = 6.3478$, $\widehat{\beta} = 4.4898$, $\widehat{se}(\widehat{\alpha}) = 1.5576$ and $\widehat{se}\left(\widehat{\beta}\right) = 2.0414$. Evaluating the analytical expressions and the `coxsnell.bc()` function, we have, respectively,

```
kum.bc(n = 20, mle = c(6.3478, 4.4898))
##   alpha    beta
##  -6.573 -13.323
```

```
pdf <- quote(alpha * beta * x^(alpha - 1) *
             (1 - x^alpha)^(beta - 1))
lpdf <- quote(log(alpha) + log(beta) + alpha * log(x) + (beta - 1) *
              log(1 - x^alpha))
coxsnell.bc(density = pdf, logdensity = lpdf, n = 20,
            parms = c("alpha", "beta"), mle = c(6.3478, 4.4898),
            lower = 0, upper = 1)$bias
## alpha  beta
## 0.514 1.013
```

29. Inverse beta distribution with shapes α and β

$$f(x \mid \alpha, \beta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\,\Gamma(\beta)}\, x^{\alpha-1}\,(1+x)^{-(\alpha+\beta)}, \quad x > 0.$$

• For bias expressions, see (Stočsić and Cordeiro, 2009).

Using the data set from Nadarajah (2008b), we have $n = 116$, $\widehat{\alpha} = 28.5719$, $\widehat{\beta} = 1.3783$, $\widehat{se}\,(\widehat{\alpha}) = 4.0367$ and $\widehat{se}\left(\widehat{\beta}\right) = 0.1637$. Evaluating the analytical expressions and the `coxsnell.bc()` function, we have, respectively,

```
invbeta.bc(n = 116, mle = c(28.5719, 1.3782))
##  alpha   beta
## 534.26  17.73
pdf <- quote(gamma(alpha + beta) * x^(alpha - 1) *
             (1 + x)^(- alpha - beta) / gamma(alpha)/gamma(beta))
lpdf <- quote(lgamma(alpha + beta) + alpha * log(x) -
              (alpha + beta) * log(1 + x) - lgamma(alpha) - lgamma(beta))
coxsnell.bc(density = pdf, logdensity = lpdf, n = 116,
            parms = c("alpha", "beta"), mle = c(28.5719, 1.3782),
            lower = 0)$bias
##  alpha   beta
## 0.8025 0.0306
```

30. Birnbaum-Saunders distribution with shape α and scale β

$$f(x \mid \alpha, \beta) = \frac{1}{2\,\alpha\,\beta\,\sqrt{2\,\pi}}\left[\left(\frac{\beta}{x}\right)^{1/2} + \left(\frac{\beta}{x}\right)^{3/2}\right] \exp\left[-\frac{1}{2,\alpha^2}\left(\frac{x}{\beta} + \frac{\beta}{x} - 2\right)\right], \quad x > 0.$$

• Bias expressions (Lemonte et al., 2007):

$$\mathcal{B}\,(\widehat{\alpha}) = -\frac{\alpha}{4\,n}\left(1 + \frac{2+\alpha^2}{\alpha\,(2\,\pi)^{-1/2}\,h(\alpha)+1}\right) \tag{36}$$

and

$$\mathcal{B}\left(\widehat{\beta}\right) = \frac{\beta^2\,\alpha^2}{2\,n\,\left[\alpha\,(2\,\pi)^{-1/2}\,h(\alpha)+1\right]}, \tag{37}$$

where

$$h(\alpha) = \alpha\sqrt{\frac{\pi}{2}} - \pi\,e^{2/\alpha^2}\left[1 - \Phi\left(\frac{2}{\alpha}\right)\right].$$

Using the data set from Gross and Clark (1976), we have $n = 20$, $\widehat{\alpha} = 0.3149$, $\widehat{\beta} = 1.8105$, $\widehat{se}\,(\widehat{\alpha}) = 0.0498$ and $\widehat{se}\left(\widehat{\beta}\right) = 0.1259$. Evaluating the analytical expressions (36), (37) and the `coxsnell.bc()` function, we have, respectively,

```
birnbaumsaunders.bc(n = 20, mle = c(0.3148, 1.8104))
##     alpha      beta
## -0.011991  0.004374
pdf <- quote(1 / (2 * alpha * beta * sqrt(2 * pi)) *
             ((beta / x)^0.5 + (beta / x)^1.5) *
             exp(- 1/(2 * alpha^2) * (x / beta + beta/ x - 2)))
lpdf <- quote(-log(alpha) - log(beta) - 1 / (2 * alpha^2) *
              (x / beta + beta/ x - 2) + log((beta / x)^0.5 +
```

```
              (beta / x)^1.5))
coxsnell.bc(density = pdf, logdensity = lpdf, n = 20,
            parms = c("alpha", "beta"), mle = c(0.3148, 1.8104),
            lower = 0)$bias
##     alpha      beta
## -0.011991  0.004374
```

31. Generalized Pareto distribution with shape ξ and scale σ

$$f(x \mid \xi, \sigma) = \frac{1}{\sigma}\left(1 + \frac{\xi\, x}{\sigma}\right)^{-(1/\xi+1)}, \quad x > 0,\ \xi \neq 0.$$

• Bias expressions (Giles et al., 2016):

$$\mathcal{B}\left(\widehat{\xi}\right) = -\frac{(1+\xi)\,(3+\xi)}{n\,(1+3\,\xi)} \tag{38}$$

and

$$\mathcal{B}\left(\widehat{\sigma}\right) = -\frac{\sigma\,\left(3+5\,\xi+4\,\xi^2\right)}{n\,(1+3\,\xi)}. \tag{39}$$

Using the data set from Ross and Lott (2003), we have $n = 58$, $\widehat{\xi} = 0.736$, $\widehat{\sigma} = 1.709$, $\widehat{se}\left(\widehat{\xi}\right) = 0.223$ and $\widehat{se}\left(\widehat{\sigma}\right) = 0.41$. Evaluating the analytical expressions (38), (39) and the `coxsnell.bc()` function, we have, respectively,

```
genpareto.bc(n = 58, mle = c(0.736, 1.709))
##       xi    sigma
## -0.03486  0.08126
pdf <- quote(1 / sigma * (1 + xi * x / sigma )^(-(1 + 1 / xi)))
Rlpdf <- quote(-log(sigma) - (1 + 1 / xi) * log(1 + xi * x / sigma))
coxsnell.bc(density = pdf, logdensity = lpdf, n = 58,
            parms = c("xi", "sigma"), mle = c(0.736, 1.709),
            lower = 0)$bias
##       xi    sigma
## -0.03486  0.08126
```

Additional Applications

In this section, we present additional numerical results returned by `cosnell.bc()`, `observed.varc()` and `expected.varcov()`. For the data describing the times between successive electric pulses on the surface of isolated muscle fiber (Cox and Lewis, 1966; Jørgensen, 1982), we fitted the exponentiated Weibull, Marshall-Olkin extended Weibull, Weibull, Marshall-Olkin extended exponential and exponential distributions. These distributions were also fitted by Cordeiro and Lemonte (2013). There are 799 observations and for each distribution we report the MLEs, the bias corrected MLEs, the observed variance-covariance obtained from the numerical Hessian $\boldsymbol{H}_1^{-1}\left(\widehat{\boldsymbol{\theta}}\right)$, the observed variance-covariance obtained from the analytical Hessian $\boldsymbol{H}_2^{-1}\left(\widehat{\boldsymbol{\theta}}\right)$, the expected variance-covariance $\boldsymbol{I}^{-1}\left(\widehat{\boldsymbol{\theta}}\right)$ and the expected variance-covariance evaluated at the bias corrected MLEs $\boldsymbol{I}^{-1}\left(\widetilde{\boldsymbol{\theta}}\right)$. The MLEs and the $\boldsymbol{H}_1^{-1}\left(\widehat{\boldsymbol{\theta}}\right)$ matrix were obtained by the **fitdistrplus** package (Delignette-Muller et al., 2017). The R codes used to obtain the numerical results are available in the supplementary material.

It is important to emphasize that for the Marshall-Olkin extended Weibull and exponentiated Weibull distributions, it is not possible to obtain analytical expressions for bias corrections. The exponentiated-Weibull family was proposed by Mudholkar and Srivastava (1993). Its probability density function is:

$$f(x \mid \lambda, \beta, \alpha) = \alpha\,\beta\,\lambda\,x^{\beta-1}\,\mathrm{e}^{-\lambda\,x^{\beta}}\left(1-\mathrm{e}^{-\lambda\,x^{\beta}}\right)^{\alpha-1},$$

where $\lambda > 0$ is the scale parameter and $\beta > 0$ and $\alpha > 0$ are the shape parameters. The Marshall-Olkin extended Weibull distribution was introduced by Marshall and Olkin (1997). Its probability density

function is:

$$f(x \mid \lambda, \beta, \alpha) = \frac{\alpha\,\beta\,\lambda\,x^{\beta-1}\,\mathrm{e}^{-\lambda\,x^{\beta}}}{\left(1-\overline{\alpha}\,\mathrm{e}^{-\lambda\,x^{\beta}}\right)^{2}},$$

where $\lambda > 0$ is the scale parameter, $\beta > 0$ is the shape parameter, $\alpha > 0$ is an additional shape parameter and $\overline{\alpha} = 1 - \alpha$.

The fitted parameter estimates and their bias corrected estimates are shown in Table 1. We see that the bias corrected MLEs for α and λ of the MOE-Weibull and exp-Weibull distributions are quite different from the original MLEs.

Table 1: MLEs and bias corrected MLEs.

Distribution	$\widehat{\alpha}$	$\widehat{\beta}$	$\widehat{\lambda}$	$\widetilde{\alpha}$	$\widetilde{\beta}$	$\widetilde{\lambda}$
MOE-Weibull	0.3460	1.3247	0.0203	0.3283	1.3240	0.0188
exp-Weibull	1.9396	0.7677	0.2527	1.8973	0.7625	0.2461
Weibull	–	1.0829	0.0723	–	1.0811	0.0723
MOE-exponential	1.1966	–	0.0998	1.1820	–	0.0994
exponential	–	–	0.0913	–	–	0.0912

It is important to assess the accuracy of MLEs. The two common ways for this are through the inverse observed Fisher information and the inverse expected Fisher information matrices. The results below show large differences between the observed $\boldsymbol{H}^{-1}$ and expected $\boldsymbol{I}^{-1}$ information matrices. As demonstrated by Cao (2013), the $\boldsymbol{I}^{-1}$ outperforms the $\boldsymbol{H}^{-1}$ under a mean squared error criterion, hence with **mle.tools** the researchers may choose one of them and not use the easier. Furthermore, in general, we observe that the bias corrected MLEs decrease the variance of estimates.

• Exponentiated Weibull distribution:

$$\boldsymbol{H}_1^{-1}\left(\widehat{\boldsymbol{\theta}}\right) = \begin{bmatrix} 0.00726 & -0.00717 & 0.03564 \\ -0.00717 & 0.00718 & -0.03493 \\ 0.03564 & -0.03493 & 0.18045 \end{bmatrix}, \quad \boldsymbol{H}_2^{-1}\left(\widehat{\boldsymbol{\theta}}\right) = \begin{bmatrix} 0.00729 & -0.00720 & 0.03579 \\ -0.00720 & 0.00721 & -0.03509 \\ 0.03579 & -0.03509 & 0.18120 \end{bmatrix},$$

$$\boldsymbol{I}^{-1}\left(\widehat{\boldsymbol{\theta}}\right) = \begin{bmatrix} 0.00532 & -0.00524 & 0.02609 \\ -0.00524 & 0.00527 & -0.02545 \\ 0.02609 & -0.02545 & 0.13333 \end{bmatrix}, \quad \boldsymbol{I}^{-1}\left(\widetilde{\boldsymbol{\theta}}\right) = \begin{bmatrix} 0.00510 & -0.00510 & 0.02482 \\ -0.00510 & 0.00519 & -0.02454 \\ 0.02482 & -0.02454 & 0.12590 \end{bmatrix}.$$

• Marshall-Olkin extended Weibull distribution:

$$\boldsymbol{H}_1^{-1}\left(\widehat{\boldsymbol{\theta}}\right) = \begin{bmatrix} 0.00004 & -0.00036 & 0.00052 \\ -0.00036 & 0.00361 & -0.00430 \\ 0.00052 & -0.00430 & 0.00748 \end{bmatrix}, \quad \boldsymbol{H}_2^{-1}\left(\widehat{\boldsymbol{\theta}}\right) = \begin{bmatrix} 0.00005 & -0.00047 & 0.00068 \\ -0.00047 & 0.00468 & -0.00582 \\ 0.00068 & -0.00582 & 0.00967 \end{bmatrix},$$

$$\boldsymbol{I}^{-1}\left(\widehat{\boldsymbol{\theta}}\right) = \begin{bmatrix} 0.00006 & -0.00056 & 0.00082 \\ -0.00056 & 0.00542 & -0.00699 \\ 0.00082 & -0.00699 & 0.01146 \end{bmatrix}, \quad \boldsymbol{I}^{-1}\left(\widetilde{\boldsymbol{\theta}}\right) = \begin{bmatrix} 0.00005 & -0.00051 & 0.00072 \\ -0.00051 & 0.00526 & -0.00651 \\ 0.00072 & -0.00651 & 0.01030 \end{bmatrix}.$$

• Weibull distribution:

$$\boldsymbol{H}_1^{-1}\left(\widehat{\boldsymbol{\theta}}\right) = \begin{bmatrix} 0.00004 & -0.00018 \\ -0.00018 & 0.00086 \end{bmatrix}, \quad \boldsymbol{H}_2^{-1}\left(\widehat{\boldsymbol{\theta}}\right) = \begin{bmatrix} 0.00004 & -0.00018 \\ -0.00018 & 0.00087 \end{bmatrix},$$

$$\boldsymbol{I}^{-1}\left(\widehat{\boldsymbol{\theta}}\right) = \begin{bmatrix} 0.00004 & -0.00018 \\ -0.00018 & 0.00089 \end{bmatrix}, \quad \boldsymbol{I}^{-1}\left(\widetilde{\boldsymbol{\theta}}\right) = \begin{bmatrix} 0.00004 & -0.00018 \\ -0.00018 & 0.00089 \end{bmatrix}.$$

• Marshall-Olkin extended exponential distribution:

$$\boldsymbol{H}_1^{-1}\left(\widehat{\boldsymbol{\theta}}\right) = \begin{bmatrix} 0.00004 & 0.00081 \\ 0.00081 & 0.02022 \end{bmatrix}, \quad \boldsymbol{H}_2^{-1}\left(\widehat{\boldsymbol{\theta}}\right) = \begin{bmatrix} 0.00004 & 0.00081 \\ 0.00081 & 0.02023 \end{bmatrix},$$

$$\boldsymbol{I}^{-1}\left(\widehat{\boldsymbol{\theta}}\right) = \begin{bmatrix} 0.00004 & 0.00083 \\ 0.00083 & 0.02094 \end{bmatrix}, \quad \boldsymbol{I}^{-1}\left(\widetilde{\boldsymbol{\theta}}\right) = \begin{bmatrix} 0.00004 & 0.00082 \\ 0.00082 & 0.02047 \end{bmatrix}.$$

- Exponential distribution:

$$\boldsymbol{H}_1^{-1}\left(\widehat{\boldsymbol{\theta}}\right) = 0.000010433, \qquad \boldsymbol{H}_2^{-1}\left(\widehat{\boldsymbol{\theta}}\right) = 0.000010436,$$

$$\boldsymbol{I}^{-1}\left(\widehat{\boldsymbol{\theta}}\right) = 0.000010436, \qquad \boldsymbol{I}^{-1}\left(\widetilde{\boldsymbol{\theta}}\right) = 0.000010410.$$

Concluding Remarks

As pointed out by several works in the literature, the Cox-Snell methodology, in general, is efficient for reducing the bias of the MLEs. However, the analytical expressions are either notoriously cumbersome or even impossible to deduce. To the best of our knowledge, there are only two alternatives to obtain the analytical expressions automatically, those presented in Stočsić and Cordeiro (2009) and Johnson et al. (2012a). They use the commercial softwares Maple (Maple, 2017) and Mathematica (Wolfram Research, Inc., 2010).

In order to calculate the bias corrected estimates in a simple way, Mazucheli (2017) developed an R (R Core Team, 2016) package, uploaded to CRAN on 2 February, 2017. Its main function, `coxsnell.bc()`, evaluates the bias corrected estimates. The usefulness of this function has been tested for thirty one continuous probability distributions. Bias expressions, for most of them, are available in the literature.

It is well known that the Fisher information can be computed using the first or second order derivatives of the log-likelihood function. In our implementation, the functions `expected.varcov()` and `coxsnell.bc()` are using the second order derivatives, analytically returned by the `D()` function. In a future work, we intend to check if there is any gain in calculating the Fisher information from the first order derivatives of the log-hazard rate function or from the first order derivatives of the log-reversed-hazard rate function. Efron and Johnstone (1990) showed that the Fisher information can be computed using the hazard rate function. Gupta et al. (2004) computed the Fisher information from the first order derivatives of the log-reversed-hazard rate function. In general, expressions of the first order derivatives of the log-hazard rate function (log-reversed-hazard rate function) are simpler than second order derivatives of the log-likelihood function. In this sense, the `integrate()` function can work better. It is important to point out that the hazard rate function and the reversed hazard rate function are given, respectively, by $h\left(x \mid \boldsymbol{\theta}\right) = -\frac{d}{dx}\log\left[S(x \mid \boldsymbol{\theta})\right]$ and $\overline{h}\left(x \mid \boldsymbol{\theta}\right) = \frac{d}{dx}\log\left[F(x \mid \boldsymbol{\theta})\right]$, where $S\left(x \mid \boldsymbol{\theta}\right)$ and $F\left(x \mid \boldsymbol{\theta}\right)$ are, respectively, the survival function and the cumulative distribution function.

In the next version of **mle.tools**, we will include, using analytical first and second-order partial derivatives, the following:

- the MLEs of $g\left(\boldsymbol{\theta}\right)$ and $\text{Var}\left[g\left(\boldsymbol{\theta}\right)\right]$,
- the negative log likelihood value $-2\log(L)$,
- the Akaike's information criterion $-2\log(L) + 2p$,
- the corrected Akaike's information criterion $-2\log(L) + \frac{2np}{n-p-1}$,
- the Schwarz's Bayesian information criterion $-2\log(L) + p\log(n)$,
- the Hannan-Quinn information criterion $-2\log(L) + 2\log\log(n)p$,

where L is the value of the likelihood function evaluated at the MLEs, n is the number of observations, and p is the number of estimated parameters.

Also, the next version of the package will incorporate analytical expressions for the distributions studied in Section 38.4 implemented in the supplementary file "analyticalBC.R".

Bibliography

J. A. Achcar, S. R. Lopes, J. Mazucheli, and R. R. Linhares. A Bayesian approach for stable distributions: Some computational aspects. *Open Journal of Statistics*, 3(04):268, 2013. [p273]

A. Alzaatreh, M. Mansoor, M. Tahir, M. Zubair, and S. Ali. The gamma half-Cauchy distribution: Properties and applications. *HACETTEPE Journal of Mathematics and Statistics*, 45(4):1143–1159, 2016. [p275]

M. Bader and A. Priest. Statistical aspects of fibre and bundle strength in hybrid composites. *Progress in Science and Engineering of Composites*, pages 1129–1136, 1982. [p274]

D. K. Bhaumik, K. Kapur, and R. D. Gibbons. Testing parameters of a gamma distribution for small samples. *Technometrics*, 51(3):326–334, 2009. [p274]

X. Cao. *Relative Performance of Expected and Observed Fisher Information in Covariance Estimation for Maximum Likelihood Estimates*. PhD thesis, Johns Hopkins University, 2013. [p284]

R. Chhikara and J. Folks. The inverse Gaussian distribution as a lifetime model. *Technometrics*, 19(4): 461–468, 1977. [p276]

G. M. Cordeiro and F. Cribari-Neto. *An Introduction to Bartlett Correction and Bias Reduction*. Springer-Verlag, New York, 2014. [p268]

G. M. Cordeiro and R. dos Santos Brito. The Beta power distribution. *Brazilian Journal of Probability and Statistics*, 26(1):88–112, 2012. [p273]

G. M. Cordeiro and R. Klein. Bias correction in ARMA models. *Statistics & Probability Letters*, 19(3): 169–176, 1994. [p269]

G. M. Cordeiro and A. J. Lemonte. On the Marshall-Olkin extended Weibull distribution. *Statistical Papers*, 54(2):333–353, 2013. [p283]

G. M. Cordeiro, E. C. Da Rocha, J. G. C. Da Rocha, and F. Cribari-Neto. Bias-corrected maximum likelihood estimation for the beta distribution. *Journal of Statistical Computation and Simulation*, 58(1): 21–35, 1997. [p268, 271, 281]

D. R. Cox and D. V. Hinkley. *Theoretical Statistics*. CRC Press, 1979. [p269]

D. R. Cox and P. A. W. Lewis. *The Statistical Analysis of Series of Events*. Springer-Verlag, Netherlands, 1966. [p283]

D. R. Cox and E. J. Snell. A general definition of residuals. *Journal of the Royal Statistical Society B*, 30(2): 248–275, 1968. [p268, 269]

F. Cribari-Neto and K. L. P. Vasconcellos. Nearly unbiased maximum likelihood estimation for the beta distribution. *Journal of Statistical Computation and Simulation*, 72(2):107–118, 2002. [p268]

D. Datta and D. Datta. Comparison of Weibull distribution and exponentiated Weibull distribution based estimation of mean and variance of wind data. *International Journal of Energy, Information and Communications*, 4(4):1–11, 2013. [p279]

M. Delignette-Muller, M. Cornu, A. S. S. Group, and others. Quantitative risk assessment for Escherichia coli o157: H7 in frozen ground beef patties consumed by young children in french households. *International Journal of Food Microbiology*, 128(1):158–164, 2008. [p277]

M. L. Delignette-Muller, C. Dutang, and A. Siberchicot. **fitdistrplus***: Help to Fit of a Parametric Distribution to Non-Censored or Censored Data*, 2017. R package version 1.0-8. [p283]

A. W. F. Edwards. *Likelihood (Expanded Edition)*. Johns Hopkins University Press, Baltimore, 1992. [p268]

B. Efron and I. M. Johnstone. Fisher's information in terms of the hazard rate. *The Annals of Statistics*, 18(1):38–62, 1990. [p285]

M. E. Ghitany, B. Atieh, and S. Nadarajah. Lindley distribution and its application. *Mathematics and Computers in Simulation*, 78(4):493–506, 2008. [p271]

M. E. Ghitany, D. K. Al-Mutairi, N. Balakrishnan, and L. J. Al-Enezi. Power Lindley distribution and associated inference. *Computational Statistics & Data Analysis*, 64:20–33, 2013. [p278]

D. E. Giles. A note on improved estimation for the Topp-Leone distribution. Technical report, Department of Economics, University of Victoria, 2012a. [p268, 273]

D. E. Giles. Bias reduction for the maximum likelihood estimators of the parameters in the half-logistic distribution. *Communication in Statistics - Theory and Methods*, 41(2):212–222, 2012b. [p268, 274]

D. E. Giles and H. Feng. Bias of the maximum likelihood estimators of the two-parameter gamma distribution revisited. Technical report, Department of Economics, University of Victoria, 2009. [p268, 277]

D. E. Giles, H. Feng, and R. T. Godwin. On the bias of the maximum likelihood estimator for the two-parameter Lomax distribution. *Communications in Statistics - Theory and Methods*, 42(11):1934–1950, 2013. [p268, 278]

D. E. Giles, H. Feng, and R. T. Godwin. Bias-corrected maximum likelihood estimation of the parameters of the generalized Pareto distribution. *Communications in Statistics – Theory and Methods*, 45(8): 2465–2483, 2016. [p268, 283]

A. E. Gomes, C. Q. da Silva, G. M. Cordeiro, and E. M. M. Ortega. A new lifetime model: The Kumaraswamy generalized Rayleigh distribution. *Journal of Statistical Computation and Simulation*, 84(2):290–309, 2014. [p279]

A. J. Gross and V. A. Clark. *Survival Distributions: Reliability Applications in the Biometrical Sciences*. John Wiley & Sons, 1976. [p282]

R. D. Gupta, R. C. Gupta, and P. G. Sankaran. Some characterization results based on factorization of the (reversed) hazard rate function. *Communication in Statistics - Theory and Methods*, 33, 2004. [p285]

Z. Javanshiri, A. Habibi Rad, and N. R. Arghami. Exp-Kumaraswamy distributions: Some properties and applications. *Journal of Sciences, Islamic Republic of Iran*, 26(1):57–69, 2015. [p281]

P. H. Johnson, Y. Qi, and Y. C. Chueh. Bias-corrected maximum likelihood estimation in actuarial science. Working papers, University of Illinois, Urbana-Champaign, Champaign, IL, 2012a. [p285]

P. H. Johnson, Y. Qi, and Y. C. Chueh. CSCK MLE bias calculation. Working papers, University of Illinois, Urbana-Champaign, Champaign, IL, 2012b. [p268, 272]

B. Jørgensen. *Statistical Properties of the Generalized Inverse Gaussian Distribution*. Springer-Verlag, 1982. [p283]

S. Kay. Asymptotic maximum likelihood estimator performance for chaotic signals in noise. *IEEE Transactions on Signal Processing*, 43(4):1009–1012, 1995. [p268]

S. Kumagai and I. Matsunaga. Changes in the distribution of short-term exposure concentration with different averaging times. *American Industrial Hygiene Association*, 56(1):24–31, 1995. [p277]

S. Kumagai, I. Matsunaga, K. Sugimoto, Y. Kusaka, and T. Shirakawa. Assessment of occupational exposures to industrial hazardous substances. III. On the frequency distribution of daily exposure averages (8-h TWA). *Japanese Journal of Industrial Health*, 31(4):216–226, 1989. [p276]

D. Kundu. *Discriminating between Normal and Laplace Distributions*, chapter 4, pages 65–79. Birkhäuser, Basel, 2005. [p275]

B. Lagos-Álvarez, M. D. Jiménez-Gamero, and V. Alba-Fernández. Bias correction in the type I generalized logistic distribution. *Communications in Statistics - Simulation and Computation*, 40(4): 511–531, 2011. [p268]

J. F. Lawless. *Statistical Models and Methods for Lifetime Data, Second Edition*, volume 362. John Wiley & Sons, 2011. [p272]

E. L. Lehmann. *Elements of Large-Sample Theory*. Springer-Verlag, 1999. [p268]

A. J. Lemonte. Improved point estimation for the Kumaraswamy distribution. *Journal of Statistical Computation and Simulation*, 81(12):1971–1982, 2011. [p268, 271, 281]

A. J. Lemonte, F. Cribari-Neto, and K. L. P. Vasconcellos. Improved statistical inference for the two-parameter Birnbaum-Saunders distribution. *Computational Statistics & Data Analysis*, 51(9): 4656–4681, 2007. [p268, 282]

H. Linhart and W. Zucchini. *Model Selection*. John Wiley & Sons, 1986. [p281]

Maple. *Maplesoft, a Division of Waterloo Maple Inc.* Waterloo, Ontario, 2017. [p268, 271, 285]

A. W. Marshall and I. Olkin. A new method for adding a parameter to a family of distributions with application to the exponential and Weibull families. *Biometrika*, 84(3):641–652, 1997. [p283]

J. Mazucheli. *Mle.tools: Expected/Observed Fisher Information and Bias-Corrected Maximum Likelihood Estimate(s)*, 2017. R package version 1.0.0. [p268, 285]

J. Mazucheli and S. Dey. Bias-corrected maximum likelihood estimation of the parameters of the generalized half-normal distribution. *submitted to Journal of Statistical Computation and Simulation*, 2017. [p268, 280]

G. S. Mudholkar and D. K. Srivastava. Exponentiated Weibull family for analyzing bathtub failure-rate data. *IEEE Transactions on Reliability*, 42(2):299–302, 1993. [p283]

S. Nadarajah. The model for fracture toughness. *Journal of Mechanical Science and Technology*, 22: 1255–1258, 2008a. [p280]

S. Nadarajah. A truncated inverted beta distribution with application to air pollution data. *Stochastic Environmental Research and Risk Assessment*, 22(2):285–289, 2008b. [p282]

S. Nadarajah, H. S. Bakouch, and R. Tahmasbi. A generalized Lindley distribution. *Sankhya B*, 73(2): 331–359, 2011. [p280]

M. D. Nichols and W. Padgett. A bootstrap control chart for Weibull percentiles. *Quality and reliability engineering international*, 22(2):141–151, 2006. [p279]

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2016. ISBN 3-900051-07-0. [p268, 285]

M. Z. Raqab, M. T. Madi, and D. Kundu. Estimation of P(Y $<$ X) for the three-parameter generalized exponential distribution. *Communications in Statistics - Theory and Methods*, 37(18):2854–2864, 2008. [p275]

J. Reath. *Improved Parameter Estimation of the Log-Logistic Distribution with Applications*. PhD thesis, Michigan Technological University, 2016. [p268, 276]

T. Ross and N. Lott. A climatology of 1980 – 2003 extreme weather and climate events. Technical Report 01, National Climatic Data Center, 2003. [p283]

K. Saha and S. Paul. Bias-corrected maximum likelihood estimator of the negative binomial dispersion parameter. *Biometrics*, 61(1):179–185, 2005. [p268]

J. Schwartz and D. E. Giles. Bias-reduced maximum likelihood estimation of the zero-inflated Poisson distribution. *Communications in Statistics - Theory and Methods*, 45(2):465–478, 2016. [p268]

J. Schwartz, R. T. Godwin, and D. E. Giles. Improved maximum-likelihood estimation of the shape parameter in the Nakagami distribution. *Journal of Statistical Computation and Simulation*, 83(3): 434–445, 2013. [p268]

R. Shanker. Shanker distribution and its applications. *International Journal of Statistics and Applications*, 5(6):338–348, 2015. [p272]

V. K. Sharma, S. K. Singh, U. Singh, and V. Agiwal. The inverse Lindley distribution: A stress-strength reliability model with application to head and neck cancer data. *Journal of Industrial and Production Engineering*, 32(3):162–173, 2015. [p272, 273]

B. D. Stočsić and G. M. Cordeiro. Using Maple and Mathematica to derive bias corrections for two parameter distributions. *Journal of Statistical Computation and Simulation*, 79(6):751–767, 2009. [p268, 271, 275, 276, 277, 279, 282, 285]

M. Tahir, M. A. Hussain, G. M. Cordeiro, G. Hamedani, M. Mansoor, and M. Zubair. The Gumbel-Lomax distribution: Properties and applications. *Journal of Statistical Theory and Applications*, 15(1): 61–79, 2016. [p278]

M. Teimouri and S. Nadarajah. Bias corrected MLEs for the Weibull distribution based on records. *Statistical Methodology*, 13:12–24, 2013. [p268]

M. Teimouri and S. Nadarajah. Bias corrected MLEs under progressive type-II censoring scheme. *Journal of Statistical Computation and Simulation*, 86(14):2714–2726, 2016. [p268]

B. X. Wang, X. K. Wang, and K. Yu. Inference on the Kumaraswamy distribution. *Communications in Statistics - Theory and Methods*, 46(5):2079–2090, 2017. [p281]

M. Wang and W. Wang. Bias-corrected maximum likelihood estimation of the parameters of the weighted Lindley distribution. *Communications in Statistics - Theory and Methods*, 46(1):530–545, 2017. [p268, 278]

W. Wang. *Bias-Corrected Maximum Likelihood Estimation of the Parameters of the Weighted Lindley Distribution*. PhD thesis, Michigan Technological University, 2015. [p272]

Wolfram Research, Inc. *Mathematica, Version 8.0*. Wolfram Research, Inc., Vienna, Champaign, Illinois, 2010. [p268, 285]

L. Xiao and D. E. Giles. Bias reduction for the maximum likelihood estimator of the parameters of the generalized Rayleigh family of distributions. *Communications in Statistics - Theory and Methods*, 43(8): 1778–1792, 2014. [p268, 274, 275, 278]

G. Zhang and R. Liu. Bias-corrected estimators of scalar skew normal. *Communications in Statistics - Simulation and Computation*, 46(2):831–839, 2015. [p268]

Josmar Mazucheli
Department of Statistics
Universidade Estadual de Maringá
Maringá, Brazil
jmazucheli@gmail.com

André Felipe Berdusco Menezes
Department of Statistics
Universidade Estadual de Maringá
Maringá, Brazil
andrefelipemaringa@gmail.com

Saralees Nadarajah
School of Mathematics
University of Manchester
Manchester M13 9PL, United Kingdom
mbbsssn2@manchester.ac.uk

afmToolkit: an R Package for Automated AFM Force-Distance Curves Analysis

by Rafael Benítez, Vicente J. Bolós and José-Luis Toca-Herrera

Abstract Atomic force microscopy (AFM) is widely used to measure molecular and colloidal interactions as well as mechanical properties of biomaterials. In this paper the **afmToolkit** R package is introduced. This package allows the user to automatically batch process AFM force-distance and force-time curves. **afmToolkit** capabilities range from importing ASCII files and preprocessing the curves (contact point detection, baseline correction…) for finding relevant physical information, such as Young's modulus, adhesion energies and exponential decay for force relaxation and creep experiments. This package also contains plotting, summary and feature extraction functions. The package also comes with several data sets so the user can test the aforementioned features with ease. The package **afmToolkit** eases the basic processing of large amount of AFM F-d/t curves at once. It is also flexible enough to easily incorporate new functions as they are needed and can be seen as a programming infrastructure for further algorithm development.

Introduction

In the last thirty years, atomic force microscopy (AFM) has become a necessary surface analytical tool for life and materials scientists (Müller and Dufrêne, 2008; Kainz et al., 2014). AFM offers the possibility to investigate molecular topographies and dynamical processes at (sub) nanometer scale as a function of time (Hansma et al., 1996; Ortega-Vinuesa et al., 1998; Kuznetsov et al., 2010; Lopez et al., 2010).

In addition, the AFM is a mechanical machine being able to measure forces between molecules, particles or surfaces (Hinterdorfer et al., 1996; Butt et al., 2005; Borkovec et al., 2012). It can also be used to measure mechanical properties of biomaterials, either indenting or stretching them (Rief, 1997; Marszalek et al., 1999; Alcaraz et al., 2003; Best et al., 2003; Kasas and Dietler, 2008; Garcia-Manyes and Sanz, 2010; Benitez and Toca-Herrera, 2014; Melzak and Toca-Herrera, 2015).

One problem that the researcher faces while performing force-distance (F-d) curves is the handling and the interpretation of the amount of data. For example, for determining the unbinding force between two molecules, hundreds of curves might be needed. Therefore, adaptive and flexible software routines are necessary to export, analyze and organize the measured data before starting the physical data interpretation. Several authors have addressed this type of problem before, proposing algorithms to F-d curve signals (Benítez et al., 2013; Kasas et al., 2000; Andreopoulos and Labudde, 2011; Crick and Yin, 2007; Lin et al., 2007a). However, these works focused mostly on contact point detection, adhesion energy quantification, or unfolding events. Here we present a set of functions bundled in a package of R statistical software that offer a compact analysis of the whole F-d curve.

Force – distance curve parts

In force-distance experiments, an AFM-tip or a colloidal probe (see Ducker et al. (1991)) is extended towards and retracted from the sample at speeds that may vary between a few nm/s and dozens of μm/s. While performing the F-d experiment, the deflection of the cantilever is quantified as a function of the displacement of the piezo-scanner (Moreno-Flores and Toca-Herrera, 2013). Thus, the force sensed by the cantilever is calculated by multiplying its deflection by its spring constant (Hooke's law), which must be evaluated in every experiment. A general F-d curve (or force-time curve) can be divided in three parts: move towards the sample, contact with the sample and separation from the sample. Every part of the curve contains valuable and different information about the sample of study. The first one delivers information about repulsive, attractive or structural forces between the tip/colloidal probe and the sample (e.g. electrostatic, van der Waals, hydration, entropic, etc.). The second part of the curve corresponds to the so-called contact regime, that is, when the cantilever is touching (compressing) the sample. This part of the curve provides information about the mechanical properties of the sample (i.e. Young's modulus, relaxation time and viscosity of cells or hydrogels). Finally, the separation curve contains information about adhesion or rupture forces (i.e. ligand-receptor interactions), the existence of tethers, and possible molecular unfolding events (i.e. mechanical unfolding of polymers).

However, the operating way of the piezo-electric does not follow this segmentation. The piezo-electric distinguishes three movements: approach, pause and retract (see Figure 1). In this way, the F-d raw data that can be acquired from the AFM is structured in these three parts. Indeed, it does not

know where the contact point is located (zero distance between tip and sample).

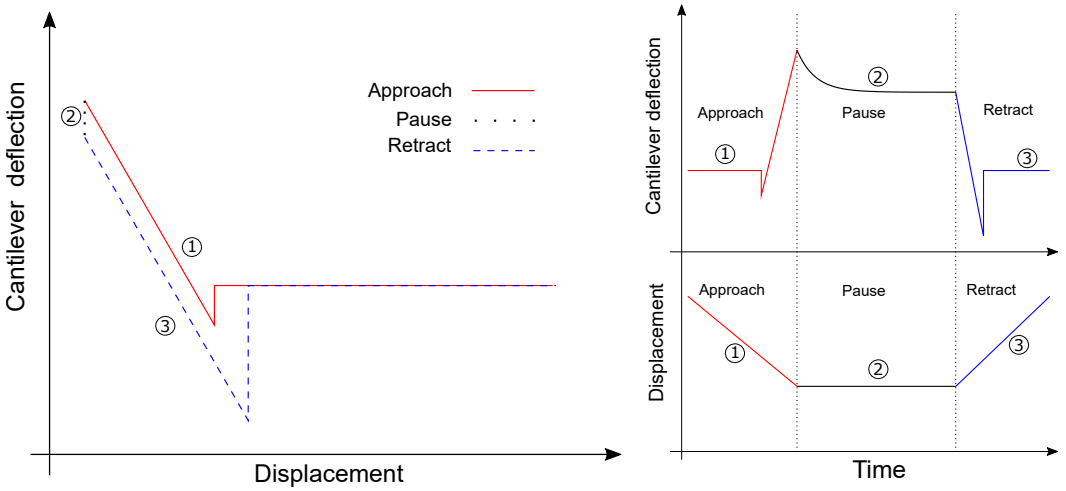


Figure 1: F-d curve. Left: Schematic representation of a typical Force-distance curve. Right (above and below): Schematic representation of a typical Force-time at constant height.

Therefore, it is necessary to extract all above mentioned physically relevant information from the F-d raw data as it is given by the device.

The afmToolkit package

In this article we introduce the **afmToolkit** package whose aim is to automate certain operations and calculations that are normally done routinely on the F-d curves.

Package **afmToolkit** is available in CRAN and can be installed via the command 'install.package'. Nevertheless, the development version of the package of **afmToolkit** is also stored in the github platform (github.com) and it can be installed directly from the R console using the 'install_github' function from package **devtools** (Wickham and Chang, 2016).

```
> install.packages("devtools") % chktex 8
> library("devtools")  % chktex 8
> install_github("rbensua/afmToolkit")
> library(afmToolkit)
```

The package depends on the **ggplot2** package (Wickham, 2016a) and uses functions from the non-standard packages **minpack.lm**, (Elzhov et al., 2015), **gridExtra** (Auguie, 2016), **scales** (Wickham, 2016b) and **dplyr** (Wickham and Francois, 2016).

The "afmdata" class

The basic data structure for AFM F-d curves analysis with the **afmToolkit** package is the "afmdata" class. It is an S3 class consisting on a list having at least the fields data and params (see Figure 2).

The field data is a data frame containing the data itself. The columns are Z, for the distance, Force and segment, being the later a factor with levels approach, pause and/or retract, denoting which part of the force – distance curve each data belongs to. In some cases also a Time column could be present.

Eventually the number of columns of the data data frame may be increased, as different analysis are performed. For example, once the baseline correction is done, a new column ForceCorrected is added.

The params field is a list containing different parameters gathering information about the experiment (e.g. the cantilever spring constant, the ID of the curve, etc.).

As further analyses are performed, the results are added as new fields to the "afmdata" list. Therefore, the "afmdata" list will eventually contain, in a single data structure, the whole F-d curve relevant information.

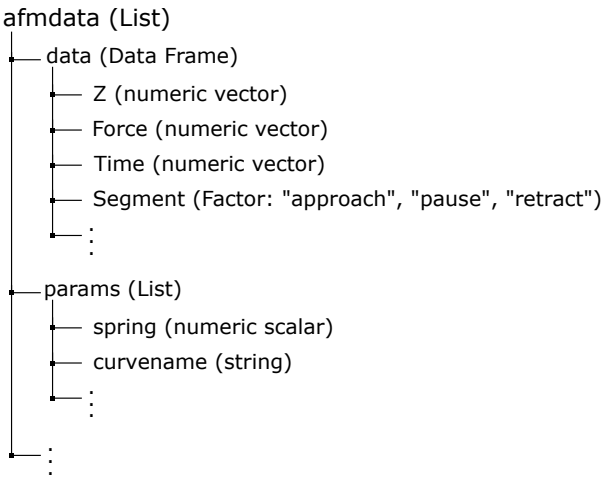


Figure 2: `"afmdata"` class. Basic `"afmdata"` class list description.

Importing data

Although the `"afmdata"` class definition function is flexible enough to create an `"afmdata"` structure from a data frame directly, it is usually more convenient, in order to speed up the work flow, to import the data from a file obtained directly from the AFM device used.

At the moment, there are only available two functions for importing F-d curves: `afmReadJPK` and `afmReadVeeco`, which import NanoWizard JPK and Veeco (Bruker) data files, respectively, provided they had previously been exported as ASCII files.

Importing data from JPK^{TM} ASCII files: In the first case, a full AFM experiment is stored in a single text file in which the different segments of the experiment (approach, contact or pause, and retract) are separated by a header (see Figure 3-Left).

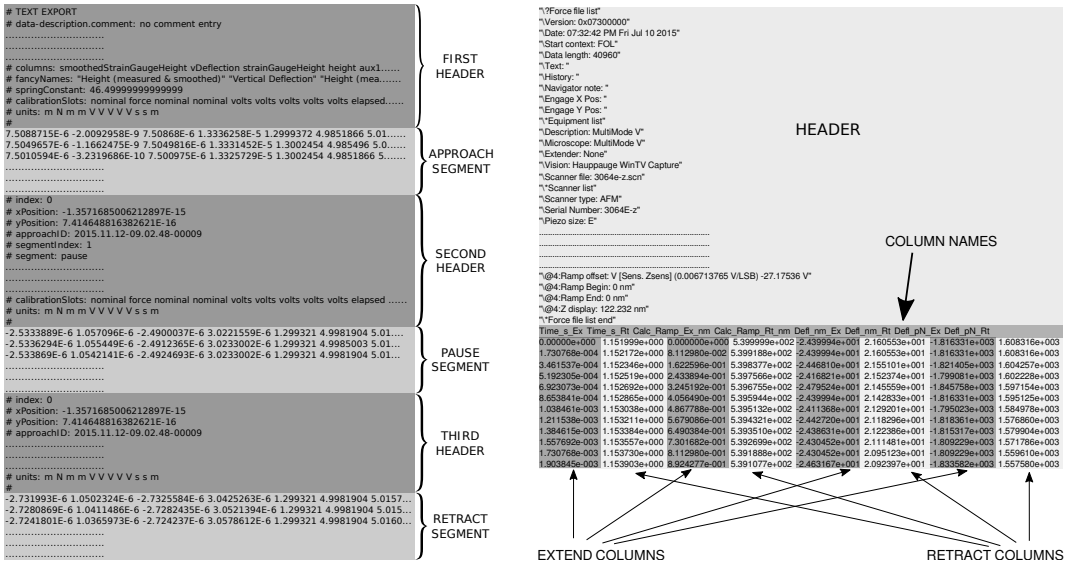


Figure 3: ASCII Files. Left: structure of a JPK ASCII file with three headers. Right: Structure of a Veeco (Bruker) ASCII file.

The function finds out how many headers the file contains (up to three headers), which columns contain the relevant information: distance, force and time (if available), and it also looks for the spring constant in the header and stores it in the `params` field. The ID (curvename) of the experiment will be the name of the ASCII file (by default).

The **afmToolkit** has two JPK text example files: 'force-save-JPK-2h.txt' and 'force-save-JPK-3h.txt', with two and three segments, respectively. Let's see an example.

```
> data <- afmReadJPK("force-save-JPK-2h.txt.gz",
                     path = path.package("afmToolkit"))
JPK file force-save-JPK-2h.txt loaded. 2 headers found.
```

Once it is loaded we can check the structure of the new "afmdata" variable created.

```
> str(data)

List of 2
 $ data  :'data.frame':        1227 obs. of  4 variables:
  ..$ Z      : num [1:1227] 6.48e-06 6.48e-06 6.48e-06 6.48e-06 ...
  ..$ Force  : num [1:1227] -1.71e-08 -1.70e-08 -1.71e-08 -1.71 ...
  ..$ Time   : num [1:1227] 0.000244 0.000732 0.001221 0.001709 ...
  ..$ Segment: Factor w/ 2 levels "approach","retract": 1 1 1 1 ...
 $ params:List of 2
  ..$ SpringConstant: num 0.16
  ..$ curvename     : chr "force-save-JPK-2h.txt"
 - attr(*, "class")= chr "afmdata"
```

From the output we can see that there are two fields in the "afmdata" structure: the data field, and the params field.

We can easily plot the whole experiment using the available plotting S3-method plot.afmdata, which makes use of the **ggplot2** package.

```
> plot(data)
```

The result is depicted in Figure 4.

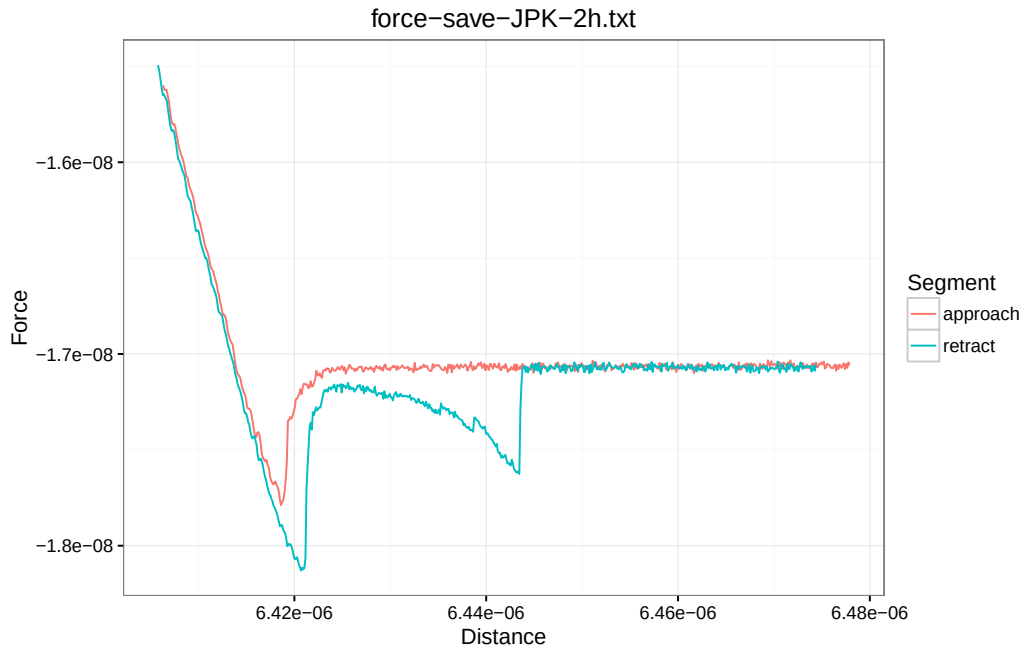


Figure 4: plot.afmdata function. Two segment JPK F-d curve plotted with the plot.afmdata function. JPK refers to the type of AFM. In this case the cantilever moves and the sample is at rest. For the Multimode (Bruker) the cantilever is at rest and the sample moves.

Note that the afmReadJPK function automatically determines the number of segments from the number of headers present in the ASCII file. However it does not find out from the headers which segment is each one of them. That is, if afmReadJPK finds only one header, it will assume that the file contains only an approach segment. If two headers are detected, it will be assumed that the first one is the approach and the second is the retract segment. Finally, if three headers are found, they will be assigned to the approach, pause and retract, respectively. At this moment no more than three segments per file are supported.

Importing data from Veeco™ ASCII files: A Veeco ASCII file is, in comparison, simpler than the JPK file. It contains a single header with the details of the parameters of the AFM experiment and the data from each part of the experiment (approach or extend, and retract) are stored on separated columns (see Figure 3-right).

The `afmReadVeeco` function reads the data file and creates an `"afmdata"` structure separating the different parts of the experiment obtained from their corresponding columns in the file.

The syntax for importing a Veeco file is very similar to the one of the JPK file.

```
> dataVeeco <- afmReadVeeco("veeco_file.txt",
                            path = path.package("afmToolkit"))
Veeco file veeco_file.txt loaded.
```

Contact point and detach point determination

Contact point: The first, and probably the most important step in the AFM F-d curve analysis is the determination of the contact point. We define such point as the location in the approach segment of the F-d curve where the deflection of the cantilever is, for the first time, significantly higher than the baseline average slope. It should be noted that we are using here the term "contact point" in a broad sense, since there is no necessity for a real contact between the tip and the sample to take place, but we are rather considering the contact point as the point at which the interactions between the sample and the tip start to appear. Such interactions could be caused by an actual contact between tip and sample, or they could be the response to a repulsive force or even a "jump to contact" attractive interaction.

Lots of different approaches to the determination of the contact point have already been made (Lin et al., 2007a,b; Rudoy et al., 2010; Benítez et al., 2013; Gavara, 2016). Function `afmContactPoint` estimates the contact point using the algorithm described in Benítez et al. (2013). This method computes, from the F-d signal, a new δ signal which is, roughly speaking, the lagged difference between two values of the slopes of the best lines fitted by a local linear regression on a rolling window of some predetermined width. High absolute values of δ are related to abrupt changes in either the original F-d curve or its slope. From the δ signal, the contact point is obtained using two given thresholds which are multiples of the δ signal noise (i.e. standard deviation) in the first part of the curve (i.e. non-contact part). See Benítez et al. (2013) for specific details on the algorithm.

For example, for the two segments example shown above, we could take the following parameters:

- width: Width of the window, given in number of points, in which the local regression is performed. We shall set `width = 20`.
- mul1: Value of the first multiplier used to determine the first threshold. It should be small enough to detect the contact point accurately (even zero is a possible value). In this case we will set `mul1 = 1`.
- mul2: Value of the second multiplier used to determine the second threshold. Its value should be large enough to distinguish the contact point from the regular noise of the signal but no so large that the contact point remains undetected. We will set this value as `mul2 = 10`.

The rest of the parameter will remain in their default values. Then the following command will detect the contact point and plot it together with the approach segment of the F-d curve.

```
> width <- 20
> mul1 <- 1
> mul2 <- 10
> data <- afmContactPoint(data, width, mul1, mul2)
> plot(data, segment = "approach") +
  geom_vline(xintercept = data$CP$CP, lty = 2)
```

Figure 5-Left shows the approach segment of the curve together with the contact point estimation (dashed vertical line). Note that the contact point is the first point in the approach curve (from right to left) in which the curve starts to deviate from the baseline. When there is an attraction force (like in the example) this contact point is overestimated and, in order to be sure that we are starting the contact regime in which the tip is indeed contacting the sample, we should calculate the *zero force point* (see below).

Detach point: The detach point is the point in the retract segment where the tip finally leaves the sample. It is computed exactly in the same way that the contact point, but the computations are performed to the retract part and the curve is traced backwards. The input parameters of function `afmDetachPoint` are the same than for function `afmContactPoint`. Let us continue with our example:

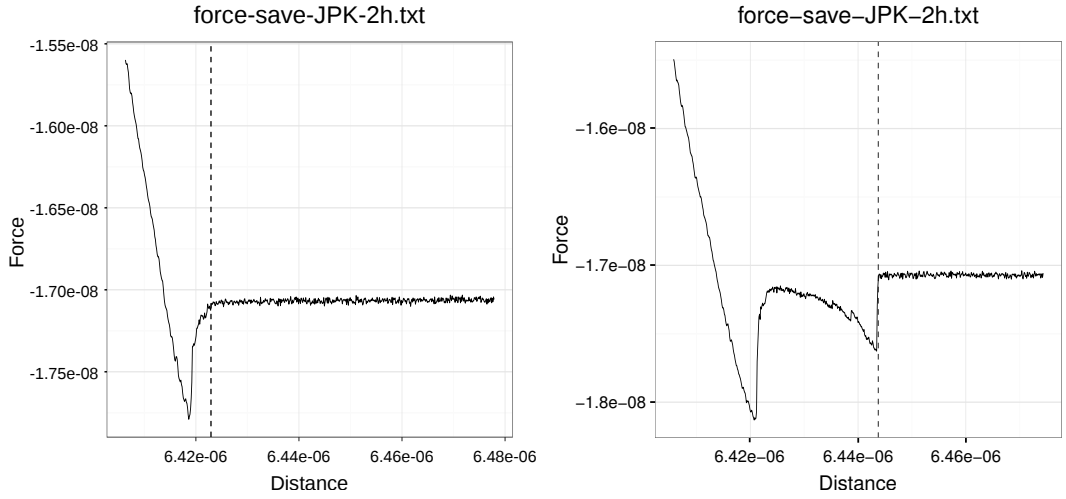


Figure 5: Contact point/Detach point detection. Left: F-d approach curve (solid line) and the estimation of the contact point (dashed line) with the `afmContactPoint` function. Right: Detach point (dashed line) estimated on the retract segment (solid line) with `afmDetachPoint`.

```
> data <- afmDetachPoint(data, width = 20, mul1 = 1, mul2 = 10)
> plot(data, segment = "retract") +
    geom_vline(xintercept = data$DP$DP, lty = 2)
```

The estimation of the detach point can be seen in Figure 5-Right.

Baseline correction

Once the contact and detach points are found, the baseline calibration can be carried out. Theoretically, when the tip is far from the sample, the deflection of the cantilever and therefore the measured force should be zero. Nevertheless, in most cases there is an offset, or even a drift that keeps this part of the curve away from the zero value. In order to fix this behaviour, a baseline correction is done. Such correction is usually done manually, by selecting the part of the curve which we know to be away from the sample and then substract to the whole F-d curve, the least squares fitted line to such selected segment of the curve.

Since we already determined the contact and detach points, we know exactly when the tip is away from the sample. Function `afmBaselineCorrection` will perform this calibration automatically. This function will add a new column called `ForceCorrected` to the `data` data frame field of the `"afmdata"` class.

```
> data <- afmBaselineCorrection(data)
> plot(data)
```

Once the baseline correction is done, futher analyses will only use the Corrected Force column of the `data` field. For example, the '`plot(data)`' command will plot the `"afmdata"` F-d curve with the calibration already done, as can be observed in Figure 2 where it can be seen that when the tip is away from the sample, the force is actually zero.

Zero force point

As we mentioned above, function `afmContactPoint` finds the first point in the approach segment for which the slope of the curve is significantly different from the baseline. This usually coincides with what would be considered the contact point by "eye inspection". Nevertheless, when there is an attraction force prior to the contact regime, the point found with function `afmContactPoint` is an overestimation of the real contact point since it does not distinguish between attraction and repulsions. Therefore, a new function, `afmZeroPointSlope`, can be used to find the point after the minimum which is the intersection between the curve and the baseline (zero force line).

With the following commands we can obtain the zero force point and the slope of the F-d curve after that point. The results will be added to the `data` structure in new field named `Slope`. We finally plot the curve and both in Figure 7, the contact point (dashed red line) and the zero force point (blue dotted line).

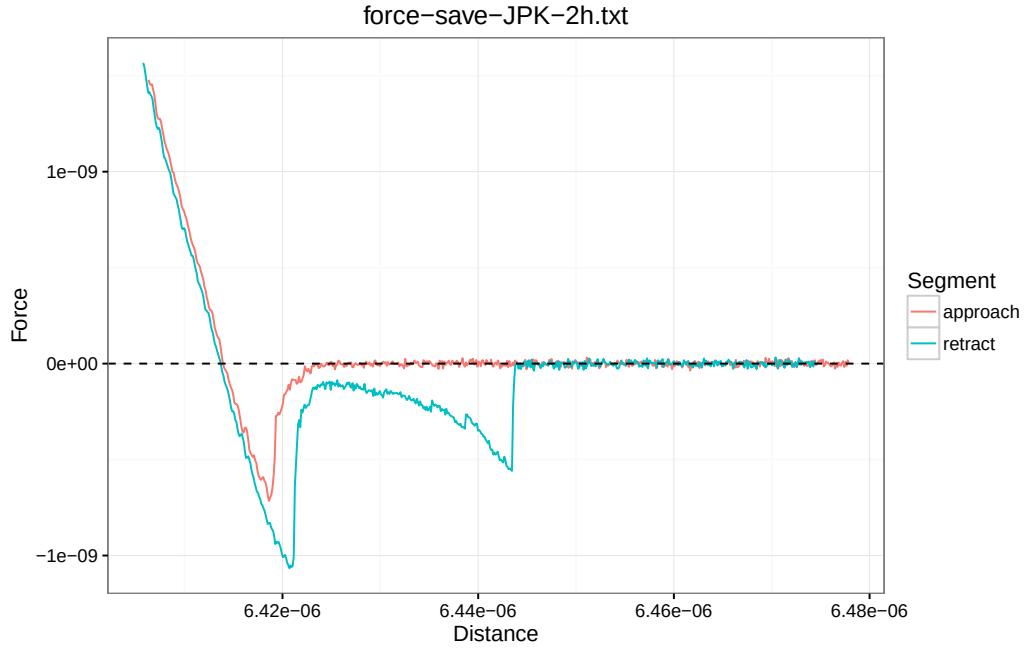


Figure 6: Base line correction. F-d curve with the baseline correction done with `afmBaselineCorrection` function.

```
> data <- afmZeroPointSlope(data, segment = "approach")
> plot(data, segment = "approach") +
    geom_vline(xintercept = data$CP$CP, col = "red", lty = 2) +
    geom_vline(xintercept = data$Slope$Z0Point, col = "blue", lty = 3)
```

Young's modulus estimation

One of the most important parameters for determining the mechanical properties of a sample is the Young's modulus. Obtaining the Young's modulus from an AFM F-d curve is not straightforward and depends on several factors. Namely, the spring constant of the cantilever, the contact area, which largely depends on the tip's geometry and the Poisson ratio, which depends on the compressibility of the sample. The typical AFM tip geometries are: spherical (colloidal probes), pyramidal and conical.

Function `afmYoungModulus` computes the Young's modulus of the sample from the approach segment fo the force-distance curve. Before it is called, be aware that the spring constant should be available in the `params` field of the `"afmdata"` structure and both, the baseline correction and the zero force point should have been obtained.

Currently, only the two most used geometries are available for this function: the four-sided pyramidal tip and the paraboloid tip. The former uses the classical Snedon formulae:

$$F = \frac{E}{1-\nu^2}\frac{\tan\alpha}{\sqrt{2}}\delta^2, \tag{1}$$

being E the Young's modulus, ν the Poisson ratio, α the pyramid face angle and δ the indentation of the tip into the sample. Parameters ν and α should be provided ($\nu = 0.5$ is the default value) and first we will need to determine the tip's indentation.

The latter uses the Hertz model given by

$$F = \frac{4\sqrt{R}}{3}\frac{E}{1-\nu^2}\delta^{3/2}. \tag{2}$$

The indentation can be obtained substracting to the piezo displacement, Z, the zero force point, Z_0 and the deflection of the cantilever,

$$\delta = Z - Z_0 - \frac{F}{\kappa},$$

where κ is the cantilever's spring constant. Function `afmIndentation` computes the indentation and adds it as a new column to the `data` field of the `"afmdata"` structure. Once the indentation is calculated,

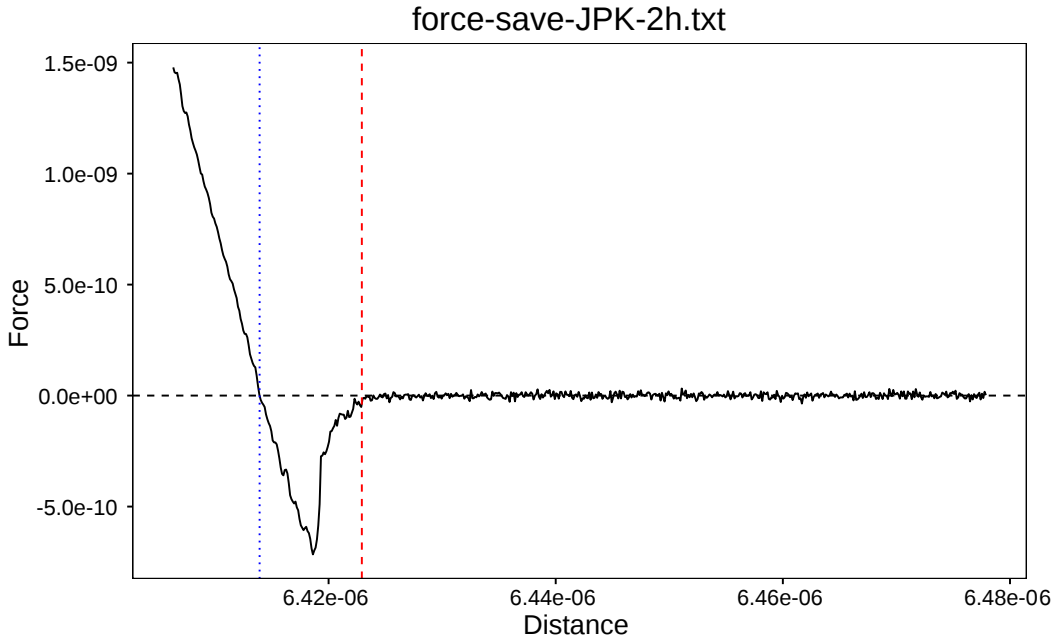


Figure 7: Zero-force point. Contact point (red dashed line) and zero force point (blue dotted line) obtained with `afmContactPoint` and `afmZeroPointSlope`, respectively.

function `afmYoungModulus` computes the Young's modulus by fitting a straight line to F vs δ^2. From the slope of the line fitted and (1), E can be obtained and it is added to the `params` field of the `"afmdata"` variable.

For the example data, using a pyramidal tip with $\alpha = 22$ deg considering an incompressible sample ($\nu = 0.5$), we have

```
> data <- afmIndentation(data) # First compute the indentation
> data <- afmYoungModulus(data, thickness = , 5e-9,
                          params = list(alpha = 22))
> data$YoungModulus$YoungModulus
[1] 59730377
```

We have, therefore, obtained a Young's modulus of $E = 59.73$ MPa.

Exponential decay fit

Another important type of experiment used to obtain viscoelastic mechanical properties of the sample is the Force relaxation – Creep experiment. In these experiments, after reaching the sample, the tip remains in contact for a predetermined elapsed time. In the Force relaxation experiment, the height of the AFM's piezo is held constant, and in presence of a viscoelastic material, an exponential decay in the force should be observed. On the other hand, in a Creep experiment, the force remains constant and, as a consequence, it is the height of the tip what shows an exponential decay behaviour.

The presence of the exponential decays in force and/or in height, can be explained in terms of the classical linear viscoelastic theory, where combinations of Maxwell and Voight elements (springs and dahspots) are used. However they will not be discussed here and we refer the reader to Riande et al. (1999) for a general discussion of linear viscoleasticity and Moreno-Flores et al. (2010a,b) for more specific models used in nanoindentation AFM experiments in cell mechanics problems.

Plainly speaking, each different viscoelastic material in the sample is characterized by a different relaxation time. Thus, the response Force vs Time or Z vs Time could be represented by a Prony series of the form (following the notation of Moreno-Flores et al. (2010a)):

$$F(t) = a_0 + \sum_{k=1}^{n} a_k e^{t/\tau_k},$$

for the Force relaxation experiment, and

$$Z(t) = c_0 + \sum_{k=1}^{n} c_k e^{x_k t},$$

for the creep experiment.

The afmToolkit can determine the parameters of the above mentioned Prony series for both types of experiments by fitting the sum of exponential to the data via a nonlinear least squares Levenberg-Mardquart algorithm provided by the **minpack.lm** package (Elzhov et al., 2015). Presently, only either one or two exponentials in the Prony series can be considered. This can be explained because in typical AFM experiments, there is usually either an homogenenous material or, for cell mechanics problems, at least two materials – cell membrane and cell cytoskeleton – are considered.

An important issue when performing nonlinear least square fits is the election of the initial values for the parameters. Often there is an extremely high sensitivity to such values, so it is critical to make good initial guesses. For a general single exponential decay function $y(t) = a_0 + a_1 \exp(-t/\tau_1)$, we find that a_0 is the horizontal asymptote, $a_0 + a_1$ is the value at $t = 0$ and τ_1 should be of the same order of magnitude of the total time. In case there is a two-exponential decay function $y(t) = a_0 + a_1 \exp(-t/\tau_1) + a_2 \exp(-t/\tau_2)$, a_0 is again the horizontal asymptote, but now $a_0 + a_1 + a_2$ is the initial value $y(0)$, so a good initial guess could be setting both parameters a_1 and a_2 with the same values (i.e. $a_1 = a_2 = (y(0) - a_0)/2$). For the initial values of the decay times (or frequencies), a usually good guess is to set one of them, say τ_1, of the same order of magnitude as the total time, and then set the second one an order of magnitude smaller (i.e. $\tau_2 = \tau_1/10$).

Let us see an example. We will need a data file with three segments: approach, contact and retract.

```
> data <- afmReadJPK("force-save-JPK-3h.txt",
                     path = path.package("afmToolkit"))
JPK file force-save-JPK-3h.txt loaded. 3 headers found.
```

Once the data is loaded, we will proceed with the contact and detach point determination, baseline correction and the zero force point estimation.

```
> data <- afmContactPoint(data, width, mul1, mul2)
> data <- afmDetachPoint(data, width , mul1, mul2)
> data <- afmBaselineCorrection(data)
> data <- afmZeroPointSlope(data, segment = "approach")
```

We may now plot the Force vs Time curve in the contact segment:

```
> plot(data, segment = "pause" , vs = "Time")
```

In Figure 8 it is shown the Force vs. Time curve together with the values of the magnitudes that will be used to make the initial guesses for the parameters.

Taking into account the values depicted in Figure 8, we shall see two options for the starting values of the fit parameters, one for the single exponential fit and other for the two exponentials fit.

```
> data1 <- afmExpDecay(data, nexp = 1, tmax = 7.5,type = "CH",
                       start = c(a0 = 8.2e-7, a1 = 3.35e-7, tau1 = 5))
> data <- afmExpDecay(data, nexp = 2, tmax = 7.5,type = "CH",
                       start = c(a0 = 8.2e-7, a1 = 1.675e-7,
                       a2 = 1.675e-7, tau1 = 5, tau2 = 0.1))
```

The output of this function is another afmdata class variable with an extra `Expfit` field, which is a list with two fields: `expdecayModel` and `expdecayFit`, being the former the `nls` class structure resulting from the `nlsLM` function from package **minpack.lm**, and the latter a numerical vector containing the exponentially decaying forces estimated by the fit.

Therefore, the standard `summary` function can be used to extract the relevant information of the fits:

```
> summary(data1$ExpFit$expdecayModel)
> summary(data$ExpFit$expdecayModel)
```

The fits are shown in Figure 9. It is seen that in this case, the double exponential clearly beats the single exponential fit, but in order to be sure we can check the goodness of fit data. From the fit summaries shown in Table 1 we may assert that the double exponential fit performs a better prediction than the single exponential fit – the Residual standard error is an order of magnitude smaller – while keeping all the parameters statistically significant, i.e. the standard error of all coefficient are at least one order of magnitude smaller than the value estimated.

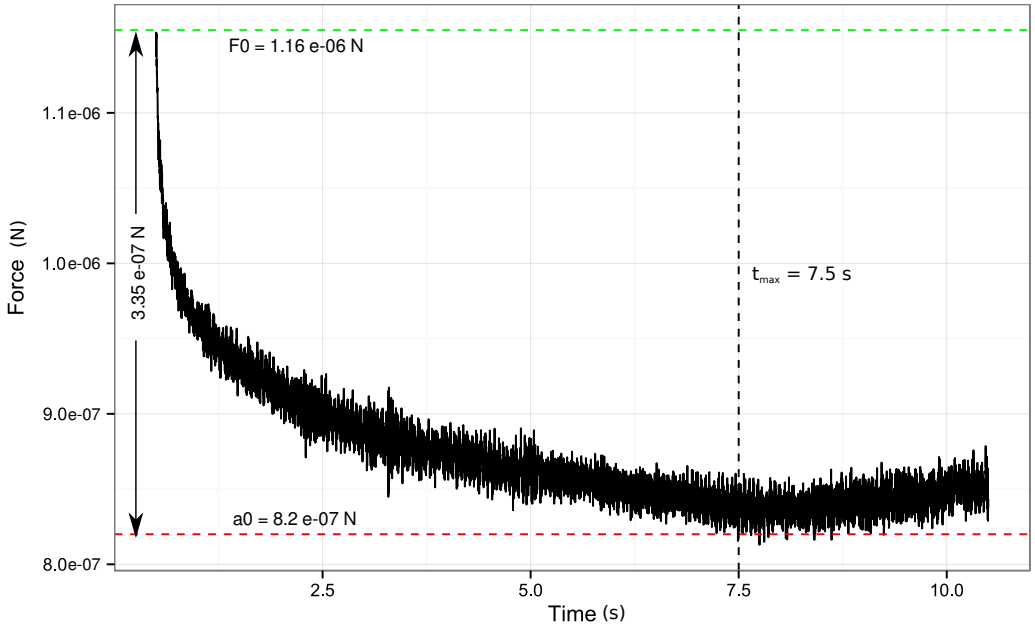


Figure 8: Force exponential decay. Force vs. Time plot in the contact segment. A clear exponential decay is observed. The important magnitudes are indicated in the plot.

Adhesion energy

When the cantilever retracts from the sample several events can take place, depending on the type of experiment. One of the most important effects that may occur is the adhesion phenomena. In a force spectroscopy experiment, the adhesion event is usually reflected in the F-d curve as an hysteresis loop, in which, as a result of the presence of non-conservative forces, the retract curve is below the approach curve.

The adhesion energy can be estimated as the area between the retract force-distance curve and the Z-axis from the zero-force point – in which it is considered that the tip starts to detach from the sample – to the point at which there is a jump-from-contact event.

Sometimes, after this jump-from-contact event, some other important phenomena, like tether formation, can take place before the tip fully detaches from the sample and the F-d curve finally enters in the off-contact region.

We can compute these energies by means of the `afmAdhesionEnergy` function. This function uses an algorithm similar to the contact point estimation method implemented in the `afmContactPoint` and `afmDetachPoint` functions. It has as inputs the `width` of the rolling window in which a best line fit is computed and a multiplier `mul` that will be used to determine the jumps in the F-d curve that determine the different adhesion events.

Following the example of the three segment F-d curve, we could easily find the adhesion energies with the commands

```
> data <- afmAdhesionEnergy(data, width = 10, mul = 15)
> data$AdhEner$Points
[1]  68.0 124.5 347.0
> data$AdhEner$Energies
         E1adh        E2adh       Etotal
1 1.142784e-14 1.171835e-15 1.259942e-14
```

The `afmAdhesionEnergy` function appends to the afmdata class `data` input another field named `AdhEner`, which is a list with two fields: `Points` and `Energies`. The `Points` field is a vector of length 3 containing the indices of the F-d curve where the three events take place: the zero-force point (left end), the jump-from contact event (middle) and the full-detach event point (right end). The `Energies` field is also a vector of three components containing the adhesion energy, `E1adh`, computed from the zero-force point to the jump-from-contact point, the remaining energy, `E2adh`, computed from the jump-from-contact point to the full-detach event point, and the total energy `Etotal` which is the sum of these two. Therefore, in this example, the total adhesion energy is around $1.26\,10^{-14}$ J.

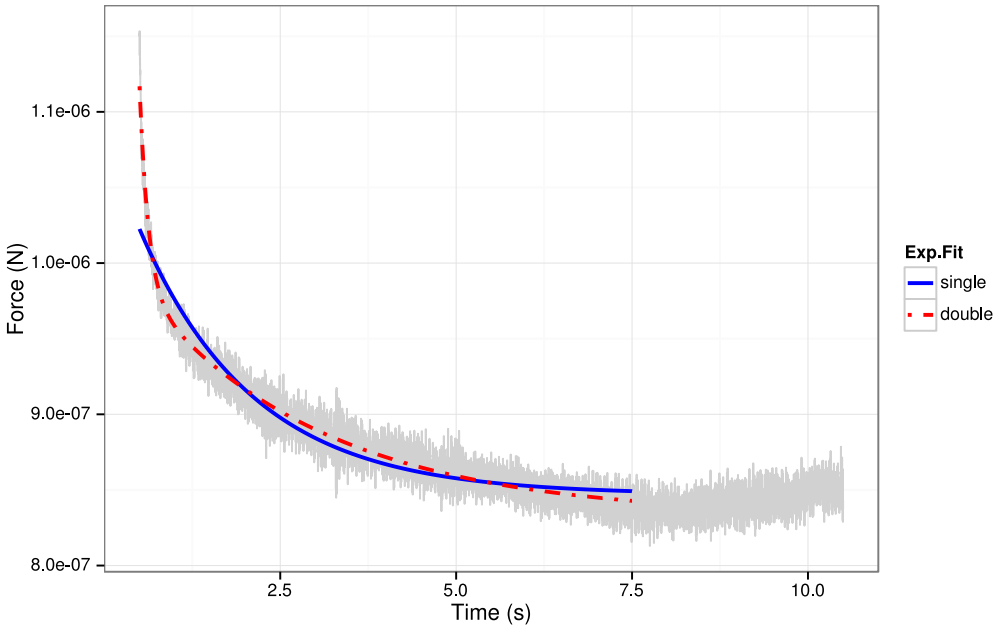


Figure 9: Exponential decay fits. Single and double exponential fits.

Summarizing an "afmdata" class

After all these analyses have been done to an F-d curve stored in an "afmdata" class, function summary can display the most relevant information about the curve in both, numerical and visual ways.

In order to illustrate a full example, we will first compute the Young's modulus of the three segments F-d curve example.

```
> data <- afmIndentation(data)
> data <- afmYoungModulus(data, thickness = 5e-8,
                          params = list(alpha = 22))
```

Now that all analyses are performed, let us see how the summary function shows us all relevant information.

```
> summary(data)
         Estimate   Std. Error    t value Pr(>|t|)
a0   8.295849e-07 3.445595e-10 2407.66801        0
a1   1.497522e-07 3.367846e-10  444.65288        0
a2   1.358410e-07 1.012757e-09  134.12989        0
tau1 2.779792e+00 2.169182e-02  128.14933        0
tau2 1.334409e-01 1.836281e-03   72.66913        0
  # of segments Spring Constant Contact Point Young's Modulus
1            3         52.9146  6.982701e-06       249153289
Zero force pt. Type of Experiment
   6.99318e-06    Constant Height
```

The graphical information provided is depicted in Figure 10. Thus, from the plots one can easily know the number of segments of the F-d curve, the goodness of the exponential decay fit, the value of the Young's modulus obtained with an Hertz's contact model and a pyramidal tip, among other useful information.

A sample R afmToolkit session for batch processing

Up to now, we have shown the capabilities of the **afmToolkit** package for dealing with one F-d curve. However, the usual workflow in AFM force spectroscopy experiments is to repeat the measures a number of times that can be very large (even hundreds of repetitions). Therefore it is absolutely necessary algorithms and methods allowing us to batch-process all the curves at once.

Table 1: Comparison of the single and double exponential fits.

Single exponential				
	Estimate	Std. Error	t value	Pr (>|t|)
a_0	8.5e-07	2.3e-10	3.6e+03	0
a_1	1.8e-07	4.3e-10	4.1e+02	0
τ_1	1.6	0.0095	1.7e+02	0
Residual standard error: 1.232e-08 on 14333 d.o.f.				

Double exponential				
	Estimate	Std. Error	t value	Pr (>|t|)
a_0	8.3e-07	3.6e-10	2.3e+03	0
a_1	1.5e-07	3.5e-10	4.3e+02	0
a_2	1.4e-07	1e-09	1.3e+02	0
τ_1	2.6	0.021	1.2e+02	0
τ_2	0.13	0.0018	71	0
Residual standard error: 8.196e-09 on 14331 d.o.f.				

The **afmToolokit** package can deal with a set of F-d curves stored as `"afmdata"` class variables and bundled together in one special data class called `"afmexperiment"`.

An `"afmexperiment"` data class is a list of `"afmdata"` variables. Almost every function in the **afmToolkit** package first checks for the input data class. If the input is of `"afmdata"` class, it performs the analysis for one curve, but if it is an `"afmexperiment"` class variable, it loops for every `"afmdata"` curve in the list, executing the function to each individual F-d curve.

Next, we will show as an example, how would it be to deal with several F-d curves at once. First we will use the `afmReadJPKFolder` function in order to read all JPK files contained in some folder.

```
> dataFolder <- paste(path.package("afmToolkit"),"afmexperiment",sep = "/")
> data <- afmReadJPKFolder(dataFolder)
JPK file force-save-1.txt loaded. 2 headers found.
JPK file force-save-2.txt loaded. 2 headers found.
JPK file force-save-3.txt loaded. 2 headers found.
JPK file force-save-4.txt loaded. 2 headers found.
```

This function will create the `"afmexperiment"` data variable:

```
> class(data)
[1] "afmexperiment"
```

As an illustrative example we can make use of the `batchExperiment` data set available in **afmToolkit**. This data set consists on an `"afmexperiment"` data class containing 14 F-d curves in `"afmdata"` format.

```
> data(batchExperiment)
```

The 14 curves correspond to two different experiments: the first one is a sample covered with Polyallylamine hydrochloride ("PAH") (6 curves) while the second one is covered with Chitosan ("CHI") (8 curves). Such factor is specified in the `params` field of the `"afmdata"` structure. For example:

```
> str(batchExperiment[[1]]$params)
List of 3
 $ SpringConstant: num 0.102
 $ curvename     : chr "force-save-2016.07.27-16.41.34.558.txt"
 $ type          : chr "PAH"
```

Now, the standard analysis procedure would be:

1. **Preprocessing the curves**: Contact and detach points detection, Baseline correction, Zero force point determination and indentation calculus.

   ```
   > width <- 50
   > mul1 <- 1
   > mul2 <- 10
   ```

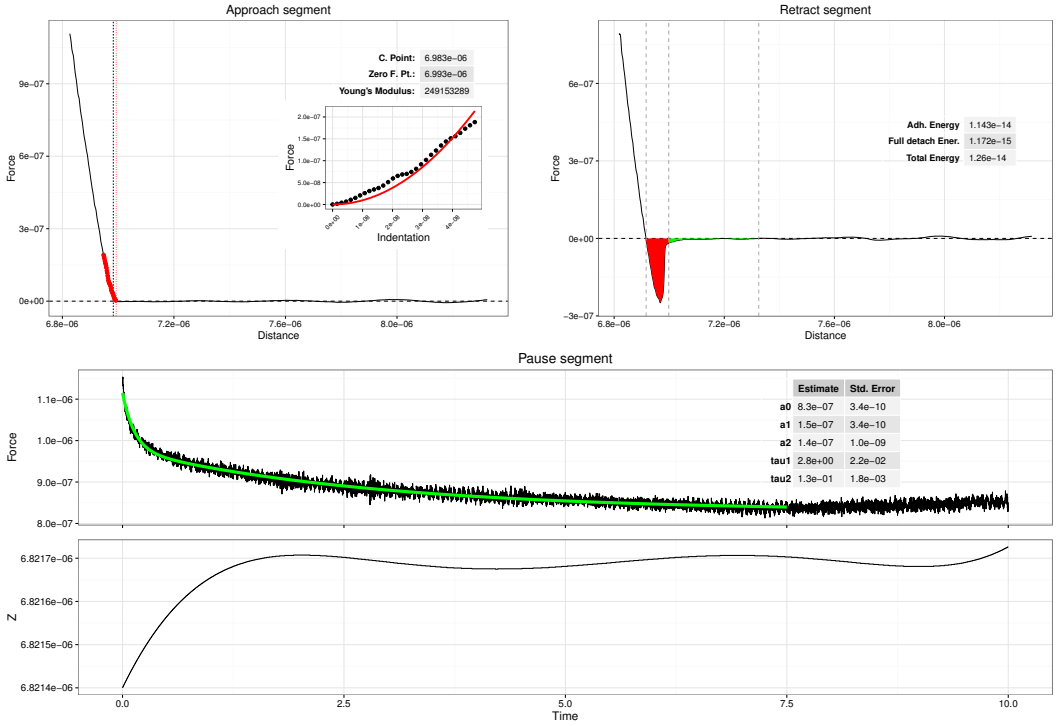


Figure 10: Summary plots. Information and plots given by the summary function.

```
> batchExperiment <- afmContactPoint(batchExperiment, width=width,
                                     mul1 = mul1, mul2 = mul2)
> batchExperiment <- afmDetachPoint(batchExperiment, width=width,
                                    mul1 = mul1, mul2 = mul2)
> batchExperiment <- afmBaselineCorrection(batchExperiment)
> batchExperiment <- afmZeroPointSlope(batchExperiment,
                                       segment = "approach")
> batchExperiment <- afmIndentation(batchExperiment)
```

2. **Curve analysis.**

```
> batchExperiment <- afmYoungModulus(batchExperiment,
                          thickness = 2.5e-7,
                          geometry = "paraboloid",
                          params = list(R = 1e-8))
> batchExperiment <- afmExpDecay(batchExperiment, nexp = 2,
                          type = "CH", plt = FALSE, tmax = 5)
> batchExperiment <- afmAdhesionEnergy(batchExperiment,
                          width = 5, mul = 15)
```

After some warning messages due to the afmExpDecay function telling us that we did not provide some initial values for the Levenberg-Mardquart algorithm, we will find that now our "afmexperiment" structure is a list of 14 "afmdata" class variables, each one with 8 fields:

```
> head(summary(batchExperiment))
                                    Length Class   Mode
force-save-2016.07.27-16.56.36.140.txt 8      afmdata list
force-save-2016.07.27-16.57.00.672.txt 8      afmdata list
force-save-2016.07.27-16.57.25.194.txt 8      afmdata list
...
```

3. **Extracting the results:** Once all the analyses are performed, we need to extract the information from the "afmexperiment" list and store it in spreadsheet-like data frames, since they are more suitable formats for further analysis or plotting.
   To that aim we use the afmExtract function.

```
>  parameters <- afmExtract(batchExperiment,
                            params = list("YM", "AE", "ED"),
                            opt.param = "type")
```

The parameter `params` is a list with the parameters we want to extract (`YM` stands for "Young's Modulus", `AE` for "Adhesion Energies" and `ED` for "Exponential decay"). The parameter `opt.param` is an optional parameter with additional information that we may want to store in our data frames (like factors describing the experiments). In this case we set '`opt.param = "type"`' so we include the information relative to the type of experiment ("PAH" or "CHI").

The result of the `afmExtract` function is a list with two data frames. The first one storing the Young's Modulus and the Adhesion Energies, and the second one containing the exponential decay results (coefficients and standard errors).

4. **Plotting the results.** Plotting the parameters is now straightforward, provided the **dplyr** package for data frame manipulation is installed and loaded:

```
> library(dplyr)
> parameters[[1]] %>% ggplot(aes(x = type, y = YM)) +
      geom_boxplot() + ylab("Young's Modulus (Pa)")

> parameters[[1]] %>% ggplot(aes(x = type, y = Etotal)) +
      geom_boxplot() + ylab("Total Adhesion Energy (J)")

> parameters[[2]] %>% ggplot(aes(x = type, y = Estimate)) +
      geom_boxplot() + facet_wrap(~parameter, scales = "free")
```

Young's Modulus results are depicted in Figure 11 (left plot), total Adhesion Energy results in Figure 11 (right plot), and the different parameter estimates for the exponential decays are shown in Figure 12.

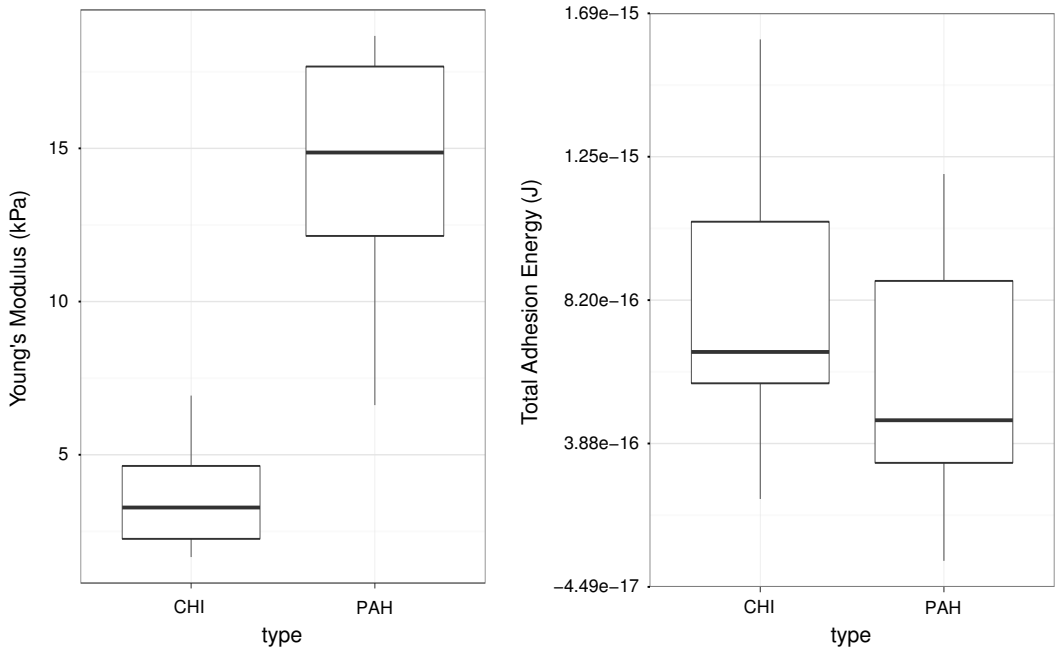


Figure 11: Batch processing: Left: Young's Modulus vs. Experiment type boxplot. Right: "Total Adhesion Energy" vs. "Experiment type" boxplot. Both plots extracted from an `"afmexperiment"` data class variable.

Conclusions

The R programming language has became in the recent years the *"lingua franca"* of statistics and data science. The number of users contributed packages has increased exponentially reaching more 10000 packages. However, although there are several packages for AFM image processing at CRAN, up to our knowledge, this is the first R package for force spectroscopy analysis.

Our goal has been to create a set of functions for the automatic analysis of force-distance curves while being flexible enough to be easily extended by adding algorithms that allow easier analysis in more and more different types of AFM experiments.

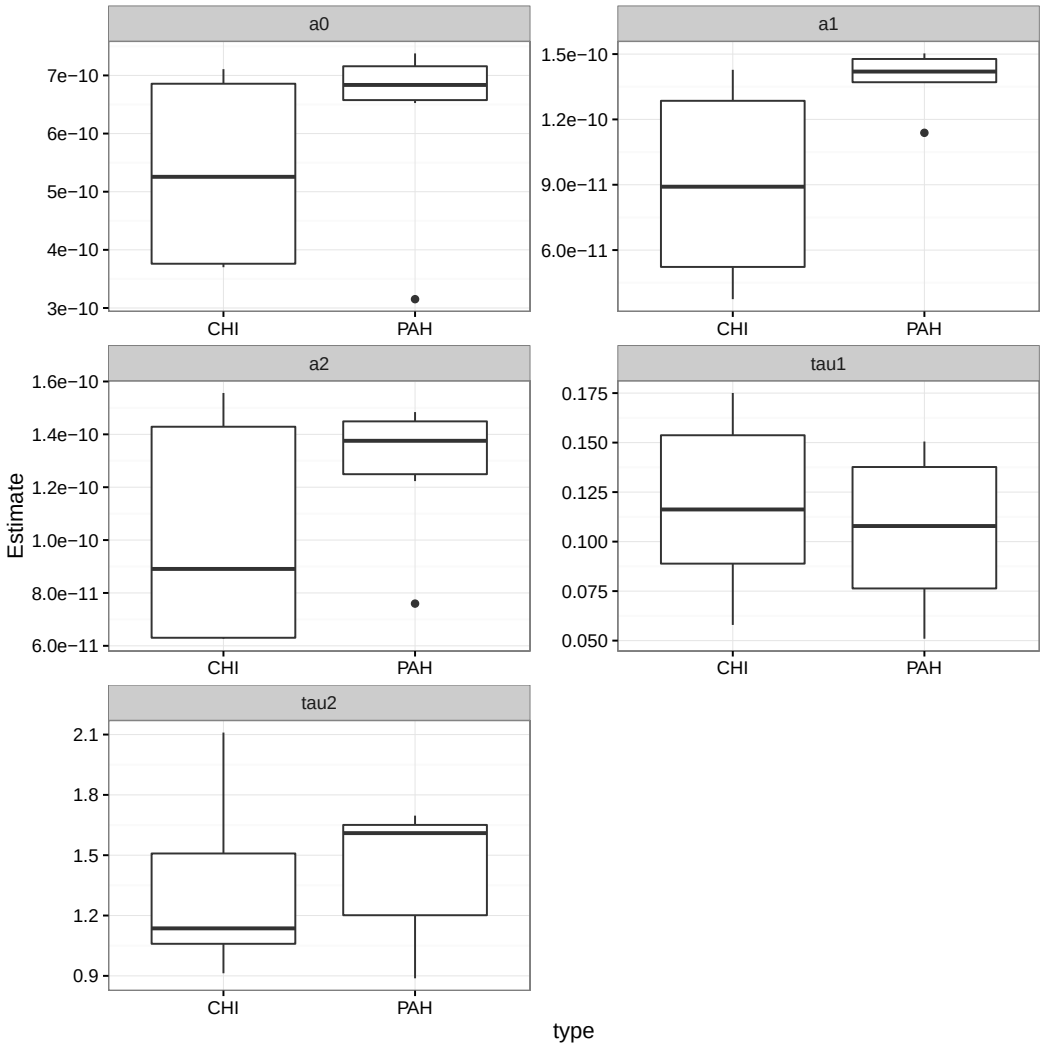


Figure 12: Batch processing: Exponential decay. Boxplot of the values of the estimated parameters in the double exponential decay fit for each experiment type.

For example, in the scope of event detection, the use of wavelets has proven to be very effective in determining protein folding events (García-Massó et al., 2016) or in the detection of jumps in the retract segment of the curve, that may be associated with the formation of tethers (Benítez and Bolós, 2017). Thus, new directions in the development of this **afmToolkit** could be the inclusion of peak detectors.

We sincerely hope that this package will be found useful by the force - spectroscopy scientists and we are willing to recieve feedback from the users.

Acknowledgements

The authors would like to thank Julia Miholich for the testing and bug-finding and Alberto Cencerrado, Jagoba Iturri and Xavi García for making the measurements.

Bibliography

J. Alcaraz, L. Buscemi, M. Grabulosa, X. Trepat, B. Fabry, R. Farré, and D. Navajas. Microrheology of human lung epithelial cells measured by atomic force microscopy. *Biophysical Journal*, 84(3): 2071–2079, 2003. URL https://doi.org/10.1016/S0006-3495(03)75014-0. [p291]

B. Andreopoulos and D. Labudde. Efficient unfolding pattern recognition in single molecule force

spectroscopy data. *Algorithms for Molecular Biology*, 6(1), 2011. URL https://doi.org/10.1186/1748-7188-6-16. [p291]

B. Auguie. *gridExtra: Miscellaneous Functions for "Grid" Graphics*, 2016. URL https://CRAN.R-project.org/package=gridExtra. R package version 2.2.1. [p292]

R. Benítez and V. Bolós. Searching events in afm force-extension curves: A wavelet approach. *Microscopy research and technique*, 80(1):153–159, 2017. URL https://doi.org/10.1002/jemt.22720. [p305]

R. Benitez and J. Toca-Herrera. Looking at cell mechanics with atomic force microscopy: experiment and theory. *Microscopy research and technique*, 77(11):947–958, 2014. URL https://doi.org/10.1002/jemt.22419. [p291]

R. Benítez, S. Moreno-flores, V. J. Bolós, and J. L. Toca-Herrera. A new automatic contact point detection algorithm for afm force curves. *Microscopy research and technique*, 76(8):870–876, 2013. URL https://doi.org/10.1002/jemt.22241. [p291, 295]

R. Best, D. Brockwell, J. Toca-Herrera, A. Blake, D. Smith, S. Radford, and J. Clarke. Force mode atomic force microscopy as a tool for protein folding studies. *Analytica Chimica Acta*, 479(1):87–105, 2003. URL https://doi.org/10.1016/S0003-2670(02)01572-6. [p291]

M. Borkovec, I. Szilagyi, I. Popa, M. Finessi, P. Sinha, P. Maroni, and G. Papastavrou. Investigating forces between charged particles in the presence of oppositely charged polyelectrolytes with the multi-particle colloidal probe technique. *Advances in colloid and interface science*, 179-182:85–98, nov 2012. URL https://doi.org/10.1016/j.cis.2012.06.005. [p291]

H.-J. Butt, B. Cappella, and M. Kappl. Force measurements with the atomic force microscope: Technique, interpretation and applications. *Surface Science Reports*, 59(1-6):1–152, oct 2005. URL https://doi.org/10.1016/j.surfrep.2005.08.003. [p291]

S. L. Crick and F. C.-P. Yin. Assessing micromechanical properties of cells with atomic force microscopy: importance of the contact point. *Biomechanics and modeling in mechanobiology*, 6(3):199–210, apr 2007. URL https://doi.org/10.1007/s10237-006-0046-x. [p291]

W. A. Ducker, T. J. Senden, and R. M. Pashley. Direct measurement of colloidal forces using an atomic force microscope. *Nature*, 353(6341):239–241, 1991. URL https://doi.org/10.1038/353239a0. [p291]

T. V. Elzhov, K. M. Mullen, A.-N. Spiess, and B. Bolker. *minpack.lm: R Interface to the Levenberg-Marquardt Nonlinear Least-Squares Algorithm Found in MINPACK, Plus Support for Bounds*, 2015. URL https://CRAN.R-project.org/package=minpack.lm. R package version 1.2-0. [p292, 299]

S. Garcia-Manyes and F. Sanz. Nanomechanics of lipid bilayers by force spectroscopy with AFM: a perspective. *Biochimica et biophysica acta*, 1798(4):741–9, apr 2010. URL https://doi.org/10.1016/j.bbamem.2009.12.019. [p291]

X. García-Massó, M. C. Huber, J. Friedmann, L. M. Gonzalez, S. M. Schiller, and J. L. Toca-Herrera. Automated detection of protein unfolding events in atomic force microscopy force curves. *Microscopy Research and Technique*, 79(11):1105–1111, 2016. URL https://doi.org/10.1002/jemt.22764. [p305]

N. Gavara. Combined strategies for optimal detection of the contact point in AFM force-indentation curves obtained on thin samples and adherent cells. *Scientific Reports*, 6:21267, Feb. 2016. URL https://doi.org/10.1038/srep21267. [p295]

H. G. Hansma, I. Revenko, K. Kim, and D. E. Laney. Atomic Force Microscopy of Long and Short Double-Stranded, Single-Stranded and Triple-Stranded Nucleic Acids. *Nucleic Acids Research*, 24(4):713–720, feb 1996. URL https://doi.org/10.1093/nar/24.4.713. [p291]

P. Hinterdorfer, W. Baumgartner, H. J. Gruber, K. Schilcher, and H. Schindler. Detection and localization of individual antibody-antigen recognition events by atomic force microscopy. *Proceedings of the National Academy of Sciences*, 93(8):3477–3481, apr 1996. URL https://10.1073/pnas.93.8.3477. [p291]

B. Kainz, E. A. Oprzeska-Zingrebe, and J. L. Toca-Herrera. Biomaterial and cellular properties as examined through atomic force microscopy, fluorescence optical microscopies and spectroscopic techniques. *Biotechnology journal*, 9(1):51–60, jan 2014. URL https://doi.org/10.1002/biot.201300087. [p291]

S. Kasas and G. Dietler. Probing nanomechanical properties from biomolecules to living cells. *Pflügers Archiv : European journal of physiology*, 456(1):13–27, apr 2008. URL https://doi.org/10.1007/s00424-008-0448-y. [p291]

S. Kasas, B. M. Riederer, S. Catsicas, B. Cappella, and G. Dietler. Fuzzy logic algorithm to extract specific interaction forces from atomic force microscopy data. *Review of Scientific Instruments*, 71(5): 2082–2086, 2000. URL https://doi.org/10.1063/1.1150583. [p291]

Y. G. Kuznetsov, J. J. Dowell, J. A. Gavira, J. D. Ng, and A. McPherson. Biophysical and atomic force microscopy characterization of the RNA from satellite tobacco mosaic virus. *Nucleic acids research*, 38(22):8284–94, dec 2010. URL https://doi.org/10.1093/nar/gkq662. [p291]

D. C. Lin, E. K. Dimitriadis, and F. Horkay. Robust strategies for automated AFM force curve analysis–I. Non-adhesive indentation of soft, inhomogeneous materials. *Journal of biomechanical engineering*, 129 (3):430–40, jun 2007a. URL https://doi.org/10.1115/1.2720924. [p291, 295]

D. C. Lin, E. K. Dimitriadis, and F. Horkay. Robust strategies for automated AFM force curve analysis-II: adhesion-influenced indentation of soft, elastic materials. *Journal of biomechanical engineering*, 129 (6):904–12, dec 2007b. URL https://doi.org/10.1115/1.2800826. [p295]

A. E. Lopez, S. Moreno-Flores, D. Pum, U. B. Sleytr, and J. L. Toca-Herrera. Surface dependence of protein nanocrystal formation. *Small*, 6(3):396–403, 2010. URL https://doi.org/10.1002/smll.200901169. [p291]

P. E. Marszalek, H. Lu, H. Li, M. Carrion-Vazquez, A. F. Oberhauser, K. Schulten, and J. M. Fernandez. Mechanical unfolding intermediates in titin modules. *Nature*, 402(6757):100–3, nov 1999. URL https://doi.org/10.1038/47083. [p291]

K. A. Melzak and J. L. Toca-Herrera. Atomic force microscopy and cells: Indentation profiles around the afm tip, cell shape changes, and other examples of experimental factors affecting modeling. *Microscopy Research and Technique*, 78(7):626–632, 2015. URL https://doi.org/10.1002/jemt.22522. [p291]

S. Moreno-Flores and J. L. Toca-Herrera. *Hybridizing Surface Probe Microscopies: Toward a Full Description of the Meso- and Nanoworlds*, volume 8. CRC. Taylor and Francis, Boca Raton, 2013. [p291]

S. Moreno-Flores, R. Benitez, M. dM Vivanco, and J. L. Toca-Herrera. Stress relaxation and creep on living cells with the atomic force microscope: a means to calculate elastic moduli and viscosities of cell components. *Nanotechnology*, 21(44):445101, nov 2010a. URL https://doi.org/10.1088/0957-4484/21/44/445101. [p298]

S. Moreno-Flores, R. Benitez, M. D. Vivanco, and J. L. Toca-Herrera. Stress relaxation microscopy: imaging local stress in cells. *Journal of biomechanics*, 43(2):349–54, jan 2010b. URL https://doi.org/10.1016/j.jbiomech.2009.07.037. [p298]

D. J. Müller and Y. F. Dufrêne. Atomic force microscopy as a multifunctional molecular toolbox in nanobiotechnology. *Nature nanotechnology*, 3(5):261–9, may 2008. URL https://doi.org/10.1038/nnano.2008.100. [p291]

J. L. Ortega-Vinuesa, P. Tengvall, and I. Lundström. Molecular packing of HSA, IgG, and fibrinogen adsorbed on silicon by AFM imaging. *Thin Solid Films*, 324(1-2):257–273, 1998. URL https://doi.org/10.1016/S0040-6090(98)00363-0. [p291]

E. Riande, R. Díaz-Calleja, M. G. Prolongo, R. M. Masesgosa, and C. Salom. *Polymer viscoelasticity : stress and strain in practice*. CRC Press, New York, 1999. [p298]

M. Rief. Single Molecule Force Spectroscopy on Polysaccharides by Atomic Force Microscopy. *Science*, 275(5304):1295–1297, feb 1997. URL https://doi.org/10.1126/science.275.5304.1295. [p291]

D. Rudoy, S. G. Yuen, R. D. Howe, and P. J. Wolfe. Bayesian change-point analysis for atomic force microscopy and soft material indentation. *Journal of the Royal Statistical Society. Series C: Applied Statistics*, 59(4):573–593, 2010. URL https://doi.org/10.1111/j.1467-9876.2010.00715.x. [p295]

H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer, New York, 2016a. [p292]

H. Wickham. *scales: Scale Functions for Visualization*, 2016b. URL https://CRAN.R-project.org/package=scales. R package version 0.4.0. [p292]

H. Wickham and W. Chang. *devtools: Tools to Make Developing R Packages Easier*, 2016. URL https://CRAN.R-project.org/package=devtools. R package version 1.12.0. [p292]

H. Wickham and R. Francois. *dplyr: A Grammar of Data Manipulation*, 2015. URL https://CRAN.R-project.org/package=dplyr. R package version 0.4.3. [p]

Rafael Benítez
Department of Bussiness Mathematics. Univeristy of Valencia
Avda. Tarongers s/n. 46022. Valencia
Spain
rabesua@uv.es

Vicente J. Bolós
Department of Bussiness Mathematics. Univeristy of Valencia
Avda. Tarongers s/n. 46022. Valencia
Spain
vicente.bolos@uv.es

José-Luis Toca-Herrera
Department of Nanobiotechnology. Institute for Biophysics.
University of Natural Resources and Life Sciences - BOKU Wien
Simon Zeisel Haus. Muthgasse 11/II.1190 Wien
Austria
jose.toca-herrera@boku.ac.at

The welchADF Package for Robust Hypothesis Testing in Unbalanced Multivariate Mixed Models with Heteroscedastic and Non-normal Data

by Pablo J. Villacorta

Abstract A new R package is presented for dealing with non-normality and variance heterogeneity of sample data when conducting hypothesis tests of main effects and interactions in mixed models. The proposal departs from an existing SAS program which implements Johansen's general formulation of Welch-James's statistic with approximate degrees of freedom, which makes it suitable for testing any linear hypothesis concerning cell means in univariate and multivariate mixed model designs when the data pose non-normality and non-homogeneous variance. Improved type I error rate control is obtained using bootstrapping for calculating an empirical critical value, whereas robustness against non-normality is achieved through trimmed means and Winsorized variances. A wrapper function eases the application of the test in common situations, such as performing omnibus tests on all effects and interactions, pairwise contrasts, and tetrad contrasts of two-way interactions. The package is demonstrated in several problems including unbalanced univariate and multivariate designs.

Introduction

The problem of testing for mean equality between several groups can be accomplished using classical techniques such as Student's t test, when only two groups are compared, or ANOVA when more than two groups are involved. Both of them have been widely applied in the past in a number of areas ranging from ecology and biology to psychology, social sciences and medicine (Levin, 1997; Coates and McKenzie-Mohr, 2010), although their use tends to decrease for the reasons mentioned next.

In order for these approaches to work well, the data must satisfy three conditions, namely independence, normality and homoscedasticity of the errors. While ANOVA is known to be robust to small departures from normality, homogeneity of population variances is crucial as concluded by simulation studies in which these methods have been found to exhibit type I error rates oscillating from too conservative to extremely liberal, specially in unbalanced designs leading to very heterogeneous cell variances. The behaviour depends on the relation between the variance of the cell with the fewest observations and the number of observations contained in it (Milligan et al., 1987). Indeed, data from a number of experiments conducted in the aforementioned research fields often exhibit non-homogeneous variances. This is not a problem for one-way designs since the built-in function `oneway.test` in package **stats** is able to account for different variances. Unfortunately, real-world studies usually require more complex designs, like the typical mixed between x within-subjects designs for clinical trials involving either animals or persons. For both reasons, the application of simple ANOVA in serious analyses of experimental data is nowadays not very common.

A distinction should be made here on what homogeneity of variances means depending on the design being considered (Lix and Keselman, 1995). In univariate settings with between-subjects factors only, all the cell variances should be equal, while in multivariate settings, it refers to the equality of the population covariance matrices across all cells. In mixed designs with at least one between-subjects factor, a set of orthonormalized contrasts on the repeated measures must have common covariance matrices (sphericity assumption), and all those matrices must be equal across all cells of the between-subject factors. When both conditions are met, it is said that multisample sphericity holds (Huynh, 1978).

A number of alternatives have been proposed to overcome the parametric restrictions (Higgins, 2003). Traditionally the most widely used choices have been nonparametric tests such as Mann-Whitney-Wilcoxon rank sum test and Kruskal test for two groups, or Wilcoxon signed-rank test and Friedman test (for which paired versions exist) for more than two groups. However, they cannot handle multiple factors or interactions. Generalized Linear Models can deal with multiple factors and interactions with non-normal data, but require specifying the link function and are unable to handle repeated measures. Since our study focuses on the most general models, i.e. those with between-subjects and within-subjects interactions, the aforementioned tests are not useful. The generalization of Mixed Models, namely Generalized Linear Mixed Models (Bolker et al., 2009), do constitute a valid alternative. They present some drawbacks, though, such as being complex to apply and interpret, not very widely available, and requiring a particular treatment for each problem since a suitable link

function must be supplied in each case, which is not always possible.

Two surveys on nonparametric techniques in experimental design can be found in Sawilowsky (1990); Salazar-Alvarez et al. (2014). In these works, several rank transformation variants are emphasized, as they constitute the most widely used nonparametric approach for detecting interactions Conover (2012). Among them, the Aligned Rank Transform (Higgins and Tashtoush, 1994), for which an implementation in R has been made available in package **ART** (Villacorta, 2015), is one of the best performing for this task, keeping type I error rates close to the theoretical significance level while preserving good power. Although it has been applied to a split-plot design (one between- and one within-subjects factor) in Beasley (2002), showing good type I error rates and power, it lacks a general unified formulation for mixed models with any number of between- and within-subjects factors that also works in unbalanced and multivariate settings. Erce-Hurn and Mirosevich (2008); Ruscio and Roche (2012) constitute two more broad surveys (the latter dealing with variance heterogeneity in depth) covering both classical nonparametric methods and recent research efforts like the one implemented here, which is cited in both works.

Some other valid alternatives for nonparametric analysis of any mixed model are the Improved General Approximation (IGA), the generalization of Welch-James (WJ) test statistic, the Kenward-Roger correction with mixed models (KR), and the modified Brown and Forsythe (MBF) procedure. They all do a correction for the degrees of freedom to account for heterogeneous variances, hence the name ADF for *approximate degrees of freedom*. The IGA (Huynh, 1978) was specifically developed to account for multisample sphericity violations in repeated measures designs by adjusting the critical value, and was generalized to any mixed model by Algina (1997). Similarly, Welch's non-pooled statistic with approximate degrees of freedom (ADF) (Welch, 1951) was also conceived for this purpose, using the sample data to estimate the error degrees of freedom. Several non-pooled ADF statistics have been proposed later but all can be derived from the general matrix formulation of Welch's statistic given by Lix and Keselman (1995) based on Johansen (1980), which makes it applicable for univariate and multivariate mixed models with an arbitrary number of effects. Lix and Keselman (1995) shows how the same statistic can be employed in different models for both omnibus contrasts (testing whether a given effect is significant or not) and pairwise comparisons for a given effect (for every pair of categories of a given effect, testing whether the response is significantly different for one of the categories against one another). Generally, the IGA and WJ statistic perform similarly; however a slight advantage favorable to WJ has been reported by Algina (1997) in some contexts. The WJ ADF approach has been successfully tested in nonparametric analysis of a variety of mixed models; see Keselman et al. (2003) and references therein. The KR correction for the degrees of freedom can be implemented on top of a conventional mixed model and performs similarly to the MBF procedure with slight advantage for the latter (Vallejo and Livacic-Rojas, 2005). Both yield reasonably good results. With respect to the comparison between the MBF and the generalized WJ statistic, there is no consensus about the results. In most conditions they perform similarly, but some authors state that MBF is better at detecting interaction effects when the number of subjects is not high enough (Vallejo et al., 2001, 2006). However, to the best of our knowledge, there is no general formulation of MBF for any mixed model with an arbitrary number of between- x within-subjects factors, although it has been tested in split-plot designs (Vallejo et al., 2006) (which are probably the most common design in medicine and particularly in psychological studies) and factorial designs (Vallejo et al., 2008).

Software available for robust testing of mixed models not meeting parametric assumptions

Wilcox (2012) constitutes an important source dealing with robust estimation. The book is accompanied by an R package called **WRS** [1] that implements all the methods reviewed in the book, including the Welch-James test following Johansen's approach with robust mean estimators described in sections 7.2, 8.6 and 8.7 which our package **welchADF** also implements. The functions described in those sections, `bwtrim`,`t1way`,`t1waybt`,`t2way`,`t2way`,`t3way`, include the most common designs such as one, two and three-way and split-plot (one between x one within subjects designs) also with bootstrapping and trimming. Although not included in **WRS2**, the original **WRS** exposes functions `bbw-`,`bww-` for between x between x within, and between x within x within subjects designs, and their trimmed and bootstrap versions. While useful, they do not provide a uniform, easy-to-use interface in a single function valid for any mixed design.

There exists a CRAN task force on robust statistical methods[2]. Unfortunately, none of the packages mentioned there implements the aforementioned approaches. Nevertheless, packages **robustbase** (Maechler et al., 2016), **robust** (Wang et al., 2017) (by the authors of Maronna et al. (2006)) and function

[1] Not on CRAN but in `http://github.com/nicebread/WRS`. A less-comprehensive but more user-friendly version can be found in package **WRS2** (Mair and Wilcox, 2017).

[2] `http://cran.r-project.org/web/views/Robust.html`

`rlmer` in package **robustlmm** (Koller, 2017) are some remarkable efforts. The latter implements the techniques proposed in Koller (2013) for linear mixed models on a basis of a parametric model having some contaminated data (Koller, 2016). However, it does not focus on testing inherently heterogeneous and non-normal data. Package **nlme** (Pinheiro et al., 2017) provides a way of capturing and modeling variance heterogeneity in mixed models through the argument `weights` of function `lme`, which can be set to different covariance matrix structures that are fitted from the data. The well-known package **lme4** (Bates et al., 2016, 2015) is also a good choice for dealing with non-normal models in presence of within-subjects effects (called generalized linear mixed models as an extension to generalized linear models where the user can specify the probabilistic model to be used). Package **glmmADMB** [3] (Skaug et al., 2016; Fournier et al., 2012) has a similar purpose.

SAS (SAS Institute, 2011) implementations of the WJ statistic, MBF statistic (split-plot and factorial designs only) and IGA do exist; see Keselman et al. (2003); Vallejo et al. (2006); Algina (1997) respectively. While SAS is still widely used mainly in social and biomedical sciences, it is proprietary software. The same applies to the ERP-PCA software (Dien, 2010) written in Matlab. Interestingly, ERP-PCA incorporates a Matlab translation of the SAS code described in Keselman et al. (2003) for the WJ statistic. For these reasons, together with the fast expansion of R and open-source statistical software in general -closely related to the growing interest in reproducible research- among researchers of many different disciplines, we consider the R package introduced here a useful effort.

Main contributions

The present work describes an R package called **welchADF** (Villacorta, 2017) that implements Johansen's formulation of the Welch-James test, with two additional improvements: first, the use of trimmed means and Winsorized variances to deal with non-normality, and second, the use of bootstrap for calculating an empirical critical value for achieving better type I error control. Both aspects are mentioned in Wilcox et al. (1998) and implemented in Keselman et al. (2003). Trimmed means and Winsorized variances are being used in medical and behavioral research in the last years; see Müller et al. (2011); Aronoff et al. (2011); Ryzin et al. (2011).

The core of our code is an R translation of the SAS program[4] described in Keselman et al. (2003). A new wrapper function has been built on top of it that poses the following benefits:

- It can be applied to univariate and multivariate mixed models with an arbitrary number of within- and between-subjects effects.
- It simplifies some common tasks such as performing omnibus tests on effects or interactions, multiple pairwise comparisons on the levels of one factor, and tetrad contrasts. All of these can be done without indicating the contrast matrices, which are automatically formed by the program depending on the kind of test required and the number of levels found in each factor.
- It provides a more natural and uniform data input mechanism through data frames that do not depend on the model specified. In the original SAS code, the input data had to be carefully arranged in matrices whose shape had to mirror the experimental design being analyzed in each problem, which can be error-prone.
- It integrates with other similar packages of the R ecosystem through a formula interface and also provides additional interfaces that accept model objects returned by some commonly used functions such as `stats::lm`, `lme4::lmer` and `stats::aov`.
- It enables selecting one among several built-in p-value correction methods when performing multiple pairwise comparisons.

There are several reasons that justify an R implementation of this particular test:

- The generalized WJ test described in Lix and Keselman (1995); Keselman et al. (2003) has good theoretical properties and has proven successful in controlling type I error rate while preserving high power. Moreover, the use of trimmed means and Winsorized variances can protect both against skewness and outliers in the data, as noted by Keselman et al. (2008). The percentage of trimming can be adjusted to deal with higher ratios of outliers and skewness. The statistic is also able to cope with heterogeneous variances, which commonly (but not only) arise when having very different cell sample sizes. In this sense, one should notice the sample size requirement stated right after this list.
- These two works have received a number of citations, and the approach explained there is being used in current research in different fields such as medicine (Dien et al., 2008; Dien, 2010), psychology (Müller et al., 2011; Kayser et al., 2014; Huang and Jun, 2015) and behavioral research (Symes et al., 2010), just to cite a few.

[3] http://glmmadmb.r-forge.r-project.org/

[4] Available at http://homepage.usask.ca/~lml321/Program.pdf

- The function interface is simple and the test can be used in a straightforward way for the most common tasks. This may contribute positively towards its adoption by the research community, specially by researchers with little expertise in statistics, but with an understanding of the importance of applying suitable, robust techniques when parametric conditions are not met.
- Despite the existence of different alternatives explained before, some of them also implemented in R, these are either not applicable to multivariate mixed models with heterogeneous variance or non-normal data, or are generally complex and more difficult to use.

The WJ approach also has some disadvantages. The first one is the sample size needed to assure an effective control of type I error under some (somewhat extreme) circumstances, specially in repeated-measures designs, like when the cell with the fewest subjects presents the largest variance (i.e. cell size and cell variance are negatively paired). In general, the number of subjects of the smallest cell should be four or five times greater than the number of repeated measures minus one, and sometimes even more when testing an interaction. However, when combined with trimmed means, robustness is increased and some of these problems are mitigated as a much smaller number of subjects are required (Keselman et al., 2000). The second drawback is that **welchADF** is only applicable to categorical predictors. In case the design has numeric predictors, the reader may try the packages mentioned in the preceding section, as well as **gamm4** (Wood and Scheipl, 2017) and **mgcv** (Wood, 2017) for fitting generalized additive mixed models.

Finally, the application of bootstrap to contaminated data has been extensively studied and even questioned by some authors in the past (Singh, 1998), specially regarding numerical instability: some bootstrap samples that intervene in the computation of the final bootstrapped estimate may contain a higher proportion of outliers than the general dataset, and therefore be too heavily influenced by them (Salibián-Barrera et al., 2008). Singh (1998) proposed using bootstrapping with Winsorized observations, which is what we do in this package when enabling trimming and bootstrapping at the same time, as it provides some additional benefits. Salibián-Barrera et al. (2008) introduce a fast and robust bootstrap (FRB) method that improves classical bootstrapping. Although it has not been incorporated to **welchADF**, it may be done in the future.

The remainder of this contribution is structured as follows. In first place, the mathematical background is briefly reviewed. In second place, we present the function exposed by the package and explain its arguments, together with some issues regarding the arrangement of the data. In third place, we address three case studies, namely a univariate one-way between-subjects design, a two-way factorial design, a mixed design, and a doubly multivariate design analyzed as a multivariate mixed design. Finally, we present some conclusions and further work.

The Welch-James ADF test statistic

Here we summarize the theoretical background given in Lix and Keselman (1995); Keselman et al. (2003). Following the General Linear Model,

$$\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\xi} \tag{1}$$

where $\mathbf{Y}$ is an $N \times p$ matrix of observations on p dependent variables or p repeated measures, N is the sample size, $\mathbf{X}$ is the design matrix (formed by zeros and ones, such that $\text{rank}(\mathbf{X}) = r$ with r being the number of different groups or cells[5]), $\boldsymbol{\beta}$ is an $r \times p$ matrix of non random (unknown) parameters (population means) and $\boldsymbol{\xi}$ is an $N \times p$ matrix of random error components.

If we denote by $\mathbf{Y}_j (j = 1, ..., r)$ the submatrix of $\mathbf{Y}$ that contains the observations of the n_j subjects of the j-th cell, the original parametric model assumes $\mathbf{Y}_j \sim \mathcal{N}(\boldsymbol{\beta}_j, \boldsymbol{\Sigma}_j)$ where $\boldsymbol{\beta}_j = (\mu_{j1}, ..., \mu_{jp})$ is the j-th row of matrix $\boldsymbol{\beta}$, and $\boldsymbol{\Sigma}_j \neq \boldsymbol{\Sigma}_{j'}$ when $j \neq j'$.

We proceed to explain how population means and variances are estimated. The matrix of population means can be estimated by the usual least-squares approach (Eq. 2) or by using robust estimation techniques such as trimming, discussed later. Now, let $\mathbf{X}_j (j = 1, ..., r)$ be the j-th column of $\mathbf{X}$, composed of zeros and ones, and let $\mathbf{1}_p$ be a $p \times 1$ vector of ones. Define $\mathbf{Y}_j = \mathbf{Y} \circ (\mathbf{X}_j \mathbf{1}_p^T)$ as the $N \times p$ matrix that results of the Hadamard (i.e. element-wise) product between matrices $\mathbf{Y}$ and $\mathbf{X}_j \mathbf{1}_p^T$. Then, the following expressions are used to estimate population means and variances:

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y} \tag{2}$$

$$\hat{\boldsymbol{\Sigma}}_j = \frac{(\mathbf{Y}_j - \mathbf{X}_j\hat{\boldsymbol{\beta}}_j)^T(\mathbf{Y}_j - \mathbf{X}_j\hat{\boldsymbol{\beta}}_j)}{n_j - 1} \tag{3}$$

[5]This does not entail that one-way designs are the only possible. The formulation is valid also for several factor designs.

The matrix formulation of the WJ statistic always tests the following general linear hypothesis:

$$H_0 : \mathbf{R}\boldsymbol{\mu} = \mathbf{0} \tag{4}$$

with $\mathbf{R} = \mathbf{C} \otimes \mathbf{U}^T$. In this expression, $\otimes$ is the Kronecker product, $\mathbf{C}$ is a $df_C \times r$ matrix that indicates the contrasts on the between-subjects effects, so that rank($\mathbf{C}$) $= df_C \leq r$, and $\mathbf{U}$ is a $p \times df_U$ matrix that indicates the contrasts on the within-subjects effects, so that rank($\mathbf{U}$) = $df_U \leq p$. Therefore $\mathbf{R}$ has $df_C df_U$ rows and rp columns. Note that $\boldsymbol{\mu} = \text{vec}(\boldsymbol{\beta}^T) = (\boldsymbol{\beta}_1, ...\boldsymbol{\beta}_r)^T$, which is a column vector with rp elements obtained by stacking the columns of $\boldsymbol{\beta}^T$, one on top of another. Matrices $\mathbf{C}$ and $\mathbf{U}$ determine the type of contrast being performed, be it an omnibus contrast, a pairwise contrast, etc. We provide details about the structure of both matrices in the general case in Section 42.2.1.

The generalized Welch-James test statistic presented by Johansen in Johansen (1980) is

$$\begin{aligned} T_{\text{WJ}} &= (\mathbf{R}\hat{\boldsymbol{\mu}})^T (\mathbf{R}\hat{\boldsymbol{\Sigma}}\mathbf{R}^T)^{-1} (\mathbf{R}\hat{\boldsymbol{\mu}}) \\ \hat{\boldsymbol{\Sigma}} &= \text{diag}(\hat{\boldsymbol{\Sigma}}_1/n_1, ..., \hat{\boldsymbol{\Sigma}}_r/n_r) \end{aligned} \tag{5}$$

where $\hat{\boldsymbol{\mu}}$ is an estimate of $\boldsymbol{\mu}$ (either with LS or any other technique), and $\hat{\boldsymbol{\Sigma}}$ is a block diagonal matrix whose blocks are $\hat{\boldsymbol{\Sigma}}_j/n_j$. It is known that T_{WJ}/c approximately follows an $F(\nu_1; \nu_2)$ where

$$\begin{aligned} \nu_1 &= df_C df_U \quad ; \quad \nu_2 = \nu_1(\nu_1+2)/(3A) \quad ; \quad c = \nu_1 + 2A - (6A)/(\nu_1+2) \\ A &= \frac{1}{2}\sum_{j=1}^{r} \frac{\text{tr}\left[\hat{\boldsymbol{\Sigma}}\mathbf{R}^T(\mathbf{R}\hat{\boldsymbol{\Sigma}}\mathbf{R}^T)^{-1}\mathbf{R}\mathbf{Q}_j\right]^2 + \left(\text{tr}\left[\hat{\boldsymbol{\Sigma}}\mathbf{R}^T(\mathbf{R}\hat{\boldsymbol{\Sigma}}\mathbf{R}^T)^{-1}\mathbf{R}\mathbf{Q}_j\right]\right)^2}{n_j - 1} \end{aligned}$$

where tr is the trace of an square matrix (sum of the elements on the main diagonal), and $\mathbf{Q_j}$ is an $rp \times rp$ matrix associated with X_j in which the (s,t)-th diagonal block of $\mathbf{Q}_j = \mathbf{I}_p$ when $s = t = j$ and is $\mathbf{0}$ otherwise.

In a between-subjects design (no within-subjects factors), $\mathbf{U}$ must be set to I_p where p is the number of dependent variables (in univariate designs, it reduces to $\mathbf{U} = 1$). In a within-subjects design (no between-subjects factors), $\mathbf{C}$ must be set to 1 both in the univariate and multivariate cases.

Structure of the contrast matrices

The preceding formulation of the model is valid for any type of contrasts. Most often the user may want to perform two types of contrasts, namely omnibus contrasts to check whether a given effect or interaction is statistically significant, and in case it is, post-hoc pairwise contrasts on one effect or interaction to check whether the response associated to some of the levels of that factor or interaction is statistically different than the responses associated to other levels of the factor.

Omnibus contrasts This test is aimed at checking whether the level adopted by a given variable of interest (effect or interaction) has an influence over the response variable. In the simplest case, consider a one-way design, either univariate or multivariate, whose single factor A has a different levels. Then $\mathbf{C}$ would be an $(a-1) \times a$ matrix specifying linearly independent contrasts between the levels. We will call this matrix $\mathbf{C}_A$ (capital A) because it is the matrix we use to conduct an omnibus test on effect A. If for instance, $a = 3$, we would have

$$\mathbf{C}_A = \begin{bmatrix} 1 & -1 & 0 \\ 1 & 0 & -1 \end{bmatrix} \quad ; \quad \mathbf{U} = \mathbf{I}_p$$

Algina and Olejnik (1984) provide a general formulation to compose matrix $\mathbf{C}$ for omnibus tests on factorial designs with several between-subjects factors. The same idea, slightly modified, is also valid to compose matrix $\mathbf{U}$ in designs with several within-subjects factors. These procedures have been implemented in our **welchADF** package. Let C_a (non-capital a) be the $(a-1) \times a$ matrix of linear contrasts associated to the a levels of effect A. Its rows define $a-1$ linearly independent contrasts between the levels of A. When the design has no more factors than A, then $\mathbf{C}_A = C_a$ as in the case above. However, when more than one between-subjects factor exists, then $\mathbf{C}$ has to be properly set according to the factor to which we want to apply an omnibus test. In those cases, $\mathbf{C}$ is the Kronecker product of one matrix (or vector) per factor existing in the design, as follows.

Assume a between-subjects design with four effects A, B, C, D. Let $\mathbf{1}_a^T$ denote a $(1 \times a)$ vector of ones. If we want to perform an omnibus contrast on a main effect, say B, then $\mathbf{C} = \mathbf{C}_B = \mathbf{1}_a^T \otimes C_b \otimes \mathbf{1}_c^T \otimes \mathbf{1}_d^T$. In other words, if the factor matches the effect being tested, the corresponding contrast matrix appears in the product; otherwise, a vector of ones appears. The same applies to interactions. If we want to test the $B \times D$ interaction, for instance, we would have $\mathbf{C} = \mathbf{C}_{BD} = \mathbf{1}_a^T \otimes C_b \otimes \mathbf{1}_c^T \otimes C_d$. Since factors B and D are involved in the interaction BD being tested, their contrast matrices appear in the

product, while a vector of ones appears in the positions of the remaining factors of the design.

For within-subjects designs a similar rule applies. Assume that now A is a within-subjects factor, and let $U_a = C_a^T$, so that the columns of U_a define $a-1$ linearly independent contrasts between the levels of A. In a within-subjects design with several factors, the transposes of the U contrast matrices of those factors involved in the effect or interaction being tested appear in the Kronecker product; otherwise, a row vector of ones is used in that place as explained before. If the design is multivariate with p dependent response variables, an additional factor $\mathbf{I}_p$ always appears in the last place of the product. At the end, the result of the Kronecker product must be transposed again to obtain $\mathbf{U}$. For example, in a two-factor within-subjects multivariate design, the $\mathbf{U}$ matrices for conducting ominbus tests on effects A, B and the interaction AB would be, respectively, $\mathbf{U}_A = (U_a^T \otimes \mathbf{1}_b^T \otimes \mathbf{I}_p)^T$, $\mathbf{U}_B = (\mathbf{1}_a^T \otimes U_b^T \otimes \mathbf{I}_p)^T$, and $\mathbf{U}_{AB} = (U_a^T \otimes U_b^T \otimes \mathbf{I}_p)^T$.

In case we want to test a main effect (either a between- or a within-subjects effect) in a design containing both between- and within-subjects factors, the same rules apply: $\mathbf{C}$ and $\mathbf{U}$ are composed separately, and one of them will (for sure) be the result of the Kronecker product of vectors of ones only (including $\mathbf{I}_p$ as well when constructing $\mathbf{U}$ if the design is multivariate). Finally, if we want to test an interaction involving one or more between-subjects factors and one or more within-subjects, $\mathbf{C}$ must be formed as if we were testing only the between-subjects factors involved in the mixed interaction, and $\mathbf{U}$ as if testing the within-subjects factors, following the rules explained above.

Pairwise contrasts Now for a given effect, we are aimed at testing for every pair of categories of the effect whether the response is significantly different for one of the categories against one another. The procedure is similar to the omnibus contrasts. The only difference is that contrast matrices C_a and U_a associated to an effect A are replaced by contrast vectors, as follows. When testing for significant differences between factor levels j and j' of an effect A, either C_a is replaced by a row vector $\mathbf{c}_{jj'}$ if A is a between-subjects factor, or U_a is replaced by a column vectors $\mathbf{u}_{jj'}$ if A is a within-subjects factor. In a pairwise contrast vector, all positions are set to 0 except for those corresponding to the factor levels j and j' being tested, which are set to 1 and -1 respectively. These vectors are then used in the corresponding positions of the Kronecker products described before. For probing interactions (once the omnibus test on such interaction proved significant), tetrad contrasts have been implemented in our package. The null hypothesis being tested in a tetrad contrast involving two factors A and B, from which the interaction between levels j and j' from A, and k and k' from B is being tested, can be written as

$$H_{jj';kk'} : (\mu_{jk} - \mu_{jk'}) - (\mu_{j'k} - \mu_{j'k'}) = 0 \tag{6}$$

Trimmed means and Winsorized variances

Trimmed means help mitigate the effects of non-normality. When least-squares means are substituted by trimmed means, the null hypotheses being tested are the equality of population trimmed means: $\mathbf{R}\boldsymbol{\mu}^{(t)} = \mathbf{0}$. Let $Y_{(1)j} \leq Y_{(2)j} \leq ... \leq Y_{(n_j)j}$ be the sorted observations of the j-th group, and $g_j = \lceil \gamma n_j \rceil$ with γ being the proportion of observations to be trimmed in each tail of the distribution. Therefore the sample size for the j-th group becomes $h_j = n_j - 2g_j$, and its sample trimmed mean is computed by averaging the h_j central observations of that group:

$$\hat{\mu}_j^{(t)} = \frac{1}{h_j} \sum_{k=g_j+1}^{n_j-g_j} Y_{(k)j} \tag{7}$$

Some authors suggest using 20 % trimming.

The sample Winsorized mean is a similar measure that is computed by replacing all observations smaller than $Y_{(g_j+1)j}$ (i.e. the 20th percentile) by that value, and those larger than $Y_{(n_j-g_j)j}$ (i.e. the 80th percentile) by that value, and then averaging over all the (modified) observations. In the j-th group:

$$\hat{\mu}_j^{(W)} = \frac{1}{n_j} \sum_{i=1}^{n_j} X_{ij}, \quad \text{where} \quad X_{ij} = \begin{cases} Y_{(g_j+1)j} & \text{if} \quad Y_{ij} \leq Y_{(g_j+1)j} \\ Y_{ij} & \text{if} \quad Y_{(g_j+1)j} < Y_{ij} < Y_{(n_j-g_j)j} \\ Y_{(n_j-g_j)j} & \text{if} \quad Y_{ij} \geq Y_{(n_j-g_j)j} \end{cases} \tag{8}$$

This measure is required to compute the sample Winsorized variance:

$$\hat{\sigma}_j^{2(W)} = \frac{1}{n_j - 1} \sum_{i=1}^{n_j} (X_{ij} - \hat{\mu}_j^{(W)})^2 \tag{9}$$

Therefore, in order to compute the trimmed version of the WJ statistic, $T_{WJ}^{(t)}$, trimmed means replace

least-squares means, Winsorized variances replace least-squares variances, and the new sample sizes h_j replace the original n_j in all groups. Past studies have found the trimmed version of the WJ statistic to be more robust and provide better type I error control to non-normality, also in more complex designs; see Keselman et al. (2000) and references therein.

Bootstrapping to obtain an empirical critical value

After computing the cell trimmed means, let $C_{ij} = Y_{ij} - \hat{\mu}_j^{(t)}$, i.e. the C_{ij} are shifted observations so that the null hypothesis of equal trimmed means is true in the sample. Repeat B times the next two steps (B is the user-supplied number of bootstrap simulations): (i) for each cell, generate a bootstrap sample of size n_j by sampling with replacement from the original sample. When the bootstrap samples of all the cells are put together, an N-sample bootstrap dataset is obtained; (ii) compute the value $F^{*(t)} = T_{WJ}^{(t)}/c$ on the bootstrap dataset. Sort the B values obtained in ascending order, $F_{(1)}^{*(t)} \leq ... \leq F_{(B)}^{*(t)}$. An estimate of an appropriate critical value is $F_{(a)}^{*(t)}$ where $a = (1-\alpha)B$ rounded to the nearest integer. This critical value should be compared with $T_{WJ}^{(t)}/c$ computed on the original data. The null hypothesis $\mathbf{R}\boldsymbol{\mu}^{(t)} = \mathbf{0}$ of trimmed means equality will be rejected if $T_{WJ}^{(t)}/c \geq F_{(a)}^{*(t)}$.

The process explained before applies to omnibus contrasts. For focused contrasts such as pairwise tests on marginal means, the same idea with minor modifications is used. For more details as well as a generalization to other designs, see Keselman et al. (2003) and references therein.

Effect size and confidence intervals

As stated in Keselman et al. (2008); APA (2013), it is now widely recommended to report an estimate of the effect size when performing a hypothesis test. Many different measures exist for this purpose, although few of them are valid for non-homogeneous variances. The approach implemented in our package was proposed in Keselman et al. (2008) and has the following formulation for the case of two groups:

$$\hat{\delta}_j^{(R)} = \eta \frac{\hat{\mu}_2^{(t)} - \hat{\mu}_1^{(t)}}{\hat{\sigma}_j^{(W)}} \tag{10}$$

Note this approach uses trimmed means and Winsorized standard deviation and for that reason, it is robust to non-normality. Factor η stands for a scaling factor of the Winsorized standard deviation. In case 20 % trimming is used (as recommended), $\eta = .642$ which is the Winsorized standard deviation for a 20% trimmed standard normal distribution. In our code, η is computed according to the percentage of trimming indicated by the user. For building a robust CI around this value, a percentile bootstrap method is run to determine the empirical bounds of the interval as recommended in Keselman et al. (2008).

An R implementation of the WJ statistic

The WJ test is implemented in our package as an S3 generic called `welchADF.test`. The name has been chosen to be compliant with other existing tests such as `t.test`,`wilcox.test`, etc. The function receives parameters to modulate its behaviour, such as the type of contrast to be performed (omnibus or pairwise), whether trimming should be employed or not, and if employed, the percentage of data to be trimmed at each side, and whether bootstrapping should be used or not. The default S3 method expects a `data.frame` in the `formula` parameter, but additional S3 methods are provided for classes `formula`, `lm`, `lmer` and `aov`, which allow our package to integrate well with other linear models functions, as described later in this section.

The prototype of the S3 default method is

```
welchADF.test(formula, response, between.s, within.s = NULL, subject = NULL,
     contrast = c("omnibus", "all.pairwise"), effect = NULL,
     correction = c("hochberg", "holm"), trimming = FALSE, per = 0.2,
     bootstrap = FALSE, numsim_b = 999, effect.size = FALSE, numsim_es = 999,
     scaling = TRUE, standardize.effsz = TRUE, alpha = 0.05, seed = 0, ...)
```

We summarize below the meaning of the arguments; the reader may refer to the package documentation for further detail. Note only the three first arguments are required.

A	B	X	W	Subject	Y_1	Y_2
A_1	B_1	X_1	W_1	1	$(1)Y_1^{A_1B_1X_1W_1}$	$(1)Y_2^{A_1B_1X_1W_1}$
⋮	⋮	⋮	⋮	1	⋮	⋮
A_1	B_1	X_1	W_w	1	$(1)Y_1^{A_1B_1X_1W_w}$	$(1)Y_2^{A_1B_1X_1W_w}$
A_1	B_1	X_2	W_1	1	$(1)Y_1^{A_1B_1X_1W_1}$	$(1)Y_2^{A_1B_1X_1W_1}$
⋮	⋮	⋮	⋮	1	⋮	⋮
A_1	B_1	X_x	W_w	1	$(1)Y_1^{A_1B_1X_xW_w}$	$(1)Y_2^{A_1B_1X_xW_w}$
A_1	B_1	X_1	W_1	2	$(2)Y_1^{A_1B_1X_1W_1}$	$(2)Y_2^{A_1B_1X_1W_1}$
⋮	⋮	⋮	⋮	2	⋮	⋮
A_1	B_1	X_x	W_w	2	$(2)Y_1^{A_1B_1X_xW_w}$	$(2)Y_2^{A_1B_1X_xW_w}$
⋮	⋮	⋮	⋮	⋮	⋮	⋮
A_1	B_1	X_x	W_w	$n_{A_1B_1}$	$(n_{A_1B_1})Y_1^{A_1B_1X_xW_w}$	$(n_{A_1B_1})Y_2^{A_1B_1X_xW_w}$
A_1	B_2	X_1	W_1	$n_{A_1B_1}+1$	$(n_{A_1B_1})Y_1^{A_1B_1X_xW_w}$	$(n_{A_1B_1})Y_2^{A_1B_1X_xW_w}$
⋮	⋮	⋮	⋮	⋮	⋮	⋮
A_a	B_b	X_x	W_x	N	$(N)Y_1^{A_aB_bX_xW_w}$	$(N)Y_2^{A_aB_bX_xW_w}$

Table 1: An example dataset with two between-subjects factors A, B, with a and b different levels, respectively; two within-subjects factors X, W with x and w different levels, and a multivariate response with $p = 2$ correlated response columns Y_1, Y_2. The total number of subjects is N.

- `formula` is a data frame object containing the observations and the level combination to which they correspond. The next four arguments refer to the column names.
- `response,between.s,within.s,subject` are strings (or string vectors) indicating the column names for the response, the between-subjects effect(s), the within-subject(s) effects, and the subject column that stores which subject corresponds to each row (hence it cannot be a vector but a single string). In case the design is multivariate, the response will be a vector of columns, one for each response variable. A sample data frame is displayed in Table 1. Each cell A_iB_j has $n_{A_iB_j}$ subjects, and $n_{A_iB_j} \cdot x \cdot w$ rows in the data frame. Here, $N = \sum_{i,j} n_{A_iB_j}$ subjects.
- `contrast` refers to the type of contrast to be performed. Both in `"omnibus"` and `"all.pairwise"` contrasts, the corresponding contrasts matrices are automatically computed as described in Section 42.2.1.
- `effect` is the effect (i.e. column name) involved in the selected contrast. If `effect` is a vector with length 2 or greater and `contrast = "omnibus"`, then an omnibus contrast on an interaction effect will be tested involving simultaneously all the effects of the vector. If `contrast = "all.pairwise"`, then `effect` must have length 1 or 2 to indicate a single effect or a two-way interaction to which tetrad contrasts will be applied; otherwise an error will be thrown. If left blank, the contrast will be applied separately to all of the existing effects and their interactions.
- The rest of arguments specify whether trimmed means and Winsorized variances will be used and the percentage of trimming (`use.robust.estimators, per`), whether bootstrapping should be used to compute an empirical critical value and how many iterations to do (`use.bootstrap, numsim_b`), and whether the effect size and a confidence interval should be computed (again via bootstrap). Effect size allows the choice of using scaling (`scaling = TRUE`) or not (if not, $\eta = 1$ in Eq. 10) and the number of effect-size bootstrap simulations (`effect.size, scaling, numsim_es, loc1, loc2`).

The aforementioned function is an R wrapper that configures the parameters needed for each type of problem by two private functions, which lie at the core of our package. Both are R translations of the SAS functions `wjglm` and `bootcom` and have been named almost the same. Function `wjglm` is used in all cases (including those in which bootstrapping is needed), except when bootstrap is applied to obtain an empirical critical value for a *family* of contrasts. A modification of function `bootcom` is only invoked in that case in order to control family-wise type I error rate (FWER) via percentile bootstrapping. This scenario arises when performing all pairwise contrasts or tetrad contrasts via bootstrap (i.e. `contrast = "all.pairwise",bootstrap = TRUE`).

The prototype of the S3 method for class `formula` is as usual:

```
welchADF.test(formula,data,subset,...)
```

where `data` is a data frame following the same rules as described above for the `formula` parameter of the default method, `subset` is an indexing vector to indicate which rows of `data` should be used

(all by default), and ... stands for the rest of arguments accepted by `welchADF.test.default` and described above to configure the behavior of the test. As with other models, the terms in `formula` are first sought in `data` and then in the environment of the formula. Note, however, that only between-subjects and within-subjects effects and interactions can be specified together with a Subject column when conducting a WJ test, but no model is fit to the data. For this reason, `formula` should be understood only as a way to indicate the factors involved and their nature (between- or within-subjects) but not as a description of a particular model structure. The presence or absence of an interaction in the formula only affects which effects are tested when `contrast = "omnibus",effect = NULL`; otherwise it does not affect at all. The structure should mirror that of the **lme4** package, e.g.

```
welchADF.test(cbind(visits,time,latency) ~ nurs*tunnel + (tunnel|Subject), miceData)
```

means that there is a multivariate response composed of three correlated variables `visits`, `time`, `latency`, and the design has one between-subjects factor `nurs` (because it appears outside but not inside the parenthesis term) and one within-subjects factor `tunnel` because it appears inside the parenthesis. While a within-subjects effect may appear outside the parenthesis to indicate an interaction with a between-subjects effect, between-subjects must not appear inside the parenthesis.

The function returns an object of class `welchADFt`, which is actually a tagged list of lists, one sub-list per effect in an omnibus contrast, or per category involved a pairwise contrast of a given effect. The call is also stored as the last element of the upper-level list with the name `call`, no matter the S3 method employed (be it `welchADF.test.default`, or the ones for class `formula`, `lm`, `aov` or `lmer`). This allows to implement S3 method `update` for class `welchADFt`, no matter which S3 method was called to calculate the model object [6]. Each sub-list has elements named `welch.T`, `numeratorDF`, `denominatorDF`, `contrast.matrix`, `mean.vector`, `sigma.matrix` which store, respectively, the value of the T_{WJ}/c statistic (or $T^{(t)}_{WJ}/c$ if the trimmed version was used), the approximate degrees of freedom of the numerator and denominator, the contrast matrix $\mathbf{R}$ obtained as $\mathbf{R} = \mathbf{C} \otimes \mathbf{U}^T$, and the estimates $\hat{\boldsymbol{\mu}}$ and $\hat{\boldsymbol{\Sigma}}$. It also stores the user arguments when the function was called and, in case the user asked for the effect size, it provides the effect size along with a confidence interval. Refer to the package documentation for further detail.

The package implements S3 methods `summary`, `format` and `print` for objects of class `welchADFt`, as well as other methods widely used on model objects such as `confint` to get confidence intervals of the effect size (in case the user requested to compute it), `model.frame` to extract the input data frame, and `formula` to extract the formula (not available if the object was generated by `welchADF.test.default`).

Case studies

All the datasets analyzed in this section were mentioned in examples designed by the authors of the SAS implementation (Lix and Keselman, 1995), and have also been included in our R package. For that reason it is not necessary to explicitly read them from text files. They are described in detail in the package documentation.

Univariate one-way between-subject design

The dataset was artificially created by Lix et al. and can be downloaded from her personal website[7]. The data recreate those reported by a real study on perception and concentration, on which 42 students were given several puzzles to be solved. The students are divided into three balanced groups as they had previously been asked to imagine solving puzzles in the distant future, near future, or not to imagine anything at all (control group). The response variable represents the number of puzzles each student was able to solve, out of 12. The data are delivered in our package in a variable named `perceptionData`.

This design presents one between-subjects variable, namely the Group to which the student belongs, and no within-subjects variables as each student is measured on only one response variable and then the student is never measured again. The following R commands demonstrate the types of analyses that can be done with this dataset. The results can be checked on the PDF file of the footnote.

```
> str(perceptionData)
'data.frame':   42 obs. of  2 variables:
 \$ Group: Factor w/ 3 levels "control","distantFuture",..: 2 2 2 2 2 2 2 2 2 2 ...
 \$ y    : int  7 5 8 9 8 8 7 7 6 2 ...
> omnibus_LSM <- welchADF.test(perceptionData, response = "y", between.s = "Group")
```

[6] The update should be done in accordance with the function that generated the `welchADFt` object, i.e. passing a formula is not allowed if the object was generated by the default method, and vice-versa.

[7] http://homepage.usask.ca/~lml321/Example1.pdf

```
> summary(omnibus_LSM, verbose = TRUE)
Call:
   welchADF.test(formula = perceptionData, response = "y", between.s = "Group")

Welch-James Approximate DF Test (Least squares means & variances)
Omnibus test(s) of effect and/or interactions

      WJ statistic Numerator DF Denominator DF Pr(>WJ)
Group        1.795            2          24.16  0.1875
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The omnibus contrast on the between-subjects effect Group determined it is not statistically significant when using least-square means. But if we apply trimming:

```
> omnibus_trimmed <- update(omnibus_LSM, trimming = TRUE)
> omnibus_trimmed_boot <- update(omnibus_trimmed, bootstrap = TRUE, seed = 12345)
> summary(omnibus_trimmed)
Call:
   welchADF.test(formula = perceptionData, response = "y", between.s = "Group",
    trimming = TRUE)

      WJ statistic Numerator DF Denominator DF Pr(>WJ)
Group        4.975            2          16.11 0.02076 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

By applying trimmed means and Winsorized variances we do get a statistically significant result. Hence, since the omnibus test was significant on this factor, we do pairwise comparisons on it in order to test which pairs of group levels make the associated groups of responses statistically different. Since the result was obtained with trimming, we continue with it in pairwise comparisons. Only the result of non-bootstrapped trimming is displayed here.

```
> pairwise_trimmed <- welchADF.test(y ~ Group, data = perceptionData, effect = "Group",
  contrast = "all.pairwise", trimming = TRUE, effect.size = TRUE)
> pairwise_trimmed_boot <- update(pairwise_trimmed, bootstrap = TRUE, seed = 12345)
> summary(pairwise_trimmed)
Call:
   welchADF.test(formula = y ~ Group, data = perceptionData, effect = "Group",
    contrast = "all.pairwise", trimming = TRUE, effect.size = TRUE)

                        WJ statistic Numerator DF Denominator DF eff.size adj.pval
control:nearFuture          0.004398            1         10.089 -0.02674   0.9484
distantFuture:nearFuture    0.876366            1          9.778  0.38620   0.7435
control:distantFuture      10.088968            1         17.510 -1.09981   0.0161 *
---
Signif. codes (Hochberg p-values):  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The output shows that the responses associated to the control group significantly differs from those associated to the distantFuture group. The confidence intervals on the effect size can be retrieved as

```
> confint(pairwise_trimmed)
                           2.5 %   97.5 %
control:nearFuture       -0.8208  0.85290
distantFuture:nearFuture -0.3402  1.74365
control:distantFuture    -2.2571 -0.09678
```

An important issue arises in this example that justifies again the use of trimmed estimators. As can be seen in the omnibus tests, the Group is not significant with Least-squares means but it is when we use trimmed means and Winsorized variances. This yields a significant result which could not be detected unless trimming is applied. As the result of the omnibus test with trimmed means is significant, we proceed to the pairwise comparisons using trimming as well. This yields that `control` and `distantFuture` have associated significantly different values of the number of puzzles solved by the students on average.

Two-way factorial (between-subjects) design

Once again, this dataset[8] was artificially created by Lix et al. Quoting from the PDF, the author used summary data presented by Wicherts et al. (2005). These authors examined the effects of stereotype threat on women's mathematics ability. Study participants were assigned to one of six groups defined by crossing the independent factors of test condition (control, nullified, stereotype threat) and sex (male, female). Originally there were four different tests administered to study participants (arithmetic, number series, word problems, and sums tests) the dataset contains only scores for the arithmetic test (out of 40) because these scores exhibited a greater magnitude of variance heterogeneity than scores for the other tests. It is an unbalanced design with cell sizes ranging from 45 to 50 participants, and a total sample size of 283. The data are delivered in our package in a variable named `womenStereotypeData`.

The output of the omnibus tests using robust estimators (trimmed means and Winsorized variances) with and without bootstrapping is shown in first place. Since the interaction between "condition" and "sex" is significant according to trimmed means, the post-hoc pairwise comparisons (tetrad contrasts) are shown using trimmed means with and without bootstrapping. The results match those presented in pages 5 and 6 of the PDF.

```
> omnibus_LSM <- welchADF.test(womenStereotypeData, response = "y", between.s =
  c("condition", "sex"), contrast = "omnibus")
> omnibus_trimmed <- update(omnibus_LSM, trimming = TRUE)
> summary(omnibus_LSM)
Call:
   welchADF.test(formula = womenStereotypeData, response = "y",
    between.s = c("condition", "sex"), contrast = "omnibus")

              WJ statistic Numerator DF Denominator DF Pr(>WJ)
condition            2.151            2          154.7 0.11986
sex                  2.933            1          216.4 0.08824 .
condition:sex        2.521            2          154.7 0.08368 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

> summary(omnibus_trimmed)
Call:
   welchADF.test(formula = womenStereotypeData, response = "y",
    between.s = c("condition", "sex"), contrast = "omnibus",
    trimming = TRUE)

              WJ statistic Numerator DF Denominator DF  Pr(>WJ)
condition            5.205            2          93.38 0.007189 **
sex                  5.754            1         130.06 0.017875  *
condition:sex        3.130            2          93.38 0.048347  *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

In this case, the omnibus test is unclear when using least-square means (p-values only slightly greater than the common significance threshold of 0.05 according to the summary of `omnibus_LSM`) but trimming helps make things clearer. This results in both factors and their interaction being statistically significant at a significance level of 0.05 (see the summary of `omnibus_trimmed` above).

Since the omnibus test confirms the significance of all effects, pairwise comparisons should be done on all of them. Due to space constraints, only the two-way `condition:sex` interaction effect was probed (as `effect = c("condition","sex")` in the call to create the `pairwise_LSM` object that was subsequently updated to account for trimming and bootstrapping). Pairwise contrasts on two-way interactions are also known as tetrad contrasts.

```
> pairwise_trimmed <- welchADF.test(y ~ condition*sex, data = womenStereotypeData,
  contrast = "all.pairwise", effect = c("condition", "sex"), trimming = TRUE)
> pairwise_trimmed_boot <- update(pairwise_trimmed, bootstrap = TRUE, seed = 12345)

> summary(pairwise_trimmed_boot, verbose = TRUE)
Call:
   welchADF.test(formula = y ~ condition * sex, data = womenStereotypeData,
```

[8] http://homepage.usask.ca/~lml321/Example2.pdf

```
    contrast = "all.pairwise", effect = c("condition", "sex"),
    trimming = TRUE, bootstrap = TRUE, seed = 12345)

Welch-James Approximate DF Test (Trimmed means [20% trimming] & Winsorized variances)
Multiple tetrad interaction contrasts with respect to condition x sex interaction
using a Bootstrap Critical Value for FWER control
                                WJ statistic Numerator DF Denominator DF significant?
control:stereotype x female:male       0.4662           1          97.49          no
nullified:stereotype x female:male     1.9846           1          79.63          no
control:nullified x female:male        5.7662           1          88.55         yes

Bootstrap critical value: 5.145
```

Pairwise comparisons on the condition:sex interaction with trimmed bootstrapped means reveal only one significant interaction between the pairs of levels `control:nullified` and `female:male`.

Multivariate (mixed) between- × within-subjects design

The problem and the data are described in Keselman et al. (2003). The data represent the reaction times in milliseconds of children with attention-deficit hyperactivity (ADHD) and normal children when they are presented four kinds of inputs: a target alone or an arrow stimuli incongruent, congruent and neutral to the target. According to the authors, the dataset was artificially generated from the summary measures given in the original study by Jonkman et al. (1999), in groups of 20 and 10 children to create an unbalanced design. The data are delivered in our package in two variables named `adhdData` and `adhdData2`.

One-way multivariate vs univariate mixed model This problem can be approached in two different ways: (a) as a one-way multivariate design, which would be the non-parametric equivalent of MANOVA (multivariate ANOVA), or (b) as a univariate mixed model having one between-subjects factor (the student's group) and one within-subjects factor (with four levels, namely the four stimuli measured in every single student). In case we were analysing the data under parametric assumptions, the second option requires sphericity while MANOVA does not (although it needs more data). On the other hand, mixed models are able to capture the covariance structure of the dependent variables and can be generalized to any number of factors. An in-depth discussion on this topic can be found in chapter 9 of Maxwell and Delaney (2004). Keselman et al. (2003) (page 593) insist on using trimmed means and/or bootstrapping with this kind of models in order to overcome deviations from sphericity.

Our package admits both types of analysis. When dealing with a one-way multivariate design, the data must be formated as in Figure 1(a), while a mixed model requires the more systematic format of Figure 1(b) which is valid for an arbitrary amount of factors of both types. As done in their paper, we will analyze this dataset as a mixed model in which the stimuli are an explicit within-subjects factor.

Implicit within-subjects effect The package admits a third way to indicate the within-subjects effects that simplifies its use. It is common to have a dataset with a within-subjects effect expressed in the form of Figure 1(a). In this case, we may want to consider the within-subjects effect underlying the multivariate response. Reshaping this file to match the structure of Figure 1(b) would require some effort by the user. To avoid this, the function allows indicating that the multivariate response is actually an implicit within-subjects effects by including the word `"multivariate"` in the vector of within-subjects column names (if this argument was empty, then we just set the argument `within.s = "multivariate"`). This can be generalized as follows: if we have K within-subjects effects, we can have $K-1$ columns in the data with their explicit names and levels, and leave one effect to be indicated in the multivariate response. In that case, we set `within.s = c("within1", "within2", ..., "within-k-1", "multivariate")` and pass a multivariate response vector argument because the data must have one response column per level of the K-th within-subjects effect. In the code below, the variables with the termination `_multi` show the equivalent calls. Unless we change the model itself (i.e. consider a mixed model or a multivariate one-way model with no within-subjects factor), the results obtained are the same in all types of analyses (omnibus, pairwise, etc), no matter the structure of the input data file. We demonstrate all the possibilities below.

```
> omnibus_LSM_mixed_implicit <- welchADF.test(adhdData, response = c("TargetAlone",
  "Congruent", "Neutral", "Incongruent"), within.s = "multivariate", between.s = "Group",
  contrast = "omnibus")
> omnibus_LSM_multi_oneway <- welchADF.test(cbind(TargetAlone, Congruent, Neutral,
```

Group	TargetAlone	Incongruent	Congruent	Neutral
Normal	568.52	433.80	658.51	711.33
Normal	1034.82	864.79	639.42	815.18
⋮	⋮	⋮	⋮	
ADHD	707.15	872.39	645.83	677.84

(a) As a one-way multivariate model, stored in variable `adhdData`

Group	Stimulus	Subject	Millisec
Normal	TargetAlone	1	568.52
Normal	Incongruent	1	433.80
Normal	Congruent	1	658.51
Normal	Neutral	1	711.33
⋮	⋮	⋮	
ADHD	TargetAlone	30	707.15
ADHD	Incongruent	30	872.39
ADHD	Congruent	30	645.83
ADHD	Neutral	30	677.84

(b) A a mixed model with one between- $\times$ one within-subjects factor, as in `adhdData2`

Figure 1: Two alternative ways of arranging the ADHD data input file (*wide* vs *long* format).

```
  Incongruent) ~ Group, data = adhdData)
> omnibus_LSM_mixed <- welchADF.test(adhdData2, response = "Milliseconds",
  between.s = "Group", within.s = "Stimulus", subject = "Subject", contrast = "omnibus")
> omnibus_LSM_mixed_formula <- welchADF.test(Milliseconds ~ Group*Stimulus +
  (Stimulus|Subject), data = adhdData2)
> omnibus_trimmed_formula <- update(omnibus_LSM_mixed_formula, trimming = TRUE)
> omnibus_trimmed_boot <- update(omnibus_trimmed_formula, bootstrap = TRUE, seed = 12345)
```

Above we have demonstrated the possibilities of an omnibus contrast to both arrangements of the data. The first and third models assume a mixed model, where the within-subjects factor is implicit in `omnibus_LSM_mixed_implicit` (using `adhdData`) and explicit in `omnibus_LSM_mixed` (using `adhdData2`). The second model assumes a multivariate one-way model with no within-subjects effects. The model `omnibus_LSM_mixed_formula` assumes an explicit mixed model described by a formula. The formula interface can be fitted only to data in long format like `adhdData2`. Finally, this model is updated to include trimming, and the resulting updated model is updated again to include bootstrapping as well.

```
> summary(omnibus_LSM_mixed_implicit)
Call:
   welchADF.test(formula = adhdData, response = c("TargetAlone",
    "Congruent", "Neutral", "Incongruent"), between.s = "Group",
    within.s = "multivariate", contrast = "omnibus")

                    WJ statistic Numerator DF Denominator DF  Pr(>WJ)
Group                     0.2249            1          24.84 0.639482
multivariate              5.6591            3          21.02 0.005282 **
Group : multivariate      0.5750            3          21.02 0.637759
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

> summary(omnibus_LSM_multi_oneway)
Call:
   welchADF.test(formula = cbind(TargetAlone, Congruent, Neutral,
    Incongruent) ~ Group, data = adhdData)

      WJ statistic Numerator DF Denominator DF Pr(>WJ)
Group       0.4227            4          20.52  0.7904
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The results match those presented in Keselman et al. (2003), page 594. The only significant effect is the Stimulus, which acts as the within-subjects factor. When we consider a one-way multivariate design, with no within-subjects factor, then the Group (the between-subjects factor) is deemed not significant. In order perform all pairwise comparisons between the levels of Stimulus, we proceed as follows (only the bootstrap results are shown due to space constraints). The comparison is statistically significant when the value of the WJ statistic is greater than or equal to the bootstrap critical value. In this case, there were two significant differences, namely Incongruent vs TargetAlone, and Incongruent vs Congruent, as mentioned in page 595 of the aforementioned work.

```
> pairwise_trimmed_formula <- update(omnibus_trimmed_formula, contrast = "all.pairwise",
  effect = "Stimulus")
> pairwise_trimmed_formula_boot <- update(pairwise_trimmed_formula, bootstrap = TRUE,
  seed = 123456)

> summary(pairwise_trimmed_formula_boot)
Call:
   welchADF.test(formula = Milliseconds ~ Group * Stimulus + (Stimulus |
    Subject), data = adhdData2, trimming = TRUE, contrast = "all.pairwise",
    effect = "Stimulus", bootstrap = TRUE, seed = 123456)

                       WJ statistic Numerator DF Denominator DF significant?
Congruent:TargetAlone        3.3278            1         15.515           no
Incongruent:TargetAlone     17.3549            1         15.436          yes
Neutral:TargetAlone          0.8251            1          8.419           no
Congruent:Neutral            0.1818            1          8.852           no
Incongruent:Neutral         13.9212            1         15.305          yes
Congruent:Incongruent        8.0013            1         15.503           no

Bootstrap critical value: 8.577
```

Note that, when using the implicit within-subjects effect format, we have to specify `effect = "multivariate"` to indicate that the effect to be tested is the within-subjects effect, even though there is no column with such name in our data. In case we are using the formula interface, this is not available because formula terms must strictly correspond to column names or variables in the environment of the formula.

Multivariate within-subjects design (doubly multivariate)

The three case studies addressed before are probably the most common in practice. Nevertheless, and with the aim of demonstrating the flexibility of the **welchADF** package, we present here an additional, not-so-common setting, namely a doubly multivariate design also proposed by Lix and Keselman (1995). In general, this design arises when several dependent variables are measured for each individual at several time points (every variable measured at each time point), or under different conditions; the latter is the case of the example addressed below.

We could not access the data employed by Lix, hence we have analyzed data from Wuensch (1992) which are freely available in the author's website[9]. In this experiment, wild strain house mice were, at birth, cross fostered onto house mouse (Mus), deer mouse (Peromyscus) or rat (Rattus) nursing mothers. Ten days after weaning, each subject was tested in an apparatus that allowed it to enter four different tunnels: one scented with clean pine shavings, and the other three tunnels with shavings bearing the scent of Mus, Peromyscus, or Rattus respectively. Three variables were measured for each tunnel: the number of visits to the tunnel during a twenty minute test, the time spent by each subject in each of the four tunnels and the latency to first visit of each tunnel.

In this design, the type of nursing mother is a between-subjects factor. The within-subjects factor is scent, with four levels (clean, Mus, Peromyscus, and Rattus). The multivariate response is composed of visits, time, and latency for each tunnel. With this approach, the multivariate response is not treated as another within-subjects factor. The data are delivered in our package in a variable named `miceData`.

```
> head(miceData)
  Subject nurs       tunnel visits   time latency
1       1  Mus        Clean      6 721.35  207.90
2       1  Mus        MusSc      4 318.15   26.78
3       1  Mus PeromyscusSc      2  48.83 1025.33
4       1  Mus     RattusSc      0   0.00 1212.75
5       2  Mus        Clean      8 119.70  685.13
6       2  Mus        MusSc      7 207.90  113.40
```

We first do an omnibus contrast. In the second call we demonstrate how the formula interface can be used in this design to obtain exactly the same result.

```
> omnibus_LSM <- welchADF.test(miceData, response = c("visits", "time", "latency"),
  between.s = "nurs", within.s = "tunnel", subject = "Subject", contrast = "omnibus")
```

[9] http://core.ecu.edu/psyc/wuenschk/SPSS/TUNNEL4b.sav

```
> omnibus_LSM_formula <- welchADF.test(cbind(visits, time, latency) ~ nurs*tunnel +
  (tunnel | Subject), data = miceData)

> summary(omnibus_LSM_formula)
Call:
   welchADF.test(formula = cbind(visits, time, latency) ~ nurs *
    tunnel + (tunnel | Subject), data = miceData)

             WJ statistic Numerator DF Denominator DF   Pr(>WJ)
nurs                4.008            6          21.46 0.0076171  **
tunnel              5.201            9          22.08 0.0007601 ***
nurs : tunnel       5.153           18          21.38 0.0002407 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The least-square means were able to deem all effects statistically significant. Therefore we move to pairwise contrasts for each of the effects, with and without trimming. The pairwise results of the interaction term are not displayed as they consist of a large table.

```
> pairwise_LSM_nurs <- update(omnibus_LSM_formula, effect = "nurs",
  contrast = "all.pairwise")
> pairwise_LSM_tunnel <- update(pairwise_LSM_nurs, effect = "tunnel")
> pairwise_tunnel_trimmed <- update(pairwise_LSM_tunnel, trimming = TRUE)
> pairwise_nurs_trimmed <- update(pairwise_LSM_nurs, trimming = TRUE)

> summary(pairwise_LSM_nurs)
Call:
   welchADF.test(formula = cbind(visits, time, latency) ~ nurs *
    tunnel + (tunnel | Subject), data = miceData, effect = "nurs",
    contrast = "all.pairwise")

                 WJ statistic Numerator DF Denominator DF adj.pval
Mus:Rattus              4.210            3          16.85  0.04287 *
Peromyscus:Rattus       6.141            3          17.34  0.01468 *
Mus:Peromyscus          1.255            3          17.44  0.32030
---
Signif. codes (Hochberg p-values):  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

> summary(pairwise_LSM_tunnel)
Call:
   welchADF.test(formula = cbind(visits, time, latency) ~ nurs *
    tunnel + (tunnel | Subject), data = miceData, effect = "tunnel",
    contrast = "all.pairwise")

                     WJ statistic Numerator DF Denominator DF adj.pval
Clean:RattusSc             0.9554            3          24.71  0.42925
MusSc:RattusSc             6.7816            3          23.36  0.01129 *
PeromyscusSc:RattusSc      2.4812            3          24.74  0.33190
Clean:PeromyscusSc         1.2393            3          22.83  0.42925
MusSc:PeromyscusSc         2.2319            3          23.84  0.33190
Clean:MusSc                3.0873            3          24.97  0.22712
---
Signif. codes (Hochberg p-values):  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

We will now incorporate bootstrapping to the pairwise comparisons and check the results:

```
> pairwise_nurs_trimmed_boot <- update(pairwise_nurs_trimmed, bootstrap = TRUE, seed = 123)
> pairwise_tunnel_trimmed_boot <- update(pairwise_nurs_trimmed_boot, effect = "tunnel")

> summary(pairwise_nurs_trimmed_boot)
Call:
   welchADF.test(formula = cbind(visits, time, latency) ~ nurs *
    tunnel + (tunnel | Subject), data = miceData, effect = "nurs",
    contrast = "all.pairwise", trimming = TRUE, bootstrap = TRUE,
    seed = 123)
```

```
                 WJ statistic Numerator DF Denominator DF significant?
Mus:Rattus             3.6147            3         10.662           no
Peromyscus:Rattus      6.3409            3          9.551          yes
Mus:Peromyscus         0.6982            3         10.438           no

Bootstrap critical value: 6.292

> summary(pairwise_tunnel_trimmed_boot)
Call:
   welchADF.test(formula = cbind(visits, time, latency) ~ nurs *
    tunnel + (tunnel | Subject), data = miceData, effect = "tunnel",
    contrast = "all.pairwise", trimming = TRUE, bootstrap = TRUE,
    seed = 123)

                     WJ statistic Numerator DF Denominator DF significant?
Clean:RattusSc              1.544            3          15.76           no
MusSc:RattusSc              6.053            3          15.20          yes
PeromyscusSc:RattusSc       3.729            3          15.03           no
Clean:PeromyscusSc          3.106            3          14.26           no
MusSc:PeromyscusSc          1.976            3          15.50           no
Clean:MusSc                 4.842            3          14.12           no

Bootstrap critical value: 5.922
```

The pairwise comparisons using least-squares means are able to detect significant differences only between MusSc and RattusSc in the tunnel effect, and the trimmed-bootstrapped comparison also supports this fact and negates significant differences for any other pair of levels. Regarding the nurs effect, the LSM comparison reveals significant differences in Mus vs Rattus and also in Peromyscus vs Rattus, but the trimmed-bootstrapped only agrees with the latest.

Conclusions and further work

This contribution has demonstrated the applicability of the new **welchADF** package in a variety of experimental designs, ranging from the most simple one, namely a univariate one-way between-subjects design, to a more exotic one like a doubly-multivariate design. The unified approach of Johansen (1980) that has been implemented here leads to a great ease of use for any case study. We have shown in the example code that specifying the factors involved in the design and the type of analysis are done in a straightforward way, and then the code automatically generates the contrast matrices needed and runs the test, no matter how complex the user's design is. Therefore, researchers from other areas may find it more friendly and hence, our effort may contribute to the diffusion of the Welch-James ADF test in applied studies.

In the future, an enhancement may be added so that custom contrasts can be done in addition to the most common omnibus and pairwise contrasts. This requires designing a simple, yet powerful mechanism for the user to describe the desired test in the function arguments.

Acknowledgments

The author wants to thank the editor and two anonymous reviewers for their useful comments and code snippets which have greatly contributed to improve the quality of the package and its integration within the R package ecosystem, and the readability of the manuscript.

Bibliography

Publication Manual of the American Psychological Association. American Psychological Association, 6th edition, 2013. [p315]

J. Algina. Generalization of Improved General Approximation tests to split-plot designs with multiple between-subjects factors and/or multiple within-subjects factors. *British Journal of Mathematical and Statistical Psychology*, 50:243–252, 1997. URL https://doi.org/10.1111/j.2044-8317.1997.tb01144.x. [p310, 311]

J. Algina and S. F. Olejnik. Implementing the welch-james procedure with factorial designs. *Educational and Psychological Measurement*, 44(1):39–48, 1984. URL https://doi.org/10.1177/0013164484441004. [p313]

J. M. Aronoff, D. J. Freed, L. M. Fisher, I. Pal, and S. D. Soli. The effect of different cochlear implant microphones on acoustic hearing individuals' binaural benefits for speech perception in noise. *Ear Hear*, 32(4):468–484, 2011. URL https://doi.org/10.1097/AUD.0b013e31820dd3f0. [p311]

D. Bates, M. Mächler, B. Bolker, and S. Walker. Fitting Linear Mixed-Effects Models Using lme4. *Journal of Statistical Software*, 67(1):1–48, 2015. URL https://doi.org/10.18637/jss.v067.i01. [p311]

D. Bates, M. Maechler, B. Bolker, and S. Walker. *lme4: Linear Mixed-Effects Models using 'Eigen' and S4*, 2016. URL https://CRAN.R-project.org/package=lme4. R package version 1.1-12. [p311]

T. M. Beasley. Multivariate Aligned Rank Test for Interactions in Multiple Group Repeated Measures Designs. *Multivariate Behavioral Research*, 37(2):197 – 226, 2002. URL https://doi.org/10.1207/S15327906MBR3702_02. [p310]

B. M. Bolker, M. E. Brooks, C. J. Clark, S. W. Geange, J. R. Poulsen, M. H. H. Stevens, and J.-S. S. White. Generalized linear mixed models: a practical guide for ecology and evolution. *Trends in Ecology & Evolution*, 24(3):127–135, 2009. URL https://doi.org/10.1016/j.tree.2008.10.008. [p309]

J. Coates and S. McKenzie-Mohr. Out of the Frying Pan, Into the Fire: Trauma in the Lives of Homeless Youth Prior to and During Homelessness. *Journal of Sociology & Social Welfare*, 37(4):65–96, 2010. [p309]

W. J. Conover. The rank transformation - an easy and intuitive way to connect many nonparametric methods to their parametric counterparts for seamless teaching introductory statistics courses. *Wiley Interdisciplinary Reviews: Computational Statistics*, 4(5):432–438, 2012. URL https://doi.org/10.1002/wics.1216. [p310]

J. Dien. The ERP PCA Toolkit: An open source program for advanced statistical analysis of event-related potential data. *Journal of Neuroscience Methods*, 187(1):138–145, 2010. URL https://doi.org/10.1016/j.jneumeth.2009.12.009. [p311]

J. Dien, M. S. Franklin, C. A. Michelson, L. C. Lemene, C. L. Adams, and K. A. Kiehlf. fMRI characterization of the language formulation area. *Brain Research*, 1229:179–192, 2008. URL https://doi.org/10.1016/j.brainres.2008.06.107. [p311]

D. M. Erce-Hurn and V. M. Mirosevich. Modern Robust Statistical Methods: An Easy Way to Maximize the Accuracy and Power of Your Research. *American Psychologist*, 63(7):591–601, 2008. URL https://doi.org/10.1037/0003-066X.63.7.591. [p310]

D. A. Fournier, H. J. Skaug, J. Ancheta, J. Ianelli, A. Magnusson, M. Maunder, A. Nielsen, and J. Sibert. AD Model Builder: using automatic differentiation for statistical inference of highly parameterized complex nonlinear models. *Optimization Methods and Software*, 27(2):233–249, 2012. URL https://doi.org/10.1080/10556788.2011.597854. [p311]

J. J. Higgins. *Introduction to Modern Nonparametric Statistics*. Cengage Learning, 2003. [p309]

J. J. Higgins and S. Tashtoush. An aligned rank transform test for interaction. *Nonlinear World*, 1: 201–221, 1994. [p310]

B. H. Huang and S.-A. Jun. Age matters, and so may ratters: Rater Differences in the Assessment of Foreign Accents. *Studies in Second Language Acquisition*, 37(4):623 – 650, 2015. URL https://doi.org/10.1017/S0272263114000576. [p311]

H. Huynh. Some approximate test for repeated measurement designs. *Psychometrika*, 43(2):161–175, 1978. URL https://doi.org/10.1007/BF02293860. [p309, 310]

S. Johansen. The Welch-James approximation to the distribution of the residual sum of squares in a weighted linear regression. *Biometrika*, 67:85–92, 1980. URL https://doi.org/10.2307/2335320. [p310, 313, 324]

L. Jonkman, C. Kemner, M. Verbaten, H. van Engeland, J. Kenemans, G. Camfferman, J. Buitelaar, and H. Koelega. Perceptual and response interference in children with attention-deficit hyperactivity disorder, and the effects of methylphenidate. *Psychophysiology*, 36(4):419 – 429, 1999. URL https://doi.org/10.1111/1469-8986.3640419. [p320]

J. Kayser, C. E. Tenke, C. J. Kroppmann, D. M. Alschuler, S. Fekri, S. Ben-David, C. M. Corcoran, and G. E. Bruder. Auditory event-related potentials and α oscillations in the psychosis prodrome: neuronal generator patterns during a novelty oddball task. *International Journal of Psychophysiology*, 91(2):104–120, 2014. URL https://doi.org/10.1016/j.ijpsycho.2013.12.003. [p311]

H. J. Keselman, R. K. Kowalchuk, J. Algina, L. M. Lix, and R. R. Wilcox. Testing treatment effects in repeated measures designs: Trimmed means and bootstrapping. *British Journal of Mathematical and Statistical Psychology*, 53(2):175–191, 2000. URL https://doi.org/10.1348/000711000159286. [p312, 315]

H. J. Keselman, R. R. Wilcox, and L. M. Lix. A generally robust approach to hypothesis testing in independent and correlated groups designs. *Psychophyisiology*, 40:586–596, 2003. URL https://doi.org/10.1111/1469-8986.00060. [p310, 311, 312, 315, 320, 321]

H. J. Keselman, J. Algina, L. M. Lix, R. R. Wilcox, and K. N. Deering. A generally robust approach for testing hypotheses and setting confidence intervals for effect sizes. *Psychological Methods*, 13(2): 110–129, 2008. URL https://doi.org/10.1037/1082-989X.13.2.110. [p311, 315]

M. Koller. *Robust Estimation of Linear Mixed Models*. PhD thesis, ETH Zurich, 2013. URL http://e-collection.library.ethz.ch/eserv/eth:6670/eth-6670-02.pdf. [p311]

M. Koller. robustlmm: An R Package for Robust Estimation of Linear Mixed-Effects Models. *Journal of Statistical Software*, 75(6):1–24, 2016. URL https://doi.org/10.18637/jss.v075.i06. [p311]

M. Koller. *robustlmm: Robust Linear Mixed Effects Models*, 2017. URL https://CRAN.R-project.org/package=robustlmm. R package version 2.1-3. [p311]

J. R. Levin. Overcoming feelings of powerlessness in "aging" researchers: a primer on statistical power in analysis of variance designs. *Psychology and aging*, 12(1):84–106, 1997. URL https://doi.org/10.1037/0882-7974.12.1.84. [p309]

L. M. Lix and H. J. Keselman. Approximate Degrees of Freedom Tests: A Unified Perspective on Testing for Mean Equality. *Psychological Bulletin*, 117(3):547–560, 1995. URL https://doi.org/10.1037/0033-2909.117.3.547. [p309, 310, 311, 312, 317, 322]

M. Maechler, P. Rousseeuw, C. Croux, V. Todorov, A. Ruckstuhl, M. Salibian-Barrera, T. Verbeke, M. Koller, E. L. T. Conceicao, and M. Anna di Palma. *robustbase: Basic Robust Statistics*, 2016. URL http://robustbase.r-forge.r-project.org/. R package version 0.92-7. [p310]

P. Mair and R. Wilcox. *WRS2: A Collection of Robust Statistical Methods*, 2017. URL https://CRAN.R-project.org/package=WRS2. R package version 0.9-2. [p310]

R. A. Maronna, D. R. Martin, and V. J. Yohai. *Robust Statistics: Theory and Methods*. Wiley, 2006. [p310]

S. E. Maxwell and H. D. Delaney. *Designing Experiments and Analyzing Data: A model comparison perspective, 2nd Ed.* Routledge, 2004. [p320]

G. W. Milligan, D. S. Wong, and P. A. Thompson. Robustness properties of nonorthogonal analysis of variance. *Psychological Bulletin*, 101(3):464–470, 1987. URL https://doi.org/10.1037/0033-2909.101.3.464. [p309]

U. C. Müller, P. Asherson, T. Banaschewski, J. K. Buitelaar, R. P. Ebstein, J. Eisenberg, M. Gill, I. Manor, A. Miranda, R. D. Oades, H. Roeyers, A. Rothenberger, J. A. Sergeant, E. J. Sonuga-Barke, M. Thompson, S. V. Faraone, and H.-C. Steinhausen. The impact of study design and diagnostic approach in a large multi-centre ADHD study. Part 1: ADHD symptom patterns. *BMC Psychiatry*, 11(54), 2011. URL https://doi.org/10.1186/1471-244X-11-54. [p311]

J. Pinheiro, D. Bates, and R-core. *nlme: Linear and Nonlinear Mixed Effects Models*, 2017. URL https://CRAN.R-project.org/package=nlme. R package version 3.1-131. [p311]

J. Ruscio and B. Roche. Variance heterogeneity in published psychological research: a review and a new index. *Methodology: European Journal of Research Methods for the Behavioral and Social Sciences*, 8 (1):1–11, 2012. URL https://doi.org/10.1027/1614-2241/a000034. [p310]

M. J. V. Ryzin, E. A. Carlson, and L. A. Sroufe. Attachment discontinuity in a high-risk sample. *Attachment & Human Development*, 13(4):381–401, 2011. URL https://doi.org/10.1080/14616734.2011.584403. [p311]

M. I. Salazar-Alvarez, V. G. Tercero-Gómez, M. del Carmen Temblador-Pérez, A. E. Cordero-Franco, and W. J. Conover. Nonparametric analysis of interactions: a review and gap analysis. In *Proceedings of the 2014 Industrial and Systems Engineering Research Conference*, 2014. [p310]

M. Salibián-Barrera, S. Van Aelst, and G. Willems. Fast and robust bootstrap. *Statistical Methods and Applications*, 17(1):41–71, 2008. URL https://doi.org/10.1007/s10260-007-0048-6. [p312]

SAS Institute. *SAS/STAT 9.3 User's guide*. 2011. [p311]

S. S. Sawilowsky. Nonparametric Tests of Interaction in Experimental Design. *Review of Educational Research*, 60(1):91–126, 1990. URL https://doi.org/10.3102/00346543060001091. [p310]

K. Singh. Breakdown theory for bootstrap quantiles. *The Annals of Statistics*, 26(5):1719–1732, 10 1998. URL https://doi.org/10.1214/aos/1024691354. [p312]

H. Skaug, D. Fournier, A. Nielsen, A. Magnusson, and B. Bolker. *glmmADMB: Generalized Linear Mixed Models using 'AD Model Builder'*, 2016. http://glmmadmb.r-forge.r-project.org, http://admb-project.org. [p311]

L. Symes, J. McFarlane, L. Frazier, M. C. Henderson-Everhardus, G. McGlory, K. B. Watson, Y. Liu, C. E. Rhodes, and R. C. Hoogeveen. Exploring violence against women and adverse health outcomes in middle age to promote women's health. *Critical Care Nursing Quarterly*, 33(3):233–243, 2010. URL https://doi.org/10.1097/CNQ.0b013e3181e6d7c4. [p311]

G. Vallejo and P. Livacic-Rojas. Comparison of Two Procedures for Analyzing Small Sets of Repeated Measures Data. *Multivariate Behavioral Research*, 40(2):179–205, 2005. URL https://doi.org/10.1207/s15327906mbr4002_2. [p310]

G. Vallejo, A. Fidalgo, and P. Fernández. Effects of covariance heterogeneity on three procedures for analyzing multivariate repeated measures designs. *Multivariate Behavioral Research*, 36(1):1–27, 2001. URL https://doi.org/10.1207/S15327906MBR3601_01. [p310]

G. Vallejo, J. Morisa, and N. M. Conejo. A SAS/IML program for implementing the modified Brown-Forsythe procedure in repeated measures designs. *Computer Methods and Programs in Biomedicine*, 83 (3):169–177, 2006. URL https://doi.org/10.1016/j.cmpb.2006.06.006. [p310, 311]

G. Vallejo, M. Ato, M. P. Fernández, and L.-R. P. E. A Practical Method for Analyzing Factorial Designs with Heteroscedastic Data. *Psychological Reports*, 102(3):643–656, 2008. URL https://doi.org/10.2466/pr0.102.3.643-656. [p310]

P. J. Villacorta. *ART: Aligned Rank Transform for Nonparametric Factorial Analysis*, 2015. URL https://CRAN.R-project.org/package=ART. R package version 1.0. [p310]

P. J. Villacorta. *welchADF: Welch-James Statistic for Robust Hypothesis Testing under Heterocedasticity and Non-Normality*, 2017. URL https://CRAN.R-project.org/package=welchADF. R package version 0.2. [p311]

J. Wang, R. Zamar, A. Marazzi, V. Yohai, M. Salibian-Barrera, R. Maronna, E. Zivot, D. Rocke, D. Martin, M. Maechler, and K. Konis. *robust: Port of the S+ "Robust Library"*, 2017. URL https://CRAN.R-project.org/package=robust. R package version 0.4-18. [p310]

B. L. Welch. On the comparison of several mean values: an alternative approach. *Biometrika*, 38: 330–336, 1951. URL https://doi.org/10.2307/2332579. [p310]

J. Wicherts, C. Dolan, and D. Hessen. Stereotype threat and group differences in test performance: a question of measurement invariance. *Journal of Personality and Social Psychology*, 89(5):696–716, 2005. URL https://doi.org/10.1037/0022-3514.89.5.696. [p319]

R. Wilcox. *Introduction to Robust Estimation & Hypothesis Testing, 3rd Ed.* Academic Press, 2012. [p310]

R. R. Wilcox, H. J. Keselman, and R. K. Kowalchuk. Can tests for treatment group equality be improved?: The bootstrap and trimmed means conjecture. *British Journal of Mathematical and Statistical Psychology*, 51:123–134, 1998. URL https://doi.org/10.1111/j.2044-8317.1998.tb00670.x. [p311]

S. Wood. *mgcv: Mixed GAM Computation Vehicle with Automatic Smoothness Estimation*, 2017. URL https://CRAN.R-project.org/package=mgcv. R package version 1.8-19. [p312]

S. Wood and F. Scheipl. *gamm4: Generalized Additive Mixed Models using 'mgcv' and 'lme4'*, 2017. URL https://CRAN.R-project.org/package=gamm4. R package version 0.2-5. [p312]

K. L. Wuensch. Fostering house mice onto rats and deer mice: Effects on response to species odors. *Animal Learning & Behavior*, 20(3):253 – 258, 1992. URL https://doi.org/10.3758/BF03213379. [p322]

Pablo J. Villacorta
Department of Computer Science and Artificial Intelligence, University of Granada
ETSIIT, C/Periodista Daniel Saucedo Aranda s/n, 18071 Granada, Spain
pjvi@decsai.ugr.es

ider: Intrinsic Dimension Estimation with R

by Hideitsu Hino

Abstract In many data analyses, the dimensionality of the observed data is high while its intrinsic dimension remains quite low. Estimating the intrinsic dimension of an observed dataset is an essential preliminary step for dimensionality reduction, manifold learning, and visualization. This paper introduces an R package, named **ider**, that implements eight intrinsic dimension estimation methods, including a recently proposed method based on a second-order expansion of a probability mass function and a generalized linear model. The usage of each function in the package is explained with datasets generated using a function that is also included in the package.

Introduction

An assumption that the intrinsic dimension is low even when the apparent dimension is high—that the data distribution is constrained onto a low dimensional manifold—is the basis of many machine learning and data analysis methods, such as dimension reduction and visualization (Cook and Yin, 2001; Kokiopoulou and Saad, 2007). Without good estimates of the intrinsic dimension, dimensionality reduction is no more than a risky bet, insofar as one does not know to what extent the dimensionality can be reduced. We may overlook important information by projecting the original data on too small dimensional subspace. By analyzing high-dimensional data unnecessarily, computation resources and time can be wasted. When we use visualization techniques to gain insights about data, it is essential to understand whether the data at hand can be safely visualized at low dimensions, and to what extent the original information will be preserved via the visualization method. Several methods for intrinsic dimension estimation (IDE) have been proposed, and they can be roughly divided into two categories:

- projection-based methods and
- distance-based methods.

The former category of IDE methods basically involve two steps. First, the given dataset is partitioned. Then, in each partition, principal component analysis (PCA) or another procedure for finding a dominant subspace is performed. This approach is generally easy to implement and suitable for exploratory data analysis (Fukunaga and Olsen, 1971; Verveer and Duin, 1995; Kambhatla and Leen, 1997; Bruske and Sommer, 1998). However, the estimated dimension is heavily influenced by how the data space is partitioned. Moreover, it is also unknown how the threshold for the eigenvalue obtained by PCA should be determined. This class of methods is useful for explanatory analysis with human interaction and trial-and-error iteration. However, it is unsuitable for plugging into a pipeline for automated data analysis, and we do not consider this sort of method in this paper.

The package **ider** implements various methods for estimating the intrinsic dimension from a set of observed data using a distance-based approach (Pettis et al., 1979; Grassberger and Procaccia, 1983; Kégl, 2002; Levina and Bickel, 2005; Hein and Audibert, 2005; Fan et al., 2009; Gupta and Huang, 2010; Eriksson and Crovella, 2012; Hino et al., 2017). The implemented algorithms work with either a data matrix or a distance matrix. There are a large number of distance-based IDE methods. Among them, methods based on the fractal dimension (Mandelbrot, 1977) are well studied in the fields of both mathematics and physics. The proposed package **ider** implements the following fractal dimension-based methods:

`corint`: the correlation integral (Grassberger and Procaccia, 1983)

`convU`: the kernel-version of the correlation integral (Hein and Audibert, 2005)

`packG`, `packT`: capacity dimension-based methods with packing number estimation (a greedy method (Kégl, 2002) and a tree-based method (Eriksson and Crovella, 2012))

`mada`: first-order local dimension estimation (Farahmand et al., 2007)

`side`: second-order local dimension estimation (Hino et al., 2017)

There are several other distance-based methods, such as one based on a maximum-likelihood estimate of the Poisson distribution (Levina and Bickel, 2005), which approximates the distance distribution from an inspection point to other points in a given dataset. This method is implemented in our package as a function `lbmle`. A similar but different approach utilizing the nearest-neighbor information has also been implemented as a function `nni` (Pettis et al., 1979).

The proposed package also provides a data-generating function `gendata` that generates several famous artificial datasets often used as benchmarks for IDE and manifold learning.

Fractal dimensions

In fractal analysis, the Euclidean concept of a dimension is replaced with the notion of a fractal dimension, which characterizes how the given shape or datasets occupy their ambient space. There are many different definitions of the fractal dimension, from both mathematical and physical perspectives. Well-known fractal dimensions include the correlation dimension and the capacity dimension. There are already some R packages for estimating the fractal dimension, such as **fractal**, **nonlinearTseries**, and **tseriesChaos**. In **fractal** and **nonlinearTseries**, the correlation dimension and its generalization estimators are implemented, and in **tseriesChaos**, the method of false nearest neighbors (Kennel et al., 1992) is implemented. These packages focus on estimates of the embedded dimension of a time series in order to characterize its chaotic property. To complement the above-mentioned packages, we implemented several fractal dimension estimators for vector-valued observations.

Global dimensions

Correlation dimension

For a set of observed data $\mathcal{D} = \{x_i\}_{i=1}^n$, the correlation integral is defined as

$$V_2(\varepsilon) = \lim_{n\to\infty} \frac{2}{n(n-1)} \sum_{i<j}^{n} I(\|x_i - x_j\| < \varepsilon) \tag{1}$$

using a sufficiently small $\varepsilon > 0$. In Eq. (1), $I(u)$ is the indicator function which returns one when the statement u is true and zero if the statement is false. The correlation integral $V_2(\varepsilon)$ is the ratio of pairs whose distance is below ε, and this number grows as a length for a one-dimensional object, as a surface for a two-dimensional object, as a volume for a three-dimensional object, and so forth. So, it is natural to assume that $V_2(\varepsilon)$ grows proportional to the intrinsic dimension, and the intrinsic dimension associated with $V_2(\varepsilon)$ is defined as the correlation dimension. To be precise, using the correlation integral, the correlation dimension is defined as

$$p_{cor} = \lim_{\varepsilon\to 0} \frac{\log V_2(\varepsilon)}{\log \varepsilon}. \tag{2}$$

Intuitively, the number of sample pairs with a distance smaller than ε should increase in proportion to ε^p, where p is the intrinsic dimension. The correlation dimension exploits this property, i.e., $V_2(\varepsilon) \propto \varepsilon^p$, to define the intrinsic dimension p_{cor}. Grassberger and Procaccia (1983) proposed the use of the empirical (finite sample) estimates $\hat{V}_2(\varepsilon_k) = \frac{2}{n(n-1)} \sum_{i<j}^{n} I(\|x_i - x_j\| < \varepsilon_k)$, $k = 1, 2$ of the correlation integral $V_2(\varepsilon)$ with two different radii, ε_1 and ε_2, in order to estimate the correlation dimension (2) as follows:

$$\hat{p}_{cor}(\varepsilon_1, \varepsilon_2) = \frac{\log \hat{V}_2(\varepsilon_2) - \log \hat{V}_2(\varepsilon_1)}{\log \varepsilon_2 - \log \varepsilon_1}. \tag{3}$$

Hein and Audibert (2005) proposed the use of a U-statistic with the form

$$\hat{V}_{2,h} = \frac{2}{n(n-1)} \sum_{i<j}^{n} \kappa_h(\|x_i - x_j\|^2) \tag{4}$$

using a kernel function κ_h with bandwidth h to count the number of samples, and replaced the correlation integral by $\hat{V}_{2,h}$. The convergence of this U-statistic with $n \to \infty$, by an argument similar to kernel bandwidth selection (Wand and Jones, 1994), requires that $h \to 0$ and $nh^p \to \infty$. These conditions are used in (Hein and Audibert, 2005) to derive a formula for estimating the global intrinsic dimension p.

In the **ider** package, the classical correlation dimension estimator proposed in Grassberger and Procaccia (1983) is performed using the function `corint` as follows.

```
> set.seed(123)
> x <- gendata(DataName='SwissRoll', n=300)
> estcorint <- corint(x=x, k1=5, k2=10)
> print(estcorint)
> [1] 1.963088
```

where $k1$ and $k2$ respectively correspond to ε_1 and ε_2 in Eq. (3). Indeed, it is easy and safe to specify an integer k for the k-th nearest neighbor rather than the radius ε, because there is no guarantee that there is a data point in ε-ball in general. In the above example, we used the function `gendata` to generate

the famous 'SwissRoll' data with an ambient dimension of three and an intrinsic dimension of two. As observed, the correlation integral method by Grassberger and Procaccia (1983) works well for this dataset. The kernel-based correlation dimension estimator is performed by using the function `convU` as follows:

```
> set.seed(123)
> x <- gendata(DataName='SwissRoll', n=300)
> estconvU <- convU(x=x, maxDim=5)
> print(estconvU)
> [1] 2
```

The method proposed by Hein and Audibert (2005) attempts to find the possible intrinsic dimension one-by-one up to `maxDim`. Consequently, the estimated dimension can only be a natural number. All IDE functions in **ider** support both vector-valued data matrices and distance matrices as the input data. This is useful in cases where we exclusively obtain a distance matrix, and in cases where the original data object cannot be represented by a finite and fixed dimensional vector. This is also useful when we treat very high-dimensional data, such that retaining its distance matrix saves memory storage. To indicate that the input is a distance matrix, we set the parameter `DM` to `TRUE` as follows:

```
> set.seed(123)
> x <- gendata(DataName='SwissRoll', n=300)
> estcorint <- corint(x=dist(x), DM=TRUE, k1=5, k2=10)
> print(estcorint)
> [1] 1.963088
```

The distance matrix can be either a `matrix` object or `dist` object.

Capacity dimension

Let $\mathcal{X}$ be a given metric space with distance metric $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_+$. The ε-covering number $N(\varepsilon)$ of a set $\mathcal{S} \subset \mathcal{X}$ is the minimum number of open balls $b(z;\varepsilon) = \{x \in \mathcal{X} | d(x,z) < \varepsilon\}$ whose union is a covering of $\mathcal{S}$. The capacity dimension (Hentschel and Procaccia, 1983) or box-counting dimension is defined by

$$p_{cap} = -\lim_{\varepsilon \to 0} \frac{\log N(\varepsilon)}{\log \varepsilon}. \tag{5}$$

The intuition behind the definition of capacity dimension is the following. Assuming a three-dimensional space divided in small cubic boxes with a fixed edge length ε, the box-counting dimension is related to the proportion of occupied boxes. For a growing one-dimensional object placed in this compartmentalized space, the number of occupied boxes grows proportionally to the object length. Similarly, for a growing two-dimensional object, the number of occupied boxes grows proportionally to the object surface, and for a growing three-dimensional object, the number grows proportionally to the volume. Considering the situation that the size of the object remains unchanged but the edge length ε of the boxes decreases justify the definition of p_{cap}.

The problem of estimating p_{cap} is reduced to the problem of estimating $N(\varepsilon)$, but finding the covering number is computationally intractable. Kégl (2002) proposed the replacement of the covering number $N(\varepsilon)$ with the ε-packing number $M(\varepsilon)$. Given a metric space $\mathcal{X}$ with distance d, a set $V \subset \mathcal{X}$ is said to be ε-separated if $d(x,y) \geq \varepsilon$ for all $x, y \in V, x \neq y$. The ε-packing number $M(\varepsilon)$ is defined by the maximum cardinality of an ε-separated subset of the data space $\mathcal{X}$, and it is known that the inequalities $N(\varepsilon) \leq M(\varepsilon) \leq N(\varepsilon/2)$ hold. Considering the fact that the capacity dimension is defined as the limit $\varepsilon \to 0$, the following holds:

$$p_{cap} = -\lim_{\varepsilon \to 0} \frac{\log M(\varepsilon)}{\log \varepsilon}. \tag{6}$$

The capacity dimension based on the packing number is estimated using the estimates of the packing number at two different radii, ε_1 and ε_2, as

$$\hat{p}_{cap} = -\frac{\log \hat{M}(\varepsilon_2) - \log \hat{M}(\varepsilon_1)}{\log \varepsilon_2 - \log \varepsilon_1}. \tag{7}$$

The ε-packing number $M(\varepsilon)$ has been estimated using a greedy algorithm (Kégl, 2002) and by using a hierarchical clustering algorithm (Eriksson and Crovella, 2012).

In the **ider** package, the capacity dimension estimation is based on the packing number with greedy approximation, and it is performed using the function `pack` as

```
> set.seed(123)
> x <- gendata(DataName='SwissRoll', n=300)
> estpackG <- pack(x=x, greedy=TRUE)  ## estimate the packing number by greedy method
> print(estpackG)
> [1] 2.289935
```

whereas the hierarchical clustering-based method is performed as

```
> estpackC <- pack(x=x, greedy=FALSE) ## estimate the packing number by cluttering
> print(estpackC)
> [1] 2.393657
```

Packing-number-based methods require two radii ε_1 and ε_2, which are specified by arguments k_1 and k_2, respectively. If one of these arguments is NULL, both can be determined by a 0.25 and 0.75 quantile of distance from all pairs of data points.

Local dimensions

The former two fractal dimensions, viz., the correlation dimension and capacity dimension, are designed to estimate a global intrinsic dimension. Any global IDE method can be converted into a local method by running the global method on a neighborhood of a point. However, we introduce two inherently local fractal dimension estimators. The relationship between global and local fractal dimensions are shown in (Hino et al., 2017).

Let μ be an absolutely continuous probability measure on a metric space $\mathcal{X}$, and let the corresponding probability density function (pdf) be $f(x)$. Consider the problem of estimating the value of the pdf at a point $z \in \mathcal{X} \subseteq \mathbb{R}^p$ using a set of observations $\mathcal{D} = \{x_i\}_{i=1}^n$.

First-order method

Let the p-dimensional hyper-ball of radius ε centered at z be $b(z;\varepsilon) = \{x \in \mathbb{R}^p | d(z,x) < \varepsilon\}$. The probability mass of the ball $b(z;\varepsilon)$ is defined as

$$\Pr(X \in b(z;\varepsilon)) = \int_{x \in b(z;\varepsilon)} \mathrm{d}\mu(x) = \int_{x \in b(z;\varepsilon)} f(x)\mathrm{d}\nu(x),$$

where ν is the uniform measure in p-dimensional Euclidean space. We assume that for a sufficiently small radius $\varepsilon > 0$, the value of the pdf $f(z)$ is approximately a constant within the ball $b(z;\varepsilon)$. Under this assumption, using the Taylor series expansion of the probability mass, we obtain

$$\begin{aligned}\Pr(X \in b(z;\varepsilon)) =& \int_{x \in b(z;\varepsilon)} \left\{ f(z) + (x-z)^\top \nabla f(z) + O(\varepsilon^2) \right\} \mathrm{d}\nu(x) \\ =& |b(z;\varepsilon)| \left(f(z) + O(\varepsilon^2) \right) = c_p \varepsilon^p f(z) + O(\varepsilon^{p+2}),\end{aligned}$$

where $\int_{x \in b(z;\varepsilon)} \mathrm{d}\nu(x) = |b(z;\varepsilon)|$. The volume of a ball with a uniform measure is $|b(z;\varepsilon)| = c_p \varepsilon^p$, where $c_p = \pi^{p/2}/\Gamma(p/2+1)$, and $\Gamma(\,\cdot\,)$ is the gamma function. In this expansion, the integration is performed within the ε-ball; hence $x - z$ is of the same order as ε. The term with the first derivative of the density function vanishes owing to symmetry. When we fix the number of samples k falling within the ball $b(z;\varepsilon)$ instead of the radius ε, the radius ε is determined by the distance between the inspection point z and its k-th nearest neighbor. In this paper, ε_k denotes the radius determined by k. Inversely, when we fix the radius ε, the number of samples falling within the ε-ball centered at an inspection point is determined and denoted by k_ε. In (Farahmand et al., 2007), $\Pr(X \in b(z;\varepsilon))$ is approximated by the ratio of $b(z;\varepsilon)$ and the sample size n as follows:

$$\Pr(X \in b(z;\varepsilon)) \simeq \frac{k_\varepsilon}{n} \simeq c_p \varepsilon^p f(z). \tag{8}$$

Then, for different radii $\varepsilon_1, \varepsilon_2$, the logarithm of the above approximation formula derives the following:

$$\begin{aligned}\log \frac{k_{\varepsilon_1}}{n} &= \log c_p f(z) + p \log \varepsilon_1, \\ \log \frac{k_{\varepsilon_2}}{n} &= \log c_p f(z) + p \log \varepsilon_2.\end{aligned}$$

Solving this system of equations with respect to the dimension yields the estimate of the local fractal dimension:

$$\hat{p}_{mada} = \frac{\log k_{\varepsilon_2} - \log k_{\varepsilon_1}}{\log \varepsilon_2 - \log \varepsilon_1}. \tag{9}$$

The convergence rate of this estimator is independent of the ambient dimension, but it depends on the intrinsic dimension. Hence, $\hat{p}_{mada}$ is called the manifold adaptive dimension estimator in (Farahmand et al., 2007). This estimator is simple and easy to implement, and it provides a finite sample error bound. We explain the usage of this first-order local IDE method in package **ider**. To demonstrate the ability of a local estimate, we use a dataset "ldbl" (i.e., line-disc-ball-line), which comprises sub-datasets with one, two, and three dimensions embedded in three-dimensional space. Using this dataset, the example of the use of a first-order local IDE called `mada` is shown below:

```
> set.seed(123)
> tmp <- gendata(DataName='ldbl', n=300)
> x <- tmp$x
> estmada <- mada(x=x, local=TRUE)
> estmada[c(which(tmp$tDim==1)[1], which(tmp$tDim==2)[1], which(tmp$tDim==3)[1])]
> 1.113473 2.545525 2.207250
```

This sample code estimates the local intrinsic dimensions of every point in the dataset x, and shows the estimates at the points with true intrinsic dimensions of one, two, and three.

Second-order method

In (Hino et al., 2017), accurate local IDE methods based on a higher-order expansion of the probability mass function and Poisson regression modeling are proposed. By using the second-order Taylor series expansion for $\Pr(X \in b(z;\varepsilon))$, we obtain the following proposition:

Proposition 1 *The probability mass* $\Pr(X \in b(z;\varepsilon))$ *of the* ε*-ball centered at* z *is expressed in the form*

$$\Pr(X \in b(z;\varepsilon)) = c_p f(z)\varepsilon^p + \frac{p}{4(p/2+1)} c_p \mathrm{tr}\nabla^2 f(z)\varepsilon^{p+2} + O(\varepsilon^{p+4}).$$

The proof for this is detailed in (Hino et al., 2017). However, there is no need to know the exact form of the second-order expansion. By approximating the probability mass $\Pr(X \in b(z;\varepsilon))$ empirically with the ratio k_ε/n, i.e., the ratio of the number of samples falling into the ε-ball to the whole sample size, we obtain the following relationship:

$$\frac{k_\varepsilon}{n} = c_p f(z)\varepsilon^p + \frac{p}{4(p/2+1)} c_p \mathrm{tr}\nabla^2 f(z)\varepsilon^{p+2}$$

by ignoring the higher-order term with respect to ε. Furthermore, by multiplying both sides of each equation by n, and letting the coefficients of ε^p and ε^{p+2} be β_1 and β_2, respectively, we obtain

$$k_\varepsilon = \beta_1 \varepsilon^p + \beta_2 \varepsilon^{p+2}. \tag{10}$$

To estimate the intrinsic dimension using the second-order Taylor series expansion, we fit a generalized linear model (GLM; (Dobson, 2002)) to Eq. (10), which expresses the counting nature of the left-hand side of the equation.

Let the intrinsic dimension at the inspection point z be p $(p = 1, \ldots, \mathrm{maxDim})$, where maxDim is the pre-determined upper limit of the intrinsic dimension. We express realizations of a vector-valued random variable $x_{\varepsilon,p} \in \mathbb{R}^2$, which is composed of ε^p and ε^{p+2}, where ε is the distance from the inspection point, by

$$x_{\varepsilon,p} \in \left\{ \begin{pmatrix} \varepsilon_1^p \\ \varepsilon_1^{p+2} \end{pmatrix}, \begin{pmatrix} \varepsilon_2^p \\ \varepsilon_2^{p+2} \end{pmatrix}, \ldots \right\}. \tag{11}$$

We also introduce realizations of a random variable $y_\varepsilon = k_\varepsilon \in \{1, 2, \ldots\}$, which is the number of samples included in the ball $b(z;\varepsilon)$. Specifically, we consider a pair of random variables (Y, X_p) and fix a radius ε corresponding to a *trial* that results in realizations $(y_\varepsilon, x_{\varepsilon,p})$. Because the realization y_ε is the number of samples within the ε-ball, and assuming that the number of observation n is sufficiently large, we assume that the error structure of Y is a Poisson distribution. Then, we can formulate the relationship between the distance from the inspection point ε and the number of samples falling within the ε-ball using a generalized linear model with a Poisson error structure and linear link function as follows:

$$E[y] = x^\top \beta. \tag{12}$$

A set of m different radii is denoted as $\mathcal{E}$, i.e., $\{\varepsilon_1, \ldots, \varepsilon_m\} \in \mathcal{E}$. We maximize the log-likelihood of the Poisson distribution with the observation $\{(y_\varepsilon, x_{\varepsilon,p})\}_{\varepsilon\in\mathcal{E}}$ with respect to the coefficient vector $\beta \in \mathbb{R}^2$. In this work, we simply consider the m-th nearest neighbor with $m = \min\{\lceil n/5 \rceil, 100\}$, and let the Euclidean distance from the inspection point z to its m-th nearest point $x_{(m)}$ be $d(z, x_{(m)})$. Then, we uniformly sample m radii $\mathcal{E} = \{\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_m\}$ from a uniform distribution in $[0, d(z, x_{(m)})]$. Let the observation vector and design matrix, which are composed of realizations of Y and X, be

$$\boldsymbol{y} = (y_{\varepsilon_1}, y_{\varepsilon_2}, \ldots, y_{\varepsilon_m})^\top \in \mathbb{R}^m \tag{13}$$

$$\begin{aligned}\boldsymbol{X}_p &= (x_{\varepsilon_1,p}, x_{\varepsilon_2,p}, \ldots, x_{\varepsilon_m,p})^\top \\ &= \begin{pmatrix} \varepsilon_1^p & \varepsilon_2^p & \ldots & \varepsilon_m^p \\ \varepsilon_1^{p+2} & \varepsilon_2^{p+2} & \ldots & \varepsilon_m^{p+2} \end{pmatrix}^\top \in \mathbb{R}^{m\times 2}.\end{aligned} \tag{14}$$

We consider a generalized linear model (Dobson, 2002) with the linear predictor $\boldsymbol{X}_p\beta$ and identity link

$$E[\boldsymbol{y}] = \boldsymbol{X}_p\beta, \tag{15}$$

and the log-likelihood function of the Poisson distribution:

$$L(\{\boldsymbol{y}, \boldsymbol{X}_p\}; \beta) = \log \prod_{\varepsilon\in\mathcal{E}} \frac{e^{-x_{\varepsilon,p}^\top \beta} (x_{\varepsilon,p}^\top \beta)^{y_\varepsilon}}{y_\varepsilon!}. \tag{16}$$

By assuming that the intrinsic dimension is p, the optimal IDE is estimated on the basis of the goodness of fit of the data to the regression model (10). We use the log-likelihood (16) to measure this goodness of fit. Note that the number of parameters is always two, even when we change the assumed IDE p; hence, the problem of over-fitting by maximizing the likelihood is avoided in our setting.

In the package **ider**, two IDE algorithms are implemented based on the maximization of Eq. (16). The first method simply assumes distinct intrinsic dimensions and maximizes the log-likelihood (16) with respect to β with fixed p. Let the ambient dimension or maximum possible dimension of the observation be maxDim. We assume that the intrinsic dimension is $p = 1, 2, \ldots,$ maxDim, and for every p, we fit the regression model (10) by maximizing the log-likelihood (16) with respect to β, and employ the dimension p that maximizes the likelihood:

$$\hat{p}_{s1} = \operatorname*{arg\,max}_{p\in\{1,\ldots,\text{maxDim}\}} \max_{\beta\in\mathbb{R}^2} L(\{\boldsymbol{y}, \boldsymbol{X}_p\}; \beta). \tag{17}$$

The second method treats the log-likelihood (16) as a function of both the regression coefficients $\beta \in \mathbb{R}^2$ and the intrinsic dimension $p \in \mathbb{R}_+$. Given a set of observations, we can maximize the log-likelihood function with respect to (p, β_1, β_2). Because it is difficult to obtain a closed-form solution for the maximizer of the likelihood (16), we numerically maximize the likelihood to obtain the estimate as

$$\hat{p}_{s2} = \operatorname*{arg\,max}_{p\in\mathbb{R}_+} \max_{\beta\in\mathbb{R}^2} L(\{\boldsymbol{y}, \boldsymbol{X}_p\}; \beta) \tag{18}$$

by using the quasi-Newton (BFGS) method. The initial point for the variables (p, β_1, β_2) is set to the estimate obtained using the first method explained above.

In the **ider** package, a second-order local IDE with discrete dimensions is performed using the function `side` (Second-order Intrinsic Dimension Estimator) as follows:

```
> set.seed(123)
> tmp <- gendata(DataName='ldbl', n=300)
> x <- tmp$x
> idx <- c(sample(which(tmp$tDim==1)[1:10], 3), sample(which(tmp$tDim==2)[1:30], 3))
> estside <- side(x=x[1:100,], local=TRUE, method='disc')
> print(estside[idx]) ## estimated discrete local intrinsic dimensions by side
[1] 1 1 1 3 1 2
```

An example of the same, using the method 'cont' is as follows:

```
> estside <- side(x=x[1:100,], local=TRUE, method='cont')
> print(estside[idx]) ## estimated continuous local intrinsic dimensions by side
[1] 1.020254 1.338089 1.000000 2.126269 3.360426 2.074643
```

It is seen that the obtained estimates are not natural numbers.

The local dimension estimate is easily aggregated to a global estimate by taking an average, median, or voting of local estimates, and this is realized when we set the argument `local = TRUE` in

mada or side. The functions mada and side have an argument comb to specify how the local estimates are combined. When comb='average', the local estimates are averaged as a global IDE. Likewise, when comb='median', the median of the local estimates is adopted; and when comb='vote', the voting of local estimates is adopted as a global IDE. Note that the combination method vote should be used only with method='disc'.

```
> set.seed(123)
> x <- gendata(DataName='SwissRoll', n=300)
> estmada <- mada(x=x, local=FALSE, comb='median')
> estmada
[1] 1.754866
> estside <- side(x=x, local=FALSE, comb='median', method='disc')
> estside
[1] 2
```

Other distance-based approaches

The package **ider** supports two other distance-based dimension-estimation methods, namely lbmle and nni.

Maximum likelihood estimation

Levina and Bickel (2005) derived the maximum likelihood estimator of the dimension p from i.i.d. observations $\mathcal{D} = \{x_i\}_{i=1}^n$. Let f be a pdf of the data points smoothly embedded in a p-dimensional space, i.e., a space with intrinsic dimension, and assume that when a point x is fixed, the value of pdf $f(x)$ is constant in a small ball $b(x;\varepsilon)$. Consider the point process

$$\{N(t,x),\ 0 \le t \le \varepsilon\}, \quad N(t,x) = \sum_{i=1}^{n} \mathbf{1}\{x_i \in b(x;t)\}, \tag{19}$$

which counts the observations within distance t from the inspection point x. This point process is approximated using a homogeneous Poisson process, with rate

$$\lambda(t) = c_p f(x) p t^{p-1}. \tag{20}$$

The log-likelihood of the observed process $N(t)$ is written as

$$L(p, \log f(x)) = \int_0^{\varepsilon} \log \lambda(t) \mathrm{d}N(t) - \int_0^{\varepsilon} \lambda(t) \mathrm{d}t. \tag{21}$$

Solving the likelihood equation, the maximum likelihood estimate of the intrinsic dimension around x is

$$\hat{p}_\varepsilon(x) = \left\{ \frac{1}{N(\varepsilon,x)} \sum_{j=1}^{N(\varepsilon,x)} \log \frac{\varepsilon}{\varepsilon_j(x)} \right\}^{-1}, \tag{22}$$

or, more conveniently in practice,

$$\hat{p}_k(x) = \left\{ \frac{1}{k-1} \sum_{j=1}^{k-1} \log \frac{\varepsilon_k(x)}{\varepsilon_j(x)} \right\}^{-1}, \tag{23}$$

where $\varepsilon_j(x)$ denotes the distance between the inspection point x to its j-th nearest point. Then, choosing two indices, k_1 and k_2, the maximum likelihood estimate $\hat{p}_{ml}$ of the intrinsic dimension is obtained as follows:

$$\hat{p}_{ml} = \frac{1}{k_2 - k_1 + 1} \sum_{k=k_1}^{k_2} \hat{q}_k, \qquad \hat{q}_k = \frac{1}{n} \sum_{i=1}^{n} \hat{p}_k(x_i). \tag{24}$$

In the **ider** package, the maximum likelihood estimation is obtained by using the function lbmle:

```
> set.seed(123)
> x <- gendata(DataName='SwissRoll', n=300)
> estmle <- lbmle(x=x, k1=3, k2=5, BC=FALSE)
> print(estmle)
[1] 3.174426
```

It was pointed out by MacKay and Ghahramani that the above MLE contains a certain bias [1]. With the function `lbmle`, however, we can calculate the bias-corrected estimate by setting the argument `BC`, which stands for "bias-correction", to `TRUE`:

```
> set.seed(123)
> x <- gendata(DataName='SwissRoll', n=300)
> estmle <- lbmle(x=x, k1=3, k2=5, BC=TRUE)
> print(estmle)
[1] 2.032756
```

Near-neighbor information

Pettis et al. (1979) proposed an IDE method based on the analysis of the distribution of distances from one point to its nearest neighbors. Pettis et al. (1979) derived that the distribution of the distance from a point x to its k-th nearest neighbor $\epsilon_{k,x}$ is, based on the Poisson approximation, given by the following probability density function

$$f_{k,x}(\epsilon_{k,x}) = nf(x)c_p \frac{\{nf(x)c_p\}^{k-1}}{\Gamma(k)} \exp(-nf(x)c_p\epsilon_{k,x}^p). \tag{25}$$

The expected value of the sample average of distance to the k-th nearest neighbor over the given dataset is

$$\mathrm{E}[\bar{\epsilon}_k] = \frac{1}{n}\sum_{i=1}^{n} \mathrm{E}[\epsilon_{k,x_i}] = \frac{1}{G_{k,p}} k^{1/p} A_n, \tag{26}$$

where

$$G_{k,p} = \frac{k^{1/p}\Gamma(k)}{\Gamma(k+1/p)}, \quad A_n = \frac{1}{n}\sum_{i=1}^{n} \{nf(x_i)c_p\}^{-1/p}. \tag{27}$$

Note that A_n is sample-dependent but independent of k. Let $\hat{p}_0$ be the first rough estimate of the intrinsic dimension. Taking logarithm of eq. (26) yields

$$\log G_{k,\hat{p}_0} + \log \bar{\epsilon}_k = \frac{1}{p}\log k + \log A_n, \tag{28}$$

where $\mathrm{E}[\epsilon_k]$ is replaced with the sample average $\bar{\epsilon}_k$. From k_1 to k_2, we calculate the left hand side of eq. (28) for each k, and treat them as the realizations of the *response variable*. Linear regression of $\log k,\ k \in [k_1, k_2]$ on those response variable yields the updated estimate $\hat{p}_1$ of the intrinsic dimension. Replacing $\hat{p}_0$ in $G_{k,p}$ with the updated $\hat{p}_1$ and repeat the procedure until the gap between the new and the old estimates $\hat{p}$ is smaller than certain threshold.

The estimator is implemented as a function `nni` in **ider** and used as follows:

```
> set.seed(123)
> x <- gendata(DataName='SwissRoll', n=300)
> estnni <- nni(x=x)
> print(estnni)
[1] 2.147266
```

The function `nni` has parameters $k1$ and $k2$, which are the same in `lbmle`. This method is based on an iterative estimate of IDE, and the function `nni` has a parameter `eps` to specify the threshold for stopping the iteration, which is set at 0.01 by default.

Data-generating function

The **ider** package is equipped with a data-generating function `gendata`. It can generate nine different artificial datasets, which are manifolds of dimension p embedded in ambient space of dimension $(\geq p)$. The dataset is specified by setting the argument `DataName` to one of the following:

`SwissRoll` SwissRoll data, a 2D manifold in 3D space.

`NDSwissRoll` Non-deformable SwissRoll data, a 2D manifold in 3D space.

`Moebius` Moebius strip, a 2D manifold in 3D space.

`SphericalShell` Spherical Shell, p-dimensional manifold in $(p+1)$-dimensional space.

`Sinusoidal` Sinusoidal data, a 1D manifold in 3D space.

[1] http://www.inference.phy.cam.ac.uk/mackay/dimension/

`Spiral` Spiral-shaped data, a 1D manifold in 2D space.

`Cylinder` Cylinder-shaped data, a 2D manifold in 3D space.

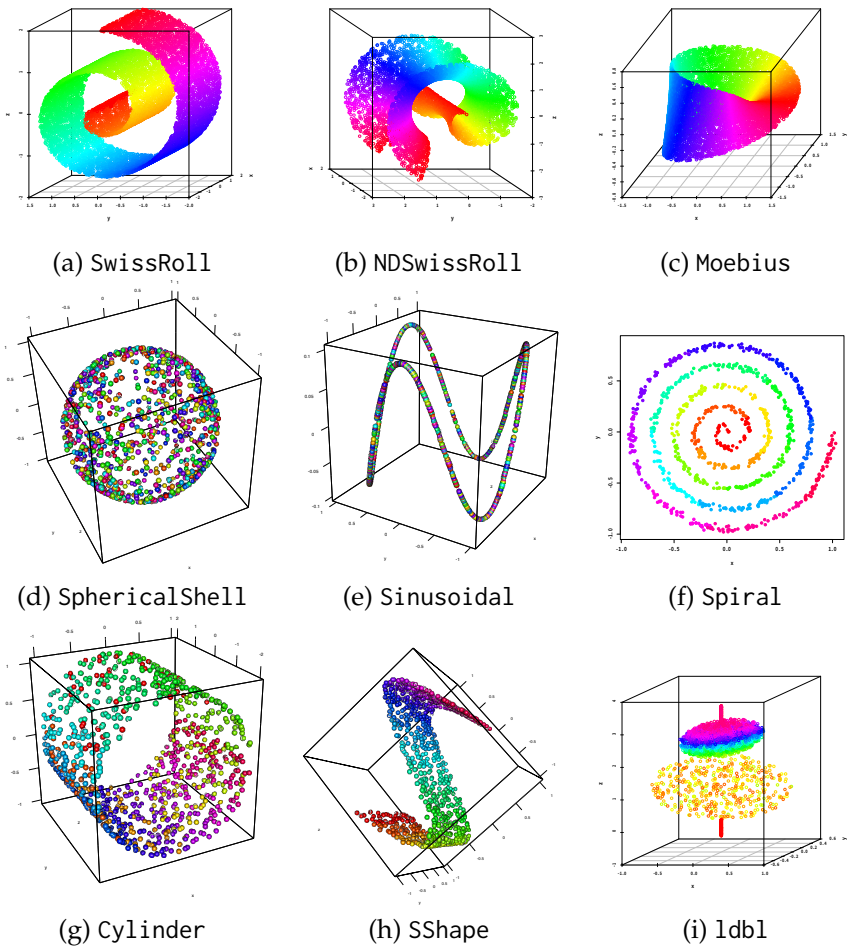
`SShape` S-shaped data, a 2D manifold in 3D space.

`ldbl` Four subspaces, line - disc - filled ball - line, in this order, along the z-axis, embedded in 3D space.

The final dataset `ldbl` is used to see the ability of local dimension estimations. The dataset comprises four sub-manifolds: line-shape (1D), disc (2D), filled ball (3D), and line-shape again, and these four sub-manifolds are concatenated in this order.

The parameter `n` of the function `gendata` specifies the number of samples in a dataset. All but the `SphericalShell` dataset have fixed ambient and intrinsic dimensions. For the `SphericalShell` dataset, an arbitrary integer can be set as the ambient dimension by setting the argument `p`.

The realizations of each dataset are shown in Fig. 1.



(a) `SwissRoll` (b) `NDSwissRoll` (c) `Moebius`

(d) `SphericalShell` (e) `Sinusoidal` (f) `Spiral`

(g) `Cylinder` (h) `SShape` (i) `ldbl`

Figure 1: Samples of datasets generated by `gendata`.

Example: estimating the degree of freedom of hand motion

The aim of this paper is to introduce the package **ider** and explain the standard usage of its implemented functions. Exhaustive experiments comparing IDE methods in various settings can be found in (Hino et al., 2017). In this paper, as a simple example of the application of the IDE methods to realistic problems, we consider estimating the intrinsic dimension of a set of images. We used the CMU Hand Rotation dataset[2], which was also used in (Kégl, 2002) and (Levina and Bickel, 2005). Examples of the hand images are shown in Fig. 2. The original CMU Hand Rotation dataset is composed of 481 images of 512×480 pixels. In the package **ider**, the distance matrix of these images is included:

[2]http://vasc.ri.cmu.edu/idb/html/motion/hand/index.html

Figure 2: Example images in the Hand Rotation dataset.

```
> data(handD)
> str(handD)
Class 'dist'  atomic [1:115440] 4.96 8.27 8.33 8.31 8.12 ...
  ..- attr(*, "Labels")= chr [1:481] "dimg" "dimg" "dimg" "dimg" ...
  ..- attr(*, "Size")= int 481
  ..- attr(*, "call")= language as.dist.default(m = handD)
  ..- attr(*, "Diag")= logi FALSE
  ..- attr(*, "Upper")= logi FALSE
 > dim(as.matrix(handD))
 [1] 481 481
```

Because the object `handD` is a distance matrix, when we apply IDE methods to this data, we must set the argument `DM` to `TRUE`:

```
>lbmle(x=handD, DM=TRUE, k1=3, k2=5, BC=TRUE, p=NULL)
[1] 4.433915
>corint(x=handD, DM=TRUE, k1=3, k2=10)
[1] 2.529079
> pack(x=handD, DM=TRUE, greedy=TRUE)
[1] 3.314233
> pack(x=handD, DM=TRUE, greedy=FALSE)
[1] 2.122698
> nni(handD, DM=TRUE)
[1] 2.646178
> side(x=handD, DM=TRUE, local=FALSE, method='disc', comb='median')
[1] 3
> side(x=handD, DM=TRUE, local=FALSE, method='cont', comb='median')
[1] 2
```

These results suggest that even the extrinsic dimension of the image is very high ($512 \times 480 = 245760$), whereas the intrinsic dimension is quite low, and there are only a few between 2 to 4.5.

Computational cost

We measured computational costs of dimension estimation methods for `SwissRoll` dataset, where the number of observations n were varied from 500 to 3000 by 500. Datasets of size n are repeatedly generated 100 times. The experiments are performed on an iMac with Mac OS X, 2.6GHz Intel Core i7 processor, with 16GB memory. The results are shown in Fig. 3 by boxplots. It is seen that the computational costs of `packT` and `side` grow almost quadratically. Among various methods, `lbmle` and `nni` are computationally very efficient. When we apply `packT` or `side` to large scale dataset, it is advised to subsample the original dataset.

Summary and future directions

In this paper, we introduced an R package, **ider**, that implements several IDE algorithms for vector-valued observation or distance matrices. Different IDE methods capture different aspects of the data distribution in low-dimensional subspace, and the estimated dimension varies. Our goal in developing **ider** is to provide a systematic method for selecting or comparing different methods for analyzing a given dataset. In practice, it is not expected that there is a ground-truth intrinsic dimension. In such cases, one possible approach is to apply all of the IDE methods provided in **ider**. In doing so, we can check whether the estimates agree with one another. If a consensus on the estimate cannot be reached, the reason why will be of interest, and this can help to characterize the nature of the data distribution.

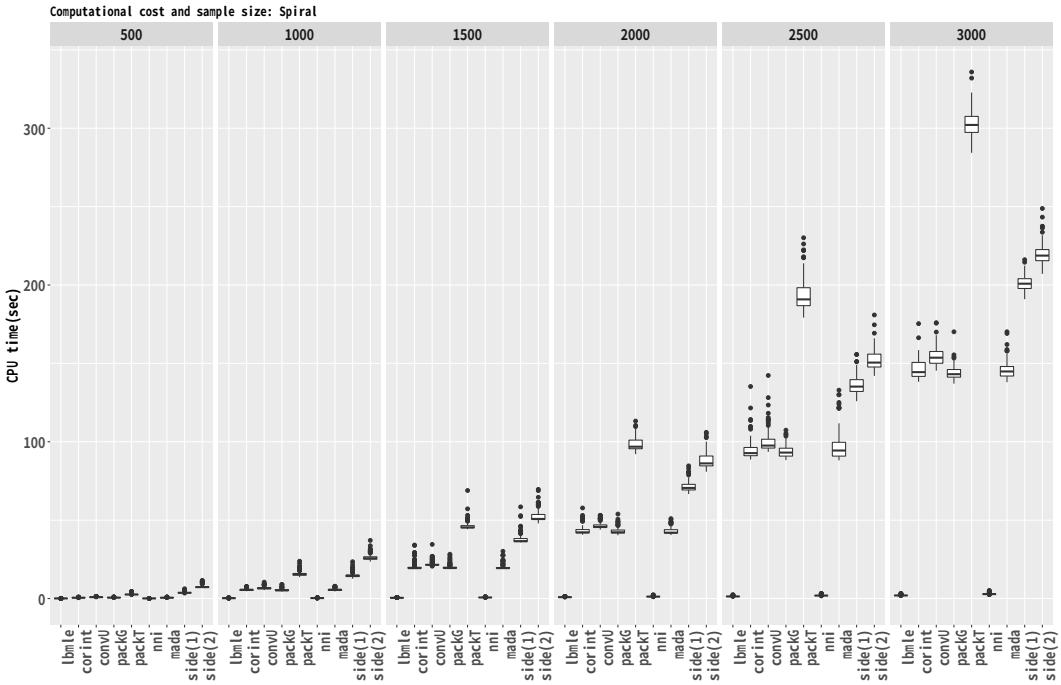


Figure 3: Computational costs and number of observations.

It is worth noting that there is another approach to IDE based on the minimum spanning tree. In (Costa et al., 2004), a method for estimating the f-divergence (Ali and Silvey, 1966; Csiszár and Shields, 2004) based on the total edge length of the minimum spanning tree of the observed objects has been proposed, and the authors used this method to estimate the intrinsic dimension (Costa and Hero, 2006; Carter et al., 2010). This kind of IDE method, and some of the projection-based methods, shall be included in a future version of the package **ider**. Furthermore, as we included representative fractal dimension estimation methods, the related methods based on them are not covered in the current version of **ider**. For example, the correlation dimension by Grassberger and Procaccia (1983) is known to have finite sample bias, and bias correction method based on simulated data is proposed in (Camastra and Vinciarelli, 2002). Also, the maximum likelihood and other approaches for correlation dimension estimation are summarized in (Camastra, 2003). We will update the package with implementations of these variations of the methods already included in current **ider**.

Finally, another important future work is improving computational efficiency. IDE methods based on packing number and second-order expansion of probability mass function are not computationally efficient. Implementation using **Rcpp** or parallelization would be an effective means for realizing the fast computation, and we will now be working on a parallel implementation.

Acknowledge

The author would like to express their special thanks to the editor, the associate editor and anonymous reviewers whose comments led to valuable improvements of the manuscript. Part of this research was supported by JSPS KAKENHI Grant Numbers 16H02842, 25120011, 16K16108, and 17H01793.

Bibliography

S. M. Ali and S. D. Silvey. A General Class of Coefficients of Divergence of One Distribution from Another. *Journal of the Royal Statistical Society B*, 28(1):131–142, 1966. URL http://www.jstor.org/stable/2984279. [p339]

J. Bruske and G. Sommer. Intrinsic dimensionality estimation with optimally topology preserving maps. *IEEE Trans. Pattern Anal. Mach. Intell.*, 20(5):572–575, 1998. ISSN 0162-8828. URL https://doi.org/10.1109/34.682189. [p329]

F. Camastra. Data dimensionality estimation methods: a survey. *Pattern Recognition*, 36(12):2945–2954, 2003. URL https://doi.org/10.1016/s0031-3203(03)00176-6. [p339]

F. Camastra and A. Vinciarelli. Estimating the intrinsic dimension of data with a fractal-based method. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(10):1404–1407, 2002. ISSN 0162-8828. URL https://doi.org/10.1109/tpami.2002.1039212. [p339]

K. M. Carter, R. Raich, and A. O. H. III. On local intrinsic dimension estimation and its applications. *IEEE Transactions on Signal Processing*, 58(2):650–663, 2010. ISSN 1053-587X. URL https://doi.org/10.1109/tsp.2009.2031722. [p339]

R. D. Cook and X. Yin. Dimension reduction and visualization in discriminant analysis (with discussion). *Australian and New Zealand Journal of Statistics*, 43(2):147–199, 2001. URL https://doi.org/10.1111/1467-842x.00164. [p329]

J. A. Costa and A. O. Hero. *Determining Intrinsic Dimension and Entropy of High-Dimensional Shape Spaces*, chapter 8, pages 231–252. Birkhäuser Boston, Boston, MA, 2006. ISBN 978-0-8176-4481-9. URL https://doi.org/10.1007/0-8176-4481-4_9. [p339]

J. A. Costa, A. O. Hero, and III. Geodesic entropic graphs for dimension and entropy estimation in manifold learning. *IEEE Transactions on Signal Processing*, 52:2210–2221, 2004. URL https://doi.org/10.1109/tsp.2004.831130. [p339]

I. Csiszár and P. C. Shields. Information theory and statistics: A tutorial. *Commun. Inf. Theory*, 1(4): 417–528, 2004. ISSN 1567-2190. URL https://doi.org/10.1561/0100000004. [p339]

A. J. Dobson. *An Introduction to Generalized Linear Models*. Chapman & Hall/CRC, Boca Raton :, 2nd ed. edition, 2002. URL https://doi.org/10.4135/9781412983273. [p333, 334]

B. Eriksson and M. Crovella. Estimating intrinsic dimension via clustering. In *Statistical Signal Processing Workshop (SSP), 2012 IEEE*, pages 760–763, 2012. URL https://doi.org/10.1109/ssp.2012.6319815. [p329, 331]

M. Fan, H. Qiao, and B. Zhang. Intrinsic dimension estimation of manifolds by incising balls. *Pattern Recogn.*, 42(5):780–787, 2009. ISSN 0031-3203. URL https://doi.org/10.1016/j.patcog.2008.09.016. [p329]

A. M. Farahmand, C. Szepesvári, and J.-Y. Audibert. Manifold-adaptive dimension estimation. In *Machine Learning, Proceedings of the Twenty-Fourth International Conference (ICML 2007), Corvallis, Oregon, USA, June 20-24, 2007*, pages 265–272, 2007. URL https://doi.org/10.1145/1273496.1273530. [p329, 332, 333]

K. Fukunaga and D. R. Olsen. An algorithm for finding intrinsic dimensionality of data. *IEEE Trans. Comput.*, 20(2):176–183, 1971. ISSN 0018-9340. URL https://doi.org/10.1109/t-c.1971.223208. [p329]

P. Grassberger and I. Procaccia. Measuring the strangeness of strange attractors. *Physica D: Nonlinear Phenomena*, 9(1–2):189–208, 1983. ISSN 0167-2789. URL https://doi.org/10.1016/0167-2789(83)90298-1. [p329, 330, 331, 339]

M. D. Gupta and T. S. Huang. Regularized maximum likelihood for intrinsic dimension estimation. In *UAI 2010, Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence, Catalina Island, CA, USA, July 8-11, 2010*, pages 220–227, 2010. [p329]

M. Hein and J.-Y. Audibert. Intrinsic dimensionality estimation of submanifolds in R^d. In *Machine Learning, Proceedings of the Twenty-Second International Conference (ICML 2005), Bonn, Germany, August 7-11, 2005*, pages 289–296, 2005. URL https://doi.org/10.1145/1102351.1102388. [p329, 330, 331]

H. G. E. Hentschel and I. Procaccia. The infinite number of generalized dimensions of fractals and strange attractors. *Physica D: Nonlinear Phenomena*, 8(3):435–444, 1983. ISSN 0167-2789. URL https://doi.org/10.1016/0167-2789(83)90235-x. [p331]

H. Hino, J. Fujiki, S. Akaho, and N. Murata. Local intrinsic dimension estimation by generalized linear modeling. *Neural Computation*, 29(7):1838–1878, 2017. URL https://doi.org/10.1162/neco_a_00969. [p329, 332, 333, 337]

N. Kambhatla and T. K. Leen. Dimension reduction by local principal component analysis. *Neural Computation*, 9(7):1493–1516, 1997. URL https://doi.org/10.1162/neco.1997.9.7.1493. [p329]

M. B. Kennel, R. Brown, and H. D. I. Abarbanel. Determining embedding dimension for phase-space reconstruction using a geometrical construction. *Phys. Rev. A*, 45:3403–3411, 1992. URL https://doi.org/10.1103/physreva.45.3403. [p330]

E. Kokiopoulou and Y. Saad. Orthogonal Neighborhood Preserving Projections: A projection-based dimensionality reduction technique. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(12):2143–2156, 2007. URL https://doi.org/10.1109/tpami.2007.1131. [p329]

B. Kégl. Intrinsic dimension estimation using packing numbers. In *Advances in Neural Information Processing Systems 15 [Neural Information Processing Systems, NIPS 2002, December 9-14, 2002, Vancouver, British Columbia, Canada]*, pages 681–688, 2002. [p329, 331, 337]

E. Levina and P. J. Bickel. Maximum likelihood estimation of intrinsic dimension. In L. K. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 777–784. MIT Press, Cambridge, MA, 2005. [p329, 335, 337]

B. B. Mandelbrot. *The Fractal Geometry of Nature*. W. H. Freeman and Company, 1977. URL https://doi.org/10.1119/1.13295. [p329]

K. W. Pettis, T. A. Bailey, A. K. Jain, and R. C. Dubes. An intrinsic dimensionality estimator from near-neighbor information. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PAMI-1 (1):25–37, 1979. ISSN 0162-8828. URL https://doi.org/10.1109/tpami.1979.4766873. [p329, 336]

P. J. Verveer and R. P. W. Duin. An evaluation of intrinsic dimensionality estimators. *IEEE Trans. Pattern Anal. Mach. Intell.*, 17(1):81–86, 1995. ISSN 0162-8828. URL https://doi.org/10.1109/34.368147. [p329]

M. P. Wand and M. C. Jones. *Kernel Smoothing*. Chapman & Hall/CRC, 1994. ISBN 0412552701. [p330]

Hideitsu Hino
Graduate School of Systems and Information Engineering, University of Tsukuba
1–1–1 Tennoudai, Tsukuba, Ibaraki, 305–8573
Japan
hideitsu.hino@gmail.com

rpsftm: An R Package for Rank Preserving Structural Failure Time Models

by Annabel Allison, Ian R White and Simon Bond

Abstract

Treatment switching in a randomised controlled trial occurs when participants change from their randomised treatment to the other trial treatment during the study. Failure to account for treatment switching in the analysis (i.e. by performing a standard intention-to-treat analysis) can lead to biased estimates of treatment efficacy. The rank preserving structural failure time model (RPSFTM) is a method used to adjust for treatment switching in trials with survival outcomes. The RPSFTM is due to Robins and Tsiatis (1991) and has been developed by White et al. (1997, 1999).

The method is randomisation based and uses only the randomised treatment group, observed event times, and treatment history in order to estimate a causal treatment effect. The treatment effect, ψ, is estimated by balancing counter-factual event times (that would be observed if no treatment were received) between treatment groups. G-estimation is used to find the value of ψ such that a test statistic $Z(\psi) = 0$. This is usually the test statistic used in the intention-to-treat analysis, for example, the log rank test statistic.

We present an R package, **rpsftm**, that implements the method.

Introduction

In a two-arm randomised controlled trial, participants are randomly allocated to receive one of two treatments or interventions. Ideally, all participants would fully receive their allocated treatment and no other treatment. However, in a recent review, 98 of 100 trials published in four high quality general medical journals reported some form of departure from randomised treatment (Dodd et al., 2012). When treatment is 'all-or-nothing' (for example, a surgical procedure), possible departures include not receiving the allocated treatment, receiving the other trial treatment, or receiving a non-trial treatment. When treatment is given over time (for example, a drug for HIV treatment), departures also occur over time, and often include starting a new treatment in response to a disease-related event.

This paper specifically focuses on the case of treatment switching over time, where participants may switch to receive the other trial treatment during the trial (Latimer et al., 2014). A common example is in a trial of active treatment versus placebo where placebo arm participants may start the active treatment in response to disease progression.

A randomised controlled trial with departures from randomised treatment is commonly analysed by intention-to-treat, in which departures from randomised treatment are ignored and all randomised participants are compared in the groups to which they were randomised (Higgins et al., 2011). This provides an unbiased comparison of two treatment policies, which accept that treatment may be changed, but does not compare the efficacies of the treatments themselves (White et al., 1999).

In order to compare the efficacies of the treatments, analysts often use per-protocol analysis, which censors participants when they depart from randomised treatment. However, this loses the advantage that randomisation produces comparable groups and instead introduces selection bias. Another possibility is to include treatment as a time-dependent variable in a Cox regression model. The issue with this method is that the estimate of treatment effect may not have a causal interpretation when switchers are prognostically different to non-switchers (Robins and Tsiatis, 1991).

There has therefore been strong interest in causal inference methods including marginal structural models (MSM) (i.e. inverse probability of censoring weighting (IPCW)) (Hernán et al., 2001) and instrumental-variable-type methods (White, 2005; Watkins et al., 2013).

The IPCW method is an extension of the per-protocol censoring approach, whereby the bias associated with censoring participants that depart from randomised treatment is removed by weighting the remaining non-switchers (Latimer et al., 2016). A model is formed for the probability of switching using baseline and time-dependent covariates. Time-varying weights are then obtained for each participant based on the inverse probability of not switching until a given time (Watkins et al., 2013). The method relies on data on prognostic factors for mortality that also affect the probability of switching being collected at both baseline and over time. This is called the 'no unmeasured confounders' assumption. Since this assumption is untestable, the method relies on good planning at the study design stage in order to collect information on all possible confounders. The method

also has two potential issues: a) if many prognostic factors are included in the model to calculate weights the model may fail to converge, and b) if only a few participants switch then large weights are assigned to the remaining participants, which may result in biased analyses. The method has already been implemented in R in the package **ipw** (van der Wal and Geskus, 2011) and so is not considered further here.

Instead, this paper focuses on a popular causal method - the rank preserving structural failure time model (RPSFTM) (Robins and Tsiatis, 1991). In contrast to the IPCW method which requires potential confounders to be collected over time, the RPSFTM is randomisation based and only requires information on the randomised treatment group, observed event times, and treatment history in order to estimate a causal treatment effect. The method has been used, for example, in submissions to the UK's National Institute for Health and Clinical Excellence whose Decision Support Unit commissioned a guidance document (Latimer and Abrams, 2014). The RPSFTM has been implemented in Stata (White et al., 2002) but not in R.

This paper describes the theory of the RPSFTM and presents a new implementation of the method with the R package **rpsftm** (Bond and Allison, 2017), using data based on a trial in HIV infection.

Theory

RPSFTM method and assumptions

The RPSFTM uses a causal model to produce counter-factual event times (the time that would be observed if no treatment were received) in order to estimate a causal treatment effect. Let $T_i = T_i^{\text{off}} + T_i^{\text{on}}$ be the observed event time for participant i, where T_i^{off} and T_i^{on} represent the time spent off and on treatment, respectively. The T_i are related to the counter-factual event times, U_i, via the following causal model:

$$U_i = T_i^{\text{off}} + T_i^{\text{on}} \exp(\psi_0), \tag{1}$$

where $\exp(-\psi_0)$ is the acceleration factor associated with treatment and ψ_0 is the true causal parameter.

A grid search (g-estimation) procedure is used to estimate the treatment effect that balances the counter-factual event times across randomised treatment groups. To estimate the causal treatment effect, ψ, we assume that the U_i are independent of randomised treatment group R_i, i.e. if the groups are similar with respect to all other characteristics except treatment, the average event times should be the same in each group if no participant were treated. In general, this assumption is plausible in a randomised controlled trial.

A g-estimation procedure is used to find the value of ψ such that U_i is independent of R_i. For a range of ψs, the hypothesis $\psi_0 = \psi$ is tested by computing $U_i(\psi)$, subject to censoring, and calculating the test statistic $Z(\psi)$. This is usually the same test statistic as for the intention-to-treat analysis, for example, the log rank test statistic to compare survival distributions between groups. In the `rpsftm()` function, the possible test options are the log rank, and the Wald test from a Cox or Weibull regression model. For the parametric Weibull test, the point estimate ($\hat{\psi}$) is the value of ψ for which $Z(\psi) = 0$. For the non-parametric tests (log rank, Cox), $\hat{\psi}$ is the value of ψ for which $Z(\psi)$ crosses 0, since $Z(\psi)$ is a step function. Confidence intervals are similarly found with the $100(1-\alpha)\%$ confidence interval being the set $\{\psi : |Z(\psi)| < z_{1-\alpha/2}\}$, where $z_{1-\alpha/2}$ is the $1-\alpha/2$ percentile of the standard normal distribution.

After finding $\hat{\psi}$, an adjusted hazard ratio can be calculated. For example, by comparing counter-factual event times at $\hat{\psi}$ in the control group to the observed event times in the experimental group we can estimate the treatment effect that would have been observed in the absence of switching (in the case where only the control group switch).

As well as assuming that the only difference between randomised groups is the treatment received, the RPSFTM also assumes a 'common treatment effect'. The common treatment effect assumption states that the treatment effect is the same for all participants (with respect to time spent on treatment) regardless of when treatment is received, which is implicit in equation (1).

Re-censoring

We assume that censoring occurs if participants survive to a specific calendar date (Robins and Tsiatis, 1991). Thus the potential censoring time C_i is known for all participants. The censoring indicators of the observed event times are initially carried over to the counter-factual event times. However, the uninformative censoring on the T_i scale may be informative on the U_i scale. Suppose we have two participants with the same U_i, one of whom receives the superior treatment. The participant receiving

the superior treatment has their U_i extended so that they are censored whilst the other participant may observe the event.

In detail, C_i, the censoring times for the T_i, are transformed to $C_i\left(1 - P_i + P_i \exp\left(\psi\right)\right)$ when we work on the U_i scale, where P_i is a random variable, the proportion of time on treatment, $T_i^{\text{on}} / \left(T_i^{\text{on}} + T_i^{\text{off}}\right)$. This transformation of censoring times occurs by representing the censoring status in two equivalent ways: $\{C_i < T_i\}$, and re-scaling both sides $\{C_i\left(1 - P_i + P_i \exp\left(\psi\right)\right) < T_i\left(1 - P_i + P_i \exp(\psi)\right) = U_i\}$. We cannot assume the variable P_i is independent of U_i. For example, adherence to a protocolised treatment may depend on the underlying prognosis.

We can overcome this induced dependence by considering the *sample space* for P_i marginally over U_i and, optionally, conditioning on R_i. If we replace P_i with a function of its sample space then any dependency on U_i is thus removed. Any alternative transformed censoring times must be smaller than the original censoring times, else we may have to impute uncensored U_i values for censored T_i observations, which would be impossible. Hence taking the minimum value from the sample space for P_i meets both desiderata. If the sample space differs between arms, say switching is impossible in one arm, then this can be utilised to potentially observe more events with gains in efficiency.

Operationally, let C_i be the potential censoring time for participant i. A participant is then re-censored at the minimum possible censoring time:

$$D_i^*\left(\psi\right) = \min\left(C_i, C_i \exp\left(\psi\right)\right).$$

If $D_i^*\left(\psi\right) < U_i\left(\psi\right)$, then U_i is replaced by D_i^* and the censoring indicator is replaced by 0. For treatment arms where switching does not occur, there can be no informative censoring and so re-censoring is not applied.

Sensitivity analysis

As previously mentioned, the RPSFTM has two assumptions:

1. The only difference between randomised groups is the treatment received.
2. The treatment effect is the same for all participants regardless of when treatment is received.

Whilst the first assumption is plausible in a randomised controlled trial, the latter may be unlikely to hold. For example, if control group participants can only switch at disease progression then the treatment benefit may be different in these participants compared to those randomised to the experimental treatment. The `rpsftm()` function allows for investigation of deviations from the common treatment effect assumption by featuring a treatment-effect modifier variable which means the treatment effect can be varied across participants. The assumption that defines the counter-factual treatment-free event times U_i is thus modified by multiplying ψ by some factor $k_i > 0$:

$$U_i = T_i^{\text{off}} + T_i^{\text{on}} \exp\left(k_i \psi\right).$$

The value taken by k_i is derived from observed data and is left to the user to assign.

The package assumes that k_i is determined at baseline, and re-censoring is undertaken in a similar way by re-censoring at the minimum possible censoring time:

$$D_i^*\left(\psi\right) = \min\left(C_i, C_i \exp\left(k_i \psi\right)\right).$$

Again, if $D_i^*\left(\psi\right) < U_i\left(\psi\right)$, then U_i is replaced by D_i^* and the censoring indicator is replaced by 0. The case where k_i is observed post randomisation is equally valid in terms of defining U_i, but more complicated re-censoring may be required, taking the minimum value of D_i^* over the sample space of k_i conditional on arm. This is not implemented in the package.

Using package rpsftm

The main function in the package **rpsftm** is of the same name, `rpsftm()`, which returns an object that has `print`, `summary`, and `plot` methods, and that inherits from the class used to define the test statistic (`survdiff`, `coxph` or `survreg`) . The arguments to `rpsftm()` are given in table 1 below.

Example

The `rpsftm` function will be illustrated using a simulated dataset `immdef` taken from (White et al., 2002), based on a randomized controlled trial (Concorde Coordinating Committee, 1994). The trial

	`rpsftm()` arguments
`formula`	a formula with a minimal structure of `Surv(time,status)` ∼ `rand(arm,rx)` where • `arm` is the randomised treatment arm • `rx` is the proportion of time spent on treatment, taking values in $[0,1]$. Further terms can be added to the right hand side to adjust for covariates.
`data`	an optional data frame containing the variables
`censor_time`	variable or constant giving the time at which censoring would, or has occurred. This should be provided for all observations unlike standard Kaplan-Meier or Cox regression where it is only given for censored observations. If no value is given then re-censoring is not applied.
`subset`	an expression indicating which subset of the rows of data should be used in the fit. This can be a logical vector, a numeric vector indicating which observation numbers are to be included, or a character vector of row names to be included. All observations are included by default.
`na.action`	a missing-data filter function. This is applied to the `model.frame` after any subset argument has been used. Default is `options()$na.action`.
`test`	one of `survdiff`, `coxph` or `survreg`. Describes the test to be used in the estimating equation. Default is `survdiff`.
`low_psi`	the lower limit of the range to search for the causal parameter. Default is -1.
`hi_psi`	the upper limit of the range to search for the causal parameter. Default is 1.
`alpha`	the significance level used to calculate the confidence intervals. Default is 0.05.
`treat_modifier`	an optional variable that ψ is multiplied by on an participant observation level to give differing impact to treatment. Default is 1.
`autoswitch`	a logical to autodetect cases of no switching. Default is `TRUE`. If all observations in an arm have perfect compliance then re-censoring is not applied in that arm. If `FALSE` then re-censoring is applied regardless of perfect compliance.
`n_eval_z`	The number of points between `hi_psi` and `low_psi` at which to evaluate the Z-statistics in the estimating equation. Default is 100.

Table 1: Arguments for the `rpsftm()` function

compares two policies (immediate or deferred treatment) of zidovudine treatment in symptom free participants infected with HIV. The immediate treatment arm received treatment at randomisation whilst the deferred arm received treatment either at onset of AIDS related complex or AIDS (CDC group IV disease) or development of persistently low CD4 count. The endpoint considered here was time from study entry to progression to AIDS, or CDC group IV disease, or death.

Data

The `immdef` data frame has 1000 rows of 9 variables as described in table 2

	`immdef` variables
`id`	participant ID number
`def`	indicator that the participant was assigned to the Deferred treatment arm
`imm`	indicator that the participant was assigned to the Immediate treatment arm
`censyrs`	censoring time, in years, corresponding to the close of study minus the time of entry for each participant
`xo`	an indicator that switching occurred
`xoyrs`	the time, in years, from entry to switching, or 0 for participants in the Immediate arm
`prog`	an indicator of disease progression (1), or censoring (0)
`progyrs`	time, in years, from entry to disease progression or censoring
`entry`	the time of entry into the study, measured in years from the date of randomisation

Table 2: Description of simulated data set

The first six observations are given below

```
> library(rpsftm)
> head(immdef)
  id def imm censyrs xo    xoyrs prog  progyrs entry
1  1   0   1       3  0 0.000000    0 3.000000     0
2  2   1   0       3  1 2.652797    0 3.000000     0
3  3   0   1       3  0 0.000000    1 1.737838     0
4  4   0   1       3  0 0.000000    1 2.166291     0
5  5   1   0       3  1 2.122100    1 2.884646     0
6  6   1   0       3  1 0.557392    0 3.000000     0
```

For example, participant 2 was randomised to the deferred arm, started treatment at 2.65 years and was censored at 3 years (the end of the study). Subject 3 was randomised to the immediate treatment arm and progressed (observed the event) at 1.74 years. Subject 5 was randomised to the deferred treatment arm, started treatment at 2.12 years and progressed at 2.88 years. The trial lasted 3 years with staggered entry over the first 1.5 years. The variable censyrs gives the time from entry to the end of the trial.

Intention-to-treat analysis

First, we estimate the effect of giving zidovudine ignoring any treatment changes during the trial. This is found by fitting an accelerated failure time model using the **eha** package (Broström, 2017):

```
> library(eha)
> itt_fit <- aftreg(Surv(progyrs, prog) ~ imm, data = immdef)
> itt_fit
Call:
aftreg(formula = Surv(progyrs, prog) ~ imm, data = immdef)

Covariate          W.mean      Coef Time-Accn  se(Coef)    Wald p
imm                 0.510    -0.147     0.863     0.077     0.056

Baseline parameters:
```

```
log(scale)                    1.404              0.062     0.000
log(shape)                    0.392              0.052     0.000
Baseline life expectancy:

Events                    312
Total time at risk        1932.5
Max. log. likelihood      -855.14
LR test statistic         3.69
Degrees of freedom        1
Overall p-value           0.0548817
```

The intention-to-treat estimate is -0.147 which means that lifetime is used up $\exp(-0.147) = 0.863$ times as fast when on zidovudine as when off zidovudine (or zidovudine extends lifetime by a factor of $\exp(0.147) = 1.16$).

Fitting the RPSFTM

We now show how to use `rpsftm` with the `immdef` data. First, a variable `rx` for the proportion of time spent on treatment must be created:

```
rx <- with(immdef, 1 - xoyrs / progyrs)
```

This sets `rx` to 1 in the immediate treatment arm (since no participants could switch to the deferred arm, and `xoyrs = 0` in such participants), 0 in the deferred arm participants that did not receive treatment (as `xoyrs = progyrs` in such participants) and `1 -xoyrs / progyrs` in the deferred arm participants that did receive treatment. Using the default options, the fitted model is

```
rpsftm_fit_lr <- rpsftm(formula = Surv(progyrs, prog) ~ rand(imm, rx),
                        data = immdef,
                        censor_time = censyrs)
```

The above formula fits an RPSFTM where `progyrs` is the observed event time, `prog` is the indicator of disease progression, `imm` is the randomised treatment group indicator, `rx` is the proportion of time spent on treatment and `censyrs` is the censoring time. The log rank test is used in finding the point estimate of $\widehat{\psi}$. Re-censoring is performed since the `censor_time` parameter is specified; if not specified then re-censoring would not be performed. After finding $\widehat{\psi}$, `rpsftm` refits the model at $\widehat{\psi}$ and produces a `survdiff` object of the counter-factual event times to be used in plotting Kaplan-Meier curves. The list of objects that the function returns is given in table 3.

The point estimate and 95% confidence interval (CI) can be returned using `rpsftm_fit_lr$psi` and `rpsftm_fit_lr$CI` which gives $\widehat{\psi} = -0.181\ (-0.350, 0.002)$, slightly larger than in the intention-to-treat analysis. This means that lifetime is used up $\exp(-0.181) = 0.834$ times as fast when on zidovudine as when off zidovudine, i.e. the time to progression to AIDS, or CDC group IV disease, or death is longer when on zidovudine (treatment is beneficial). However, the confidence interval contains zero, which suggests the treatment effect is non-significant. The function `plot()` produces Kaplan-Meier curves of the counter-factual event times in each group and can be used to check that the distributions are indeed the same at $\widehat{\psi}$ as shown in figure 1

We now provide examples of using the Cox regression model and the Weibull model in place of the log rank test. To use the Wald test from a Cox regression model, we specify `test = coxph` in the function parameters. Covariates can also be included in the estimation procedure by adding them to the right hand side of the formula. For example, baseline covariates that are included in the intention-to-treat analysis may also be incorporated into the estimation procedure of the RPSFTM. In the following example we add entry time as a covariate and use `summary()` to find the value of $\widehat{\psi}$ and its 95% CI.

```
> rpsftm_fit_cph <- rpsftm(formula = Surv(progyrs, prog) ~ rand(imm, rx) + entry,
+                          data = immdef,
+                          censor_time = censyrs,
+                          test = coxph)
> summary(rpsftm_fit_cph)
   arm   rx.Min. rx.1st Qu. rx.Median   rx.Mean rx.3rd Qu.   rx.Max.
 1   0 0.0000000  0.0000000 0.0000000 0.1574062  0.2547779 0.9770941
 2   1 1.0000000  1.0000000 1.0000000 1.0000000  1.0000000 1.0000000
   n= 1000, number of events= 286

        coef exp(coef) se(coef)     z Pr(>|z|)
```

	rpsftm() outputs
psi	the estimated parameter
fit	a survdiff object to produce Kaplan-Meier curves of the estimated counter-factual event times in each treatment arm using plot()
CI	a vector of the confidence interval around psi
Sstar	the (possibly) re-censored Surv() data using the estimate value of psi to give counterfactual untreated failure times.
rand	the rand() object used to specify the allocated and observed amount of treatment.
ans	the values from uniroot.all used to solve the estimating equation, but embedded within a list as per uniroot, with an extra element root_all, a vector of all roots found in the case of multiple solutions. The first element of root_all is subsequently used.
eval_z	a data frame with the Z-statistics from the estimating equation evaluated at a sequence of values of psi. Used to plot and check if the range of values to search for solution and limits of confidence intervals need to be modified.
	Further elements corresponding to either a survdiff, coxph, or survreg object. This will always include:
call	the R call object
formula	a formula representing any adjustments, strata or clusters - used for the update() function
terms	a more detailed representation of the model formula

Table 3: Outputs from the rpsftm() function

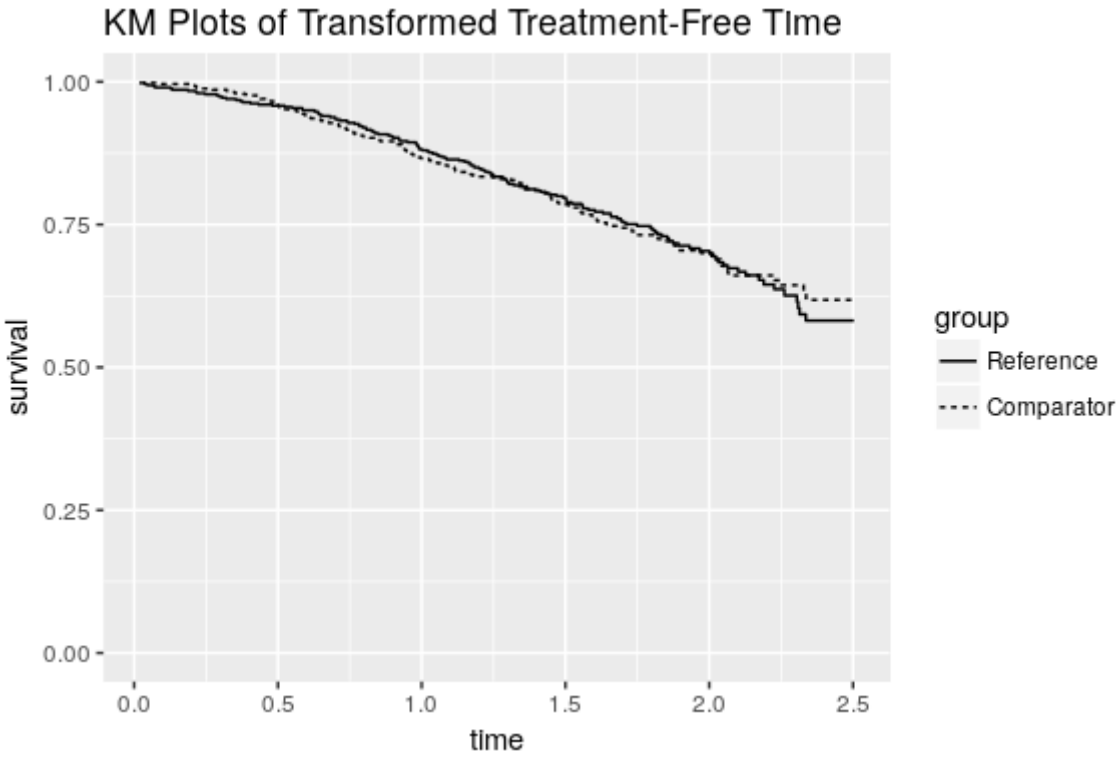


Figure 1: Output from plot(rpsftm_fit_lr)

```
 entry 0.1235    1.1315   0.1487 0.831    0.406

       exp(coef) exp(-coef) lower .95 upper .95
 entry     1.131     0.8838    0.8454     1.514

 Concordance= 0.514  (se = 0.018 )
 Rsquare= 0.001   (max possible= 0.976 )

 psi: -0.1811697
 exp(psi): 0.8342938
 Confidence Interval, psi -0.3496874 0.003370267
 Confidence Interval, exp(psi)  0.7049084 1.003376
```

For the Weibull model we specify `test = survreg` in the function parameters. :

```
> rpsftm_fit_wb <- rpsftm(formula = Surv(progyrs, prog) ~ rand(imm, rx) + entry,
+                         data = immdef,
+                         censor_time = censyrs,
+                         test = survreg)
> summary(rpsftm_fit_wb)
   arm   rx.Min. rx.1st Qu. rx.Median   rx.Mean rx.3rd Qu.   rx.Max.
 1   0 0.0000000  0.0000000 0.0000000 0.1574062  0.2547779 0.9770941
 2   1 1.0000000  1.0000000 1.0000000 1.0000000  1.0000000 1.0000000

 Call:
 rpsftm(formula = Surv(progyrs, prog) ~ rand(imm, rx) + entry,
     data = immdef, censor_time = censyrs, test = survreg)
              Value Std. Error      z        p
 (Intercept)  1.3881     0.0857 16.197 5.34e-59
 entry       -0.0582     0.0906 -0.642 5.21e-01
 Log(scale)  -0.4176     0.0568 -7.349 2.00e-13

 Scale= 0.659

 Weibull distribution
 Loglik(model)= -759.8   Loglik(intercept only)= -760
 Number of Newton-Raphson Iterations: 6
 n= 1000


 psi: -0.1811851
 exp(psi): 0.8342809
 Confidence Interval, psi -0.3501459 0.005170935
 Confidence Interval, exp(psi)  0.7045852 1.005184
```

As we can see from the output for the Cox and Weibull models, the estimates of ψ are similar to the estimate obtained from using the log rank test. Both fitted models could be used as inputs to the `plot()` function, which produce figures very similar to figure 1.

As a sensitivity analysis we can investigate what would happen to the estimate of ψ if the treatment effect in the deferred treatment group was half of that in the immediate treatment group by setting $k_i = 1$ for participants in the latter group and $k_i = 0.5$ for participants in the former group.

```
> weight <- with(immdef, ifelse(imm == 1, 1, 0.5))
> rpsftm(Surv(progyrs, prog) ~ rand(imm, rx), data = immdef, censor_time = censyrs,
+        treat_modifier = weight
+        )
   arm   rx.Min. rx.1st Qu. rx.Median   rx.Mean rx.3rd Qu.   rx.Max.
 1   0 0.0000000  0.0000000 0.0000000 0.1574062  0.2547779 0.9770941
 2   1 1.0000000  1.0000000 1.0000000 1.0000000  1.0000000 1.0000000
 Call:
 rpsftm(formula = Surv(progyrs, prog) ~ rand(imm, rx), data = immdef,
     censor_time = censyrs, treat_modifier = weight)

          N Observed Expected (O-E)^2/E (O-E)^2/V
```

```
 .arm=0 500      157      157  5.75e-07  1.22e-06
 .arm=1 500      143      143  6.31e-07  1.22e-06

  Chisq= 0  on 1 degrees of freedom, p= 0.999

 psi: -0.1704745
 exp(psi): 0.8432646
```

In this case the estimate of treatment effect is reduced slightly to -0.170 and lifetime is used up 0.843 times as fast when on treatment as when off treatment.

Trouble shooting

There is no guarantee that unique solutions exist to the estimating equations for the estimate and confidence interval limits, or that we have searched a wide enough interval to find them if they do exist and are unique.

There are three instances where `rpsftm` will produce warning messages due to the search interval. The function first evaluates $Z(\psi)$ at `low_psi` and `hi_psi` and will produce a warning message if $Z(\psi)$ is the same sign at these two points and no root exists within the interval.

```
Warning messages:
1: In rpsftm(formula = Surv(progyrs, prog) ~ rand(imm, rx), data = immdef,  :

The starting interval (-1, -0.9) to search for a solution for psi
gives values of the same sign (7.53, 6.88).
Try a wider interval. plot(obj$eval_z, type="s"), where obj is the output of rpsftm()
2: In rpsftm(formula = Surv(progyrs, prog) ~ rand(imm, rx), data = immdef,  :
  Evaluation of the estimated values of psi failed. It is set to NA
3: In rpsftm(formula = Surv(progyrs, prog) ~ rand(imm, rx), data = immdef,  :
  Evaluation of a limit of the Confidence Interval failed.  It is set to NA
4: In rpsftm(formula = Surv(progyrs, prog) ~ rand(imm, rx), data = immdef,  :
  Evaluation of a limit of the Confidence Interval failed.  It is set to NA
```

This suggests widening the search interval via trial and error until a root for ψ can be found between `low_psi` and `hi_psi`. The second warning message occurs when `uniroot.all`, the function used to solve the estimating equation for $\widehat{\psi}$ and its 95% confidence interval limits, fails to find any one of these. It will set the value to NA and produce the following warning message

```
> rpsftm_fit <- rpsftm(formula = Surv(progyrs, prog) ~ rand(imm, rx),
+                      data = immdef,
+                      censor_time = censyrs,
+                      low_psi = -1,
+                      hi_psi = -0.1)
Warning message:
In rpsftm(formula = Surv(progyrs, prog) ~ rand(imm, rx), data = immdef,  :
  Evaluation of a limit of the Confidence Interval failed.  It is set to NA
```

Investigation of a plot of $Z(\psi)$ against ψ in figure 2 for a range of values of ψ could show why the function fails to find a root. The fitted object, `rpsftm_fit`, returns a data frame `rpsftm_fit$eval_z` with values of the Z-statistic evaluated at 100 points between the limits of the search interval. The data frame can be supplied as the argument to `plot()` to visualise the estimating equation.

```
plot(rpsftm_fit$eval_z, type = "s", ylim = c(-2, 6))
abline(h = qnorm(c(0.025, 0.5, 0.975)))
abline(v = rpsftm_fit$psi)
abline(v = rpsftm_fit$CI)
```

In this case, we see that the search interval used was not wide enough to find the upper confidence limit. The third warning message occurs when multiple roots are found:

```
Warning message:
In root(0) : Multiple Roots found
```

In order to show what happens when multiple roots are found, a subset of the `immdef` dataset was created and is stored within the test area of **rpsftm**. Whilst the function `uniroot.all` will return

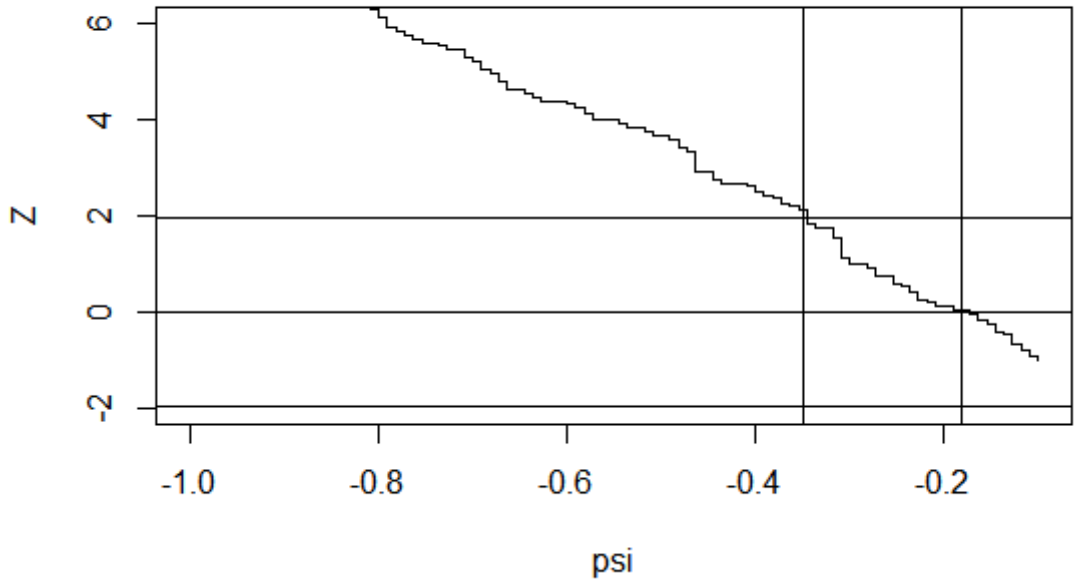


Figure 2: Plot of $Z(\psi)$ against ψ to detect problems with the choice of search interval

multiple roots, the function `rpsftm` will only display the first root found by `uniroot.all`. A plot of $Z(\psi)$ against ψ can be used to find the other roots.

The limits of the confidence intervals may not exist at all if $Z(\cdot)$ reaches an asymptote before it hits the required quantile of the standard normal distribution, in which case the limits should be reported as $\pm\infty$. This scenario would be illustrated by a similar plot to figure 2.

Another possibility is for the `coxph` function to fail to converge. This occurs when the maximum likelihood estimate of a coefficient is infinity, e.g. if one of the treatment groups has no events. The `coxph` documentation states that the Wald statistic should be ignored in this case and therefore the `rpsftm` output should be taken with caution.

Limitations

As well as the potential computational issues highlighted in the previous section, the RPSFTM itself has some limitations. The method relies on the assumption that the treatment effect is the same for all participants regardless of when treatment is received. We have allowed for investigation of deviations from this assumption within the `rpsftm()` function by adding a treatment-effect modifier variable. Another possible limitation of the model is its requirement for only the total amount of time spent on/off treatment. In instances where participants can switch back and forth between treatments the model may be inefficient.

Conclusion

The intention is to fill a void in the methodology available to R users by providing this package and thus facilitate the adoption of newer methods in application to real data in future. The sensitivity analyses, which generalise the definition of the counter-factual treatment-free event times, are an original development.

Further developments to the package may include expanding the functionality to allow multi-armed studies with three or more arms. Such an extension would have at its core a set of g-estimating equations to solve, one for each element of the vector of ψ parameters. Defining the syntax to capture the multi-dimensional metric of time on treatments is challenging, as will be the numerical computational details when the g-estimating equations are step functions.

Acknowledgement

Ian White was supported by the Medical Research Council Unit Programme MC_UU_12023/21. Simon Bond is a visiting worker at the MRC Biostatistics Unit, with primary affiliation to Cambridge Clinical Trials Unit. Annabel Allison contributed to the article initially whilst affiliated to the MRC Biostatistics Unit, but has since moved post.

Bibliography

S. Bond and A. Allison. *rpsftm: Rank Preserving Structural Failure Time Models*, 2017. URL https://CRAN.R-project.org/package=rpsftm. R package version 1.2.1. [p343]

G. Broström. *eha: Event History Analysis*, 2017. URL https://CRAN.R-project.org/package=eha. R package version 2.5.0. [p346]

Concorde Coordinating Committee. Concorde: MRC/ANRS Randomised Double-Blind Controlled Trial of Immediate and Deferred Zidovudine in Symptom-Free HIV Infection. *Lancet*, 343:871–881, 1994. URL https://doi.org/10.1016/s0140-6736(94)90006-x. [p344]

S. Dodd, I. R. White, and P. Williamson. Nonadherence to Treatment Protocol in Published Randomised Controlled Trials: a Review. *Trials*, 13(1):84, 2012. ISSN 1745-6215. URL https://doi.org/10.1186/1745-6215-13-84. [p342]

M. A. Hernán, B. Brumback, and J. M. Robins. Marginal structural models to estimate the joint causal effect of nonrandomized treatments. *Journal of the American Statistical Association*, 96(454):440–448, 2001. URL https://doi.org/10.1198/016214501753168154. [p342]

J. P. Higgins, J. J. Deeks, and D. G. Altman. *Special Topics in Statistics*. John Wiley & Sons, Chichester, UK, 2011. ISBN 9780470712184. URL https://doi.org/10.1002/9780470712184.ch16. [p342]

N. R. Latimer and K. R. Abrams. Adjusting Survival Time Estimates in the Presence of Treatment Switching. Technical Report July, NICE DSU Technical Support Document 16, 2014. URL http://scharr.dept.shef.ac.uk/nicedsu/technical-support-documents/treatment-switching-tsd/. [p343]

N. R. Latimer, K. Abrams, P. Lambert, M. Crowther, A. Wailoo, J. Morden, R. Akehurst, and M. Campbell. Adjusting for Treatment Switching in Randomised Controlled Trials - A Simulation Study and a Simplified Two-Stage Method. *Statistical Methods in Medical Research*, 2014. ISSN 0962-2802. URL https://doi.org/10.1177/0962280214557578. [p342]

N. R. Latimer, C. Henshall, U. Siebert, and H. Bell. Treatment switching: Statistical and decision-making challenges and approaches. *International journal of technology assessment in health care*, 32(3):160–166, 2016. URL https://doi.org/10.1017/S026646231600026X. [p342]

J. M. Robins and A. A. Tsiatis. Correcting for non-compliance in randomized trials using rank preserving structural failure time models. *Communications in Statistics Theory and Methods*, 20:2609–2631, 1991. URL https://doi.org/10.1080/03610929108830654. [p342, 343]

W. M. van der Wal and R. B. Geskus. ipw: An R package for inverse probability weighting. *Journal of Statistical Software*, 43(13):1–23, 2011. URL http://www.jstatsoft.org/v43/i13/. [p343]

C. Watkins, X. Huang, N. Latimer, Y. Tang, and E. J. Wright. Adjusting Overall Survival for Treatment Switches: Commonly Used Methods and Practical Application. *Pharmaceutical Statistics*, 12(6):348–357, 2013. ISSN 1539-1612. URL https://doi.org/10.1002/pst.1602. [p342]

I. R. White. Uses and Limitations of Randomization-Based Efficacy Estimators. *Statistical Methods in Medical Research*, 14(4):327–347, 2005. ISSN 0962-2802. URL https://doi.org/10.1191/0962280205sm406oa. [p342]

I. R. White, S. Walker, A. G. Babiker, and J. H. Darbyshire. Impact of treatment changes on the interpretation of the Concorde trial. *AIDS*, 11:999–1006, 1997. URL https://doi.org/10.1097/00002030-199708000-00008. [p342]

I. R. White, A. G. Babiker, S. Walker, and J. H. Darbyshire. Randomisation-based methods for correcting for treatment changes: Examples from the Concorde trial. *Statistics in Medicine*, 18:2617–2634, 1999. URL https://doi.org/10.1002/(sici)1097-0258(19991015)18:19<2617::aid-sim187>3.0.co;2-e. [p342]

I. R. White, S. Walker, and A. Babiker. Strbee: Randomization-based efficacy estimator. *The Stata Journal*, 2:140–150, 2002. [p343, 344]

Simon Bond
Cambridge Clinical Trials Unit
Cambridge University Hospitals Foundation NHS Trust, Hills Rd, Cambridge
UK
simon.bond@addenbrookes.nhs.uk

Ian R White
MRC Clinical Trials Unit at UCL
90 High Holborn, London
UK
ian.white@ucl.ac.uk

Annabel Allison
Cambridge Clinical Trials Unit
Cambridge University Hospitals Foundation NHS Trust, Hills Rd, Cambridge
UK
ara41@medschl.cam.ac.uk

anomalyDetection: Implementation of Augmented Network Log Anomaly Detection Procedures

by Robert J. Gutierrez, Bradley C. Boehmke, Kenneth W. Bauer, Cade M. Saie, Trevor J. Bihl

Abstract As the number of cyber-attacks continues to grow on a daily basis, so does the delay in threat detection. For instance, in 2015, the Office of Personnel Management discovered that approximately 21.5 million individual records of Federal employees and contractors had been stolen. On average, the time between an attack and its discovery is more than 200 days. In the case of the OPM breach, the attack had been going on for almost a year. Currently, cyber analysts inspect numerous potential incidents on a daily basis, but have neither the time nor the resources available to perform such a task. **anomalyDetection** aims to curtail the time frame in which anomalous cyber activities go unnoticed and to aid in the efficient discovery of these anomalous transactions among the millions of daily logged events by i) providing an efficient means for pre-processing and aggregating cyber data for analysis by employing a tabular vector transformation and handling multicollinearity concerns; ii) offering numerous built-in multivariate statistical functions such as Mahalanobis distance, factor analysis, principal components analysis to identify anomalous activity, iii) incorporating the pipe operator (`%>%`) to allow it to work well in the **tidyverse** workflow. Combined, **anomalyDetection** offers cyber analysts an efficient and simplified approach to break up network events into time-segment blocks and identify periods associated with suspected anomalies for further evaluation.

Introduction

Organizations worldwide rely heavily on the systems of cyberspace, and the wider Internet as a whole, for commerce, defense operations, infrastructure control systems, financial management, transportation, and other critical services. Unfortunately, the number of cyber-attacks are growing on a daily basis and the ability of organizations to spot anomalous cyber activitiy is becoming more and more delayed (Gutierrez et al., 2017). On average, the time between an attack and its discovery is more than 200 days (Koerner, 2016). In the case of the 2015 Office of Personnel Management breach, in which approximately 21.5 million individual records of Federal employees and contractors had been stolen, the attack had been going on for almost a year prior to the anomalous activity being identified (U.S. Office of Personnel Management, 2015).

Cyber analysts inspect numerous potential incidents on a daily basis, but lack the time and resources to scour the high volumes of data in an efficient manner to identify anomalous activity worth further investigation (Samuelson, 2016). Firewalls and intrusion detection and prevention systems (IDPS) are one line of defense in identifying and stopping suspicious internet traffic. When a suspicious event occurs, these devices generate a log file containing details of what preprogrammed rules were violated and how it was handled (Goodall et al., 2009). Such log files contain details of the event, (i.e. source and destination IP addresses, port numbers, and protocols), but not the packet and data that led to the event. A primary activity of cyber analysts is the analysis of these log files to detect anomalies (Gutierrez et al., 2017). Although a reduced form of anomalous cyber data, these data sets can still represent millions of cyberspace transactions per minute (Jayathilake, 2012). Unfortunately, analysis of these log files has, historically, been heavily manual in nature and often leverages subject matter expertise to find possible threats in logged events to further investigate (Goodall et al., 2009; Jayathilake, 2012; Samuelson, 2016). This approach is inefficient and can suffer from biased, subjective assessments (Zamani, 2013). Moreover, much of the related research has focused on anomaly detection at the device/software level (i.e. Lazarevic et al., 2003; Denning, 1987; Garcia-Teodoro et al., 2009), with little exploration into anomaly detection in the log files generated from the preexisting devices or software (i.e McDonald et al., 2012; Winding et al., 2006; Breier and Branišová, 2015). Consequently, efficient analytic approaches are desirable to help detect anomalous activity in cyber network log data (Gutierrez et al., 2017).

This research introduces the **anomalyDetection** package (Boehmke and Gutierrez, 2017) to provide cyber analysts efficient means for performing anomaly detection in log files. The purpose of the package is to make *identifying* abnormal activity more efficient so that cyber analysts can spend more time researching the potential threat. It is important to note that there is no guarantee that anomalous activity is evidence of malicious cyber activity; however, identification of anomalous activity provides cyber security experts a starting point in their search for undetected malicious activity. **anomalyDetection** simplifies this process.

This paper proceeds as follows. First, we introduce the **anomalyDetection** functions and the

security_logs data set that will be leveraged for the illustrative examples. Next, we demonstrate how **anomalyDetection** can pre-process a data set for follow on analysis. This includes converting a data set with non-numeric attributes into numeric data using a tabulated state vector approach. **anomalyDetection** further prepares this tabulated state vector by inspecting and correcting for multicollinearity. We then illustrate how **anomalyDetection** provides efficient multivariate statistical analysis processes to help identify anomalous activity. Last, we end with some concluding remarks.

anomalyDetection functions

anomalyDetection provides 13 functions to aid in the detection of potential cyber anomalies, which are listed in Table 1. The package also incorporates the pipe operator (%>%) from the **magrittr** package (Bache and Wickham, 2014) for streamlining function composition. To illustrate the functionality of **anomalyDetection** we will use the security_logs data that mimics common information that appears in security logs and comes with **anomalyDetection**. Note that we also load the **tidyverse** package (Wickham, 2017) for common manipulation and visualization tasks.

Table 1: anomalyDetection Functions

Function	Purpose
tabulate_state_vector()	Employs a tabulated vector approach to transform security log data into unique counts of data attributes based on time blocks.
block_inspect()	Creates a list where the original data has been divided into blocks denoted in the state vector.
mc_adjust()	Handles issues with multicollinearity.
mahalanobis_distance()	Calculates the distance between the elements in data and the mean vector of the data for outlier detection.
bd_row()	Indicates which variables in data are driving the Mahalanobis distance for a specific row, relative to the mean vector of the data.
horns_curve()	Computes Horn's Parallel Analysis to determine the factors to retain within a factor analysis.
factor_analysis()	Reduces the structure of the data by relating the correlation between variables to a set of factors, using the eigen-decomposition of the correlation matrix.
factor_analysis_results()	Provides easy access to factor analysis results.
kaisers_index()	Computes scores designed to assess the quality of a factor analysis solution. It measures the tendency towards unifactoriality for both a given row and the entire matrix as a whole.
principal_components()	Relates the data to a set of a components through the eigen-decomposition of the correlation matrix, where each component explains some variance of the data.
principal_components_results()	Provides easy access to principal component analysis results.
get_all_factors()	finds all factor pairs for a given integer.

```
library(tidyverse)
library(anomalyDetection)

security_logs
# A tibble: 300 × 10
#   Device_Vendor Device_Product Device_Action          Src_IP
#           <chr>          <chr>         <chr>           <chr>
# 1         McAfee            NSP       Attempt   223.70.128.61
# 2          CISCO            ASA       Failure 174.110.206.174
# 3            IBM          SNIPS       Success 174.110.206.174
# 4         McAfee            NSP       Success   227.12.127.87
# 5        Juniper            SRX       Success     28.9.24.154
# 6         McAfee            NSP       Success     28.9.24.154
# 7         McAfee            NSP       Attempt     28.9.24.154
# 8         McAfee            ePO       Attempt   223.70.128.61
# 9         McAfee            ePO       Attempt 174.110.206.174
```

```
# 10         CISCO            ASA       Attempt   227.12.127.87
# ... with 290 more rows, and 6 more variables: Dst_IP <chr>,
#   Src_Port <int>, Dst_Port <int>, Protocol <chr>, Country_Src <chr>,
#   Bytes_TRF <int>
```

Data pre-processing

In order to develop a statistical framework for firewall log analysis, data pre-processing is necessary prior to applying any multivariate analytic techniques. To assist in this process **anomalyDetection** offers two main approaches to pre-process log file data - aggregating the data into a tabulated state vector and managing multicollinearity concerns.

First, we can employ the tabulated vector approach introduced by Gutierrez et al. (2017). This approach transforms the security log data into unique counts of data attributes based on pre-defined time blocks. Therefore, as each time block is generated, the categorical fields are separated by their levels and a count of occurrences for each level are recorded into a vector. All numerical fields, such as bytes in and bytes out, are recorded as a summation within the time block. The result is what we call a *state vector matrix*.

Thus, for our `security_logs` data we can create our state vector matrix based on our data being divided into 10 time blocks. What results is the summary of instances for each categorical level in our data for each time block. Consequently, row one represents the first time block and there were 2 instances of CISCO as the device vendor, 1 instance of IBM, etc. By adjusting the `block_length`, `level_limit`, and `level_keep` arguments, the user can refine the level of aggregation and variables to retain and analyze.

```
tabulate_state_vector(security_logs, 10)
# A tibble: 30 × 43
#   CISCO   IBM Juniper McAfee `Palo Alto Networks`   NA1   ASA   ePO
#   <int> <int>   <int>  <int>                <int> <int> <int> <int>
# 1     2     1       1      6                    0     0     2     2
# 2     0     2       4      2                    2     0     0     2
# 3     2     4       2      2                    0     0     2     2
# 4     5     1       2      1                    1     0     5     1
# 5     3     1       1      3                    2     0     3     1
# 6     2     1       2      4                    1     0     2     1
# 7     2     2       1      3                    2     0     2     0
# 8     3     3       1      3                    0     0     3     2
# 9     0     1       4      3                    2     0     0     1
# 10    2     2       2      4                    0     0     2     3
# ... with 20 more rows, and 35 more variables: Firewall <int>, NSP <int>,
#   SNIPS <int>, SRX <int>, NA2 <int>, Attempt <int>, Failure <int>,
#   Success <int>, NA3 <int>, `174.110.206.174` <int>,
#   `223.70.128.61` <int>, `227.12.127.87` <int>, `28.9.24.154` <int>,
#   `89.130.69.91` <int>, NA4 <int>, `145.114.4.203` <int>,
#   `151.194.233.198` <int>, `219.142.109.8` <int>, `32.73.26.223` <int>,
#   `56.137.121.203` <int>, NA5 <int>, TCP <int>, UDP <int>, NA6 <int>,
#   China <int>, India <int>, Korea <int>, Netherlands <int>,
#   Russia <int>, `United Kingdom` <int>, US <int>, NA7 <int>,
#   Src_Port <int>, Dst_Port <int>, Bytes_TRF <int>
```

The state vector matrix provides us with a numerical construct to analyze our log file data; however, prior to proceeding with any multivariate statistical analyses we should inspect the state vector for multicollinearity, to avoid issues such as matrix singularity, rank deficiency, and strong correlation values, and remove any columns that pose an issue. We can use `mc_adjust()` to handle issues with multicollinearity by first removing any columns whose variance is close to or less than a minimum level of variance (`min_var`). Then, it removes linearly dependent columns. Finally, it removes any columns that have a high absolute correlation value equal to or greater than that defined by the user (`max_cor`).

```
(state_vec <- security_logs %>%
  tabulate_state_vector(10) %>%
  mc_adjust())
# A tibble: 30 × 26
#   CISCO   IBM Juniper McAfee `Palo Alto Networks`   ePO Attempt Failure
#   <int> <int>   <int>  <int>                <int> <int>   <int>   <int>
```

```
# 1      2     1       1      6                    0     2       5       1
# 2      0     2       4      2                    2     2       4       2
# 3      2     4       2      2                    0     2       3       3
# 4      5     1       2      1                    1     1       0       5
# 5      3     1       1      3                    2     1       3       3
# 6      2     1       2      4                    1     1       4       1
# 7      2     2       1      3                    2     0       3       3
# 8      3     3       1      3                    0     2       6       2
# 9      0     1       4      3                    2     1       4       4
# 10     2     2       2      4                    0     3       3       0
# ... with 20 more rows, and 18 more variables: `174.110.206.174` <int>,
#   `223.70.128.61` <int>, `227.12.127.87` <int>, `28.9.24.154` <int>,
#   `145.114.4.203` <int>, `151.194.233.198` <int>, `219.142.109.8` <int>,
#   `32.73.26.223` <int>, TCP <int>, China <int>, India <int>,
#   Korea <int>, Netherlands <int>, Russia <int>, `United Kingdom` <int>,
#   Src_Port <int>, Dst_Port <int>, Bytes_TRF <int>
```

By default, `mc_adjust()` removes *all* columns that violate the variance, dependency, and correlation thresholds. Alternatively, we can use `action = "select"` as an argument, which provides interactivity where the user can select the variables that violate the correlation threshold that they would like to remove.

Multivariate statistical analyses

With our data adjusted for multicollinearity we can now proceed with multivariate analyses to identify anomalies in our log file. First we'll use the `mahalanobis_distance()` function to compare the distance between each observation by its distance from the data mean, independent of scale. This is computed as

$$MD = \sqrt{(x - \bar{x})C^{-1}(x - \bar{x})} \tag{1}$$

where x is a vector of p observations, $x = (x_1, \ldots, x_p)$, $\bar{x}$ is the mean vector of the data, $\bar{x} = (\bar{x}_1, \ldots, \bar{x}_p)$, and C^{-1} is the inverse data covariance matrix. Here, we include `output = "both"` to return both the Mahalanobis distance and the absolute breakdown distances and `normalize = TRUE` so that we can compare relative magnitudes across our data.

```
state_vec %>%
  mahalanobis_distance("both", normalize = TRUE) %>%
  as_tibble
# A tibble: 30 × 27
#           MD    CISCO_BD      IBM_BD  Juniper_BD   McAfee_BD
#        <dbl>       <dbl>       <dbl>       <dbl>       <dbl>
# 1  0.4548638 0.005536981 0.013005457 0.016020254 0.021822626
# 2  4.3843567 0.024664733 0.005573767 0.044055697 0.008985787
# 3  0.3604934 0.005536981 0.042732214 0.004005063 0.008985787
# 4  0.8456701 0.050839552 0.013005457 0.004005063 0.016687890
# 5  3.3541555 0.020637838 0.013005457 0.016020254 0.001283684
# 6  1.0900635 0.005536981 0.013005457 0.004005063 0.006418419
# 7  0.6769615 0.005536981 0.005573767 0.016020254 0.001283684
# 8  0.6968967 0.020637838 0.024152991 0.016020254 0.001283684
# 9  0.9910771 0.024664733 0.013005457 0.044055697 0.001283684
# 10 5.7822393 0.005536981 0.005573767 0.004005063 0.006418419
# ... with 20 more rows, and 22 more variables: `Palo Alto
#   Networks_BD` <dbl>, ePO_BD <dbl>, Attempt_BD <dbl>, Failure_BD <dbl>,
#   `174.110.206.174_BD` <dbl>, `223.70.128.61_BD` <dbl>,
#   `227.12.127.87_BD` <dbl>, `28.9.24.154_BD` <dbl>,
#   `145.114.4.203_BD` <dbl>, `151.194.233.198_BD` <dbl>,
#   `219.142.109.8_BD` <dbl>, `32.73.26.223_BD` <dbl>, TCP_BD <dbl>,
#   China_BD <dbl>, India_BD <dbl>, Korea_BD <dbl>, Netherlands_BD <dbl>,
#   Russia_BD <dbl>, `United Kingdom_BD` <dbl>, Src_Port_BD <dbl>,
#   Dst_Port_BD <dbl>, Bytes_TRF_BD <dbl>
```

We can use this information in a modified heatmap visualization (Figure 1) to identify outlier values across our security log attributes and time blocks. Brighter columns represent time blocks

deserving greater attention and further investigation. Larger circles represent variables within a time block that have more anomalous activity. Thus, the larger and brighter the dot the more significant the outlier is and the more it deserves attention.

```
state_vec %>%
  mahalanobis_distance("both", normalize = TRUE) %>%
  as_tibble %>%
  dplyr::mutate(Block = 1:n()) %>%
  gather(Variable, BD, -c(MD, Block)) %>%
  ggplot(aes(factor(Block), Variable, color = MD, size = BD)) +
  geom_point()
```

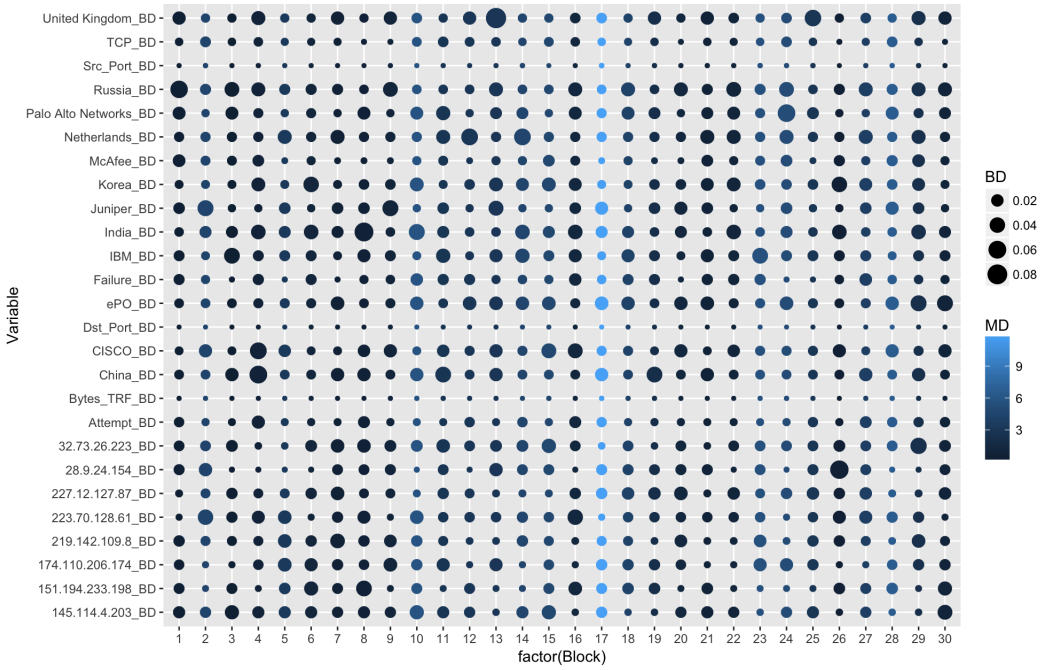


Figure 1: Modified heatmap for time block and feature outlier detection.

We can build onto this with the `bd_row()` function to identify which security log attributes in the data are driving the Mahalanobis distance. `bd_row()` measures the relative contribution of each variable, x_i, to MD by computing

$$BD_i = \left| \frac{x_i - \bar{x}_i}{\sqrt{C_{ii}}} \right| \tag{2}$$

where C_{ii} is the variance of x_i. Furthermore, `bd_row()` will look at a specified row and rank-order the columns by those that are driving the Mahalanobis distance. For example, the plot above identified block 17 as having the largest Mahalanobis distance suggesting some abnormal activity may be occurring during that time block. We can drill down into that block and look at the top 10 security log attributes that are driving the Mahalanobis distance as these may be areas that require further investigation.

```
state_vec %>%
  mahalanobis_distance("bd", normalize = TRUE) %>%
  bd_row(17, 10)
#      Src_Port_BD     Bytes_TRF_BD      Dst_Port_BD  32.73.26.223_BD
#        3.2733887        2.1016995        1.3575754        1.3398650
# 223.70.128.61_BD       McAfee_BD          IBM_BD         Korea_BD
#        1.2376147        1.0828415        1.0372208        0.9979392
#        Russia_BD       Juniper_BD
#        0.9937290        0.8478386
```

Next, we can use factor analysis as a dimensionality reduction technique to identify the underlying structure of the data and identify factors (features) in the data that appear abnormal. Factor analysis

relates the correlations between variables through a set of factors to link together seemingly unrelated variables. The basic factor analysis model is

$$X = \Lambda f + e \tag{3}$$

where X is the vector of responses $X = (x_1, \ldots, x_p)$, f are the common factors $f = (f_1, \ldots, f_q)$, e is the unique factors $e = (e_1, \ldots, e_p)$, and Λ is the factor loadings. Factor loadings are correlations between the factors and the original data and can thus range from -1 to 1, which indicate how much that factor affects each variable. Values close to 0 imply a weak effect on the variable. For the desired results, **anomalyDetection** uses the correlation matrix in its factor analysis computation.

A factor loadings matrix can be computed to understand how each original data variable is related to the resultant factors. This can be computed as

$$\Lambda = \left[\sqrt{\lambda_1} * e_1, \ldots, \sqrt{\lambda_p} * e_p \right] \tag{4}$$

where λ_1 is the eigenvalue for each factor, e_i is the eigenvector for each factor, and p is the number of columns. Factor scores are used to examine the behavior of the observations relative to each factor and can be used to identify anomaly detection. Factor scores are calculated as

$$\hat{f} = X_s R^{-1} \Lambda \tag{5}$$

where X_s is the standardized observations, R^{-1} is the inverse of the correlation matrix, and Λ is the factor loadings matrix. To simplify the results for interpretation, the factor loadings can undergo an orthogonal or oblique rotation. Orthogonal rotations assume independence between the factors while oblique rotations allow the factors to correlate. **anomalyDetection** utilizes the most common rotation option known as varimax. Varimax rotates the factors orthogonally to maximize the variance of the squared factor loadings which forces large factors to increase and small ones to decrease, providing easier interpretation.

To begin using factor analysis, the dimensions of the reduced state vector matrix are first passed to `horns_curve()`, which computes Horn's Parallel Analysis (Horn, 1965) to determine the factors to retain within a factor analysis.

```
horns_curve(state_vec)
# [1] 3.421431145 2.920860390 2.562571534 2.260732869 2.007756392
# [6] 1.789346403 1.595986779 1.418585893 1.255915765 1.106646364
# [11] 0.968567809 0.844067782 0.730563435 0.628690131 0.534405206
# [16] 0.450673920 0.374269067 0.306115502 0.244354685 0.191335630
# [21] 0.144295808 0.103873773 0.070143272 0.043042783 0.022553583
# [26] 0.008092665
```

Next, the dimensionality is determined by finding the eigenvalues of the correlation matrix of the state vector matrix and retaining only those factors whose eigenvalues are greater than or equal to those produced by `horns_curve()`. We use `factor_analysis()` to reduce the state vector matrix into resultant factors. The `factor_analysis()` function generates a list containing five outputs:

`$fa_loadings` numerical matrix with the original factor loadings

`$fa_scores` numerical matrix with the row scores for each factor

`$fa_loadings_rotated` numerical matrix with the varimax rotated factor loadings

`$fa_scores_rotated` numerical matrix with the row scores for each varimax rotated factor

`$num_factors` : numeric vector identifying the number of factors

```
state_vec %>%
  horns_curve() %>%
  factor_analysis(state_vec, hc_points = .) %>%
  names()
# [1] "fa_loadings"         "fa_scores"           "fa_loadings_rotated"
# [4] "fa_scores_rotated"   "num_factors"
```

For easy access to these results we can use the `factor_analysis_results()` parsing function. The `factor_analysis_results()` function will parse the results either by their list name or by location. For instance to extract the rotated factor scores you can use `factor_analysis_results(data, results = fa_scores_rotated)` or `factor_analysis_results(data, results = 4)` as demonstrated below.

```
state_vec %>%
  horns_curve() %>%
  factor_analysis(state_vec, hc_points = .) %>%
  factor_analysis_results(4) %>%
  as_tibble
# A tibble: 30 × 11
#             V1         V2         V3          V4          V5         V6
#          <dbl>      <dbl>      <dbl>       <dbl>       <dbl>      <dbl>
# 1  -0.362674625  0.3348386  0.6915550  0.08187007 -0.09327471  1.6577954
# 2  -0.006043119 -0.4959245 -1.7713249  1.48113530  0.75941566  0.4949437
# 3   0.783216952 -0.8394997 -0.4961150  1.31512814  0.97310016 -0.4546721
# 4   0.462460483  1.3648730 -0.1082969 -2.54886990 -0.42566777 -1.9486609
# 5   0.175061099 -0.8789010  1.2417347 -0.36983659  1.55454286 -0.5180766
# 6  -0.207615911  1.2025271  0.2300188  0.40146898  1.23209508  0.3752129
# 7   0.731099082 -1.9734310 -0.9490889 -0.62509695  1.04029889 -0.3571424
# 8   0.030558029 -1.2308883  1.1503857  0.08635927 -1.60839067  1.9569930
# 9   0.265720779 -0.0830922 -1.7467551 -0.13258281  0.47299019 -0.4580762
# 10 -0.875234891 -1.5917696  0.8289798 -1.23316029  1.25799551  0.3656262
# ... with 20 more rows, and 5 more variables: V7 <dbl>, V8 <dbl>,
#   V9 <dbl>, V10 <dbl>, V11 <dbl>
```

To evaluate the quality of a factor analysis solution, Kaiser (Kaiser, 1974) proposed the Index of Factorial Simplicity (IFS). The IFS is computed as

$$IFS = \frac{\sum_i \left[q \sum_s \lambda_{js}^4 - (\sum_s \lambda_{js}^2)^2\right]}{\sum_i \left[(q-1)(\sum_s \lambda_{js}^2)^2\right]} \tag{6}$$

where q is the number of factors, j the row index, s the column index, and λ_{js} is the value in the loadings matrix. Furthermore, Kaiser created the following evaluations of the score produced by the IFS as shown below:

In the .90s Marvelous

In the .80s Meritorious

In the .70s Middling

In the .60s Mediocre

In the .50s Miserable

Less than .50 : Unacceptable

Thus, to assess the quality of our factor analysis results we apply `kaisers_index()` to the rotated factor loadings and, as the results show below, our output value of 0.702 suggests that our results are "middling".

```
state_vec %>%
  horns_curve() %>%
  factor_analysis(data = state_vec, hc_points = .) %>%
  factor_analysis_results(fa_loadings_rotated) %>%
  kaisers_index()
# [1] 0.7018006
```

Furthermore, Figure 2 visualizes the factor analysis results to show the correlation between the columns of the reduced state vector to the rotated factor loadings. Strong negative correlations are depicted as red while strong positive correlations are shown as blue. This helps to identify which factors are correlated with each security log data attribute. Furthermore, this helps to identify two or more security log data attributes that appear to have relationships with their occurrences. For example, this shows that Russia is highly correlated with IP address `223.70.128` since both these attributes are strongly correlated with factor 5. If there is an abnormally large amount of instances with Russian occurrences this would be the logical IP address to start investigating.

```
fa_loadings <- state_vec %>%
  horns_curve() %>%
  factor_analysis(state_vec, hc_points = .) %>%
  factor_analysis_results(fa_loadings_rotated)

row.names(fa_loadings) <- colnames(state_vec)
```

```
gplots::heatmap.2(fa_loadings, dendrogram = 'both', trace = 'none',
            density.info = 'none', breaks = seq(-1, 1, by = .25),
            col = RColorBrewer::brewer.pal(8, 'RdBu'))
```

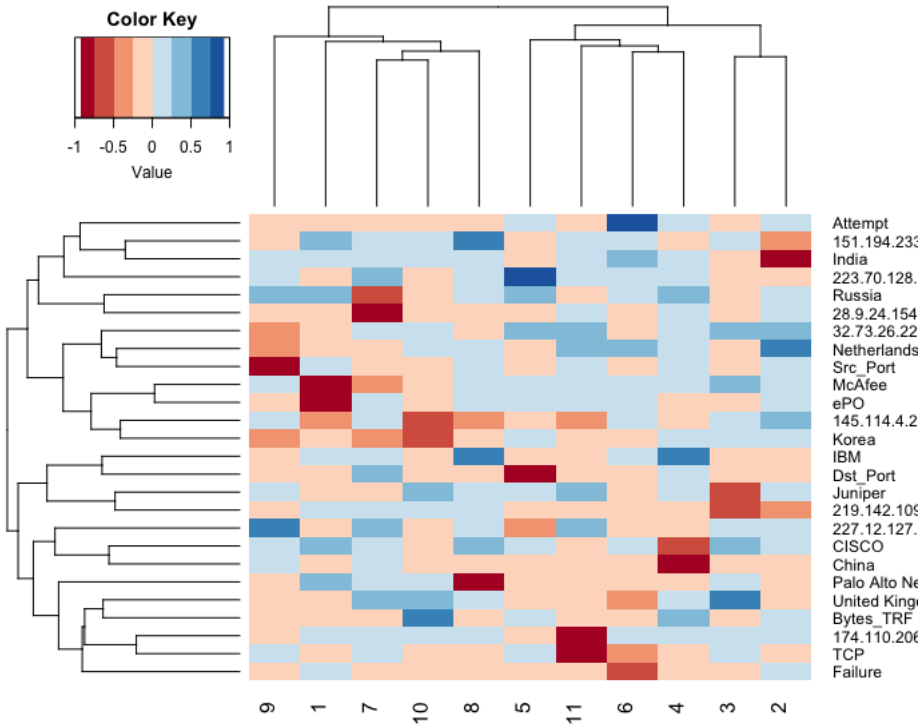


Figure 2: Modified factor analysis heatmap to identify correlated attributes.

We can also visualize the rotated factor score plots as in Figure 3 to see which time blocks appear to be outliers and deserve closer attention.

```
state_vec %>%
  horns_curve() %>%
  factor_analysis(state_vec, hc_points = .) %>%
  factor_analysis_results(fa_scores_rotated) %>%
  as_tibble() %>%
  dplyr::mutate(Block = 1:n()) %>%
  gather(Factor, Score, -Block) %>%
  dplyr::mutate(Absolute_Score = abs(Score)) %>%
  ggplot(aes(Factor, Absolute_Score, label = Block)) +
  geom_text(size = 2) +
  geom_boxplot(outlier.shape = NA)
```

This allows us to look across the factors and identify outlier blocks that may require further intra-block analysis. If we assume that an absolute rotated factor score ≥ 2 represents our outlier cut-off then we see that time blocks 4, 13, 15, 17, 24, 26, and 27 require further investigation. We saw block 17 being highlighted with `mahalanobis_distance()` earlier, but these other time blocks were not as obvious, so by performing and comparing these multiple anomaly detection approaches we can gain greater insights or confirm prior suspicions.

An alternative, yet similar approach to factor analysis is principal component analysis. These two approaches can produce similar outcomes, especially when the error component in equation 4 is close to zero (Fabrigar et al., 1999). However, the results often differ and there are important distinctions in the interpretation of these results (Park et al., 2002). First, a primary difference between the two approaches is that factor analysis estimates errors while principal component analysis does not. This indicates that principal component analysis assumes that the measurement is without error. Second, the goal in factor analysis is to explain the covariances or correlations between the variables. Therefore anomaly detection using factor analysis will identify time blocks and variable attributes in which their

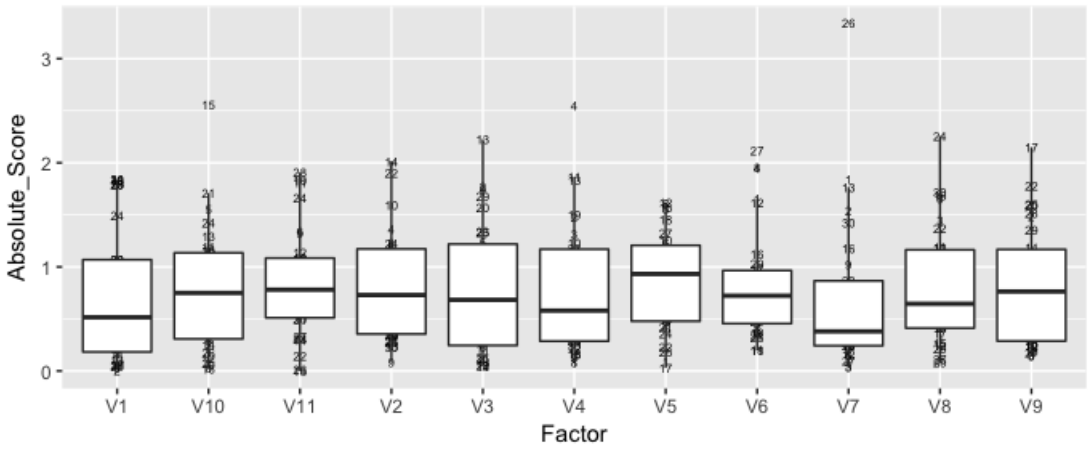


Figure 3: Detecting outlier time blocks based on rotated factor analysis scores.

abnormal behavior is highly correlated to one another. This allows you to identify latent features in the data. By contrast, the goal of principal component analysis is to explain as much of the total variance in the variables as possible. Therefore, if your goal is to reduce the log file variables into a composite component for further analysis, principal component analysis would be appropriate. To maintain clarity between these two approaches the following discussion leverages different notation.

The first principal component of a set of features $X_1, X_2, \ldots, X_p$ is the normalized linear combination of the features

$$Z_1 = \phi_{11}X_1 + \phi_{21}X_2 + \cdots + \phi_{p1}X_p \tag{7}$$

that has the largest variance. By *normalized*, we mean that $\sum_{j=1}^{p} \phi_{j1}^2 = 1$. We refer to the elements $\phi_{11}, \ldots, \phi_{p1}$ as the loadings of the first principal component; together, the loadings make up the principal component loading vector, $\phi_1 = (\phi_{11}, \phi_{21}, \ldots, \phi_{p1})^T$. The loadings are constrained so that their sum of squares is equal to one, since otherwise setting these elements to be arbitrarily large in absolute value could result in an arbitrarily large variance. After the first principal component Z_1 of the features has been determined, we can find the second principal component Z_2. The second principal component is the linear combination of $X_1, \ldots, X_p$ that has maximal variance out of all linear combinations that are uncorrelated with Z_1. The second principal component scores $z_{12}, z_{22}, \ldots, z_{n2}$ take the form

$$z_{12} = \phi_{12}x_{i1} + \phi_{22}x_{i2} + \cdots + \phi_{p2}x_{ip} \tag{8}$$

where ϕ_2 is the second principal loading vector, with elements $\phi_{12}, \phi_{22}, \ldots, \phi_{p2}$. This continues until all principal components have been computed. Therefore anomaly detection using PCA will maximize the difference in behaviors across time blocks and variable attributes. Thus, identifying anomalies with PCA will identify those attributes that behave very differently than all the other features. To perform a principal components analysis we use `principal_components()` which will create a list containing:

`$pca_sdev` the standard deviations of the principal components (i.e., the square roots of the eigenvalues of the covariance/correlation matrix, though the calculation is actually done with the singular values of the data matrix).

`$pca_loadings` the matrix of variable loadings (i.e., a matrix whose columns contain the eigenvectors).

`$pca_rotated` the value of the rotated data (the centered, and scaled if requested, data multiplied by the rotation matrix) is returned.

`$pca_center` the centering used.

`$pca_scale` a logical response indicating whether scaling was used.

```
principal_components(state_vec) %>% names
# [1] "pca_sdev"     "pca_loadings" "pca_rotated"  "pca_center"
# [5] "pca_scale"
```

For easy access to these results we can use the `principal_components_result` parsing function. The `principal_components_result` will parse the results either by their list name or by location. For example, to extract the computed component scores as outlined in Eq. 8 you can use

`principal_components_result(data, results = pca_rotated)` or `principal_components_result(data, results = 3)` as demonstrated below.

```
state_vec %>%
  principal_components() %>%
  principal_components_result(pca_rotated) %>%
  as_tibble
# A tibble: 30 × 26
#          PC1        PC2       PC3        PC4        PC5        PC6
#        <dbl>      <dbl>     <dbl>      <dbl>      <dbl>      <dbl>
# 1    1326.436 -285.25443  36.72628  3.2968478 -0.2977835 -1.3016917
# 2   20404.603  420.02358 236.94988  3.8917064  2.1798436  2.4849683
# 3    1884.370 -229.68499  45.39825  1.2682210  2.4212424  1.2380853
# 4    1555.892  171.11547 116.14809 -6.0588320 -0.5539837 -0.1773544
# 5  -37890.373 -470.91871  55.73837 -0.9419323 -0.3674156 -0.7529821
# 6  -19041.547 -307.40718 -44.48265  0.3985038 -1.2587089 -2.3739508
# 7  -19117.070  130.64729  30.69814 -2.1478778  3.1516592  1.0870282
# 8   21346.072   35.75772  12.44375  1.5784997  2.8835997 -2.4794129
# 9  -18901.619 -251.41102  55.19655  0.1510258 -0.3283588  4.0431385
# 10 -37321.917 -848.69603 -77.71478  2.1905376  2.3692681 -0.3439843
# ... with 20 more rows, and 20 more variables: PC7 <dbl>, PC8 <dbl>,
#   PC9 <dbl>, PC10 <dbl>, PC11 <dbl>, PC12 <dbl>, PC13 <dbl>, PC14 <dbl>,
#   PC15 <dbl>, PC16 <dbl>, PC17 <dbl>, PC18 <dbl>, PC19 <dbl>,
#   PC20 <dbl>, PC21 <dbl>, PC22 <dbl>, PC23 <dbl>, PC24 <dbl>,
#   PC25 <dbl>, PC26 <dbl>
```

We can then follow the principal components analysis with similar visualization activities as performed post-factor analysis to identify features that exhibit abnormal behavior. Since visualizing principal components analysis to identify anomolies mirrors that which we performed in the factor analysis section, we will leave this to the reader as an independent exercise.

Conclusion

Cyber attacks continue to be a growing concern for organizations. Unfortunately, the process of analyzing log files has, historically, been unorganized and lacked efficient approaches. The presented **anomalyDetection** package makes the log file analysis process more efficient and facilitates the identification and analysis of anomalies within log files.

First, **anomalyDetection** improves the pre-processing of cyber data. The package offers functions that help to narrow down abnormal behavior by aggregating internet traffic data into customizable time blocks. Aggregated activity at a higher-level time block should be easier to analyze while still offering a map to suspicious areas to drill down using smaller time blocks. For very large data sets, the analyst can tune the function parameters to start with less blocks and more aggregated data and then iteratively drill down into less aggregated data. Furthermore, **anomalyDetection** improves the process of adjusting for multicollinearity concerns.

Second, **anomalyDetection** improves the modeling process to perform multivariate statistical analysis by offering built-in functions to perform Mahalanobis distance, factor analysis, and principal components analysis along with functions to improve the efficiency of extracting and assessing the results from these multivariate approaches.

Third, we demonstrated how the **anomalyDetection** incorporates the pipe operator (%>%) to allow it to work well in the **tidyverse** workflow, which helps to improve the overall efficiency of the data analysis process.

It is also important to note that although the authors' focus with this package was to target and improve the analysis of network log-file data, **anomalyDetection** can also be used for other large data sets that contain arbitrary features that require data aggregation and anomaly analysis. Furthermore, as with any package, we readily admit that further improvements to the package can be made. Future versions of **anomalyDetection** plan to integrate additional multivariate and time series approaches to offer analysts a wider suite of modeling tools. Integrating plotting functions are also planned for future iterations to further enhance the efficiency of visualizing analytic results. Furthermore, future updates could explore how **anomalyDetection** could interact with distributable systems such as spark by integrating capabilities from packages such as **SparkR**. This would improve **anomalyDetection**'s ability to work with Big data architectures.

Bibliography

S. M. Bache and H. Wickham. *magrittr: A Forward-Pipe Operator for R*, 2014. URL https://CRAN.R-project.org/package=magrittr. R package version 1.5. [p355]

B. Boehmke and R. Gutierrez. *anomalyDetection: Implementation of Augmented Network Log Anomaly Detection Procedures*, 2017. URL https://CRAN.R-project.org/package=anomalyDetection. R package version 0.1.1. [p354]

J. Breier and J. Branišová. Anomaly detection from log files using data mining techniques. In *Information Science and Applications*, pages 449–457. Springer, 2015. doi: 10.1007/978-3-662-46578-3_53. URL https://doi.org/10.1007/978-3-662-46578-3_53. [p354]

D. E. Denning. An intrusion-detection model. *IEEE Transactions on software engineering*, (2):222–232, 1987. doi: 10.1109/TSE.1987.232894. URL https://doi.org/10.1109/TSE.1987.232894. [p354]

L. R. Fabrigar, D. T. Wegener, R. C. MacCallum, and E. J. Strahan. Evaluating the use of exploratory factor analysis in psychological research. *Psychological methods*, 4(3):272, 1999. [p361]

P. Garcia-Teodoro, J. Diaz-Verdejo, G. Maciá-Fernández, and E. Vázquez. Anomaly-based network intrusion detection: Techniques, systems and challenges. *computers & security*, 28(1):18–28, 2009. doi: 10.1016/j.cose.2008.08.003. URL https://doi.org/10.1016/j.cose.2008.08.003. [p354]

J. R. Goodall, W. G. Lutters, and A. Komlodi. Developing expertise for network intrusion detection. *Information Technology & People*, 22(2):92–108, 2009. doi: 10.1108/09593840910962186. URL http://dx.doi.org/10.1108/09593840910962186. [p354]

R. Gutierrez, K. Bauer, B. Boehmke, C. Saie, and T. Bihl. Cyber anomaly detection: Using tabulated vectors and embedded analytics for efficient data mining. *Journal of Algorithms and Computational Technology*, forthcoming, 2017. [p354, 356]

J. L. Horn. A rationale and test for the number of factors in factor analysis. *Psychometrika*, 30(2):179–185, 1965. doi: 10.1007/BF02289447. URL https://doi.org/10.1007/BF02289447. [p359]

D. Jayathilake. Towards structured log analysis. In *Computer Science and Software Engineering (JCSSE), 2012 International Joint Conference on*, pages 259–264. IEEE, 2012. doi: 10.1109/JCSSE.2012.6261962. URL https://doi.org/10.1109/JCSSE.2012.6261962. [p354]

H. F. Kaiser. An index of factorial simplicity. *Psychometrika*, 39(1):31–36, 1974. doi: 10.1007/BF02291575. URL https://doi.org/10.1007/BF02291575. [p360]

B. I. Koerner. Inside the cyberattack that shocked the US government. *Wired.com*, 2016. URL https://www.wired.com/2016/10/inside-cyberattack-shocked-us-government/. [p354]

A. Lazarevic, L. Ertoz, V. Kumar, A. Ozgur, and J. Srivastava. A comparative study of anomaly detection schemes in network intrusion detection. In *Proceedings of the 2003 SIAM International Conference on Data Mining*, pages 25–36. SIAM, 2003. doi: 10.1137/1.9781611972733.3. URL http://dx.doi.org/10.1137/1.9781611972733.3. [p354]

J. T. McDonald, E. D. Trias, M. R. Grimaila, J. Myers, R. F. Mills, and G. Peterson. Design and analysis of a dynamically configured log-based distributed security event detection methodology. *The Journal of Defense Modeling and Simulation*, 9(3):219–241, 2012. doi: 10.1177/1548512911399303. URL https://doi.org/10.1177/1548512911399303. [p354]

H. S. Park, R. Dailey, and D. Lemus. The use of exploratory factor analysis and principal components analysis in communication research. *Human Communication Research*, 28(4):562–577, 2002. [p361]

D. A. Samuelson. Using big data in cybersecurity. *ORMS Today*, 43(5), 2016. [p354]

U.S. Office of Personnel Management. Cybersecurity resource center. Technical report, U.S. Office of Personnel Management, mar 2015. URL https://www.opm.gov/cybersecurity/cybersecurity-incidents. [p354]

H. Wickham. *tidyverse: Easily Install and Load 'Tidyverse' Packages*, 2017. URL https://CRAN.R-project.org/package=tidyverse. R package version 1.1.1. [p355]

R. Winding, T. Wright, and M. Chapple. System anomaly detection: Mining firewall logs. In *Securecomm and Workshops, 2006*, pages 1–5. IEEE, 2006. doi: 10.1109/SECCOMW.2006.359572. URL https://doi.org/10.1109/SECCOMW.2006.359572. [p354]

M. Zamani. Machine learning techniques for intrusion detection. *CoRR*, abs/1312.2177, 2013. URL http://arxiv.org/abs/1312.2177. [p354]

Robert J. Gutierrez
Air Force Institute of Technology
2950 Hobson Way
Wright-Patterson AFB, OH 45433
rjgutierrez2015@gmail.com

Bradley C. Boehmke
Air Force Institute of Technology
2950 Hobson Way
Wright-Patterson AFB, OH 45433
bradleyboehmke@gmail.com

Kenneth W. Bauer
Air Force Institute of Technology
2950 Hobson Way
Wright-Patterson AFB, OH 45433
kenneth.bauer@afit.edu

Cade M. Saie
Air Force Institute of Technology
2950 Hobson Way
Wright-Patterson AFB, OH 45433
cade.saie@gmail.com

Trevor J. Bihl
Air Force Institute of Technology
2950 Hobson Way
Wright-Patterson AFB, OH 45433
trevor.bihl@gmail.com

Simulating Noisy, Nonparametric, and Multivariate Discrete Patterns

by Ruby Sharma, Sajal Kumar, Hua Zhong and Mingzhou Song

Abstract Requiring no analytical forms, nonparametric discrete patterns are flexible in representing complex relationships among random variables. This makes them increasingly useful for data-driven applications. However, there appears to be no software tools for simulating nonparametric discrete patterns, which prevents objective evaluation of statistical methods that discover discrete relationships from data. We present a simulator to generate nonparametric discrete functions as contingency tables. User can request strictly many-to-one functional patterns. The simulator can also produce contingency tables representing dependent non-functional and independent relationships. An option is provided to apply random noise to contingency tables. We demonstrate the utility of the simulator by showing the advantage of the FunChisq test over Pearson's chi-square test in detecting functional patterns. This simulator, implemented in the function `simulate_tables` in the R package **FunChisq** (version 2.4.0 or greater), offers an important means to evaluate the performance of nonparametric statistical pattern discovery methods.

Introduction

The demand for pattern discovery methodologies has elevated as massive automatic data collection takes place in every application domain. Consequently, an increasingly critical task in data science is to design effective algorithms to recognize complex meaningful patterns from data. Nonparametric discrete patterns do not require an analytical form, allowing them to flexibly represent functional and associative relationships among random variables. This makes them appealing to data-driven capture of complex relationships from big data sources. Indeed, many classical statistical methods can detect associative patterns between discrete random variables, including the widely used Pearson's chi-square test (Pearson, 1900), Fisher's exact test (Fisher, 1922), *G*-test (McDonald, 2014), and Barnard's test (Barnard, 1945).

However, these methods are not specifically designed for detecting functional patterns, where a dependent variable is a mathematical function of other independent variables. Functional relationships are considered powerful indicators of causality (Simon and Rescher, 1966). Recent methods for detecting discrete functional patterns such as the functional chi-square test (FunChisq) (Zhang and Song, 2013) have demonstrated their effectiveness in identifying causal molecular interactions in human breast cancer from both real and simulated protein abundance data (Hill et al., 2016). To evaluate such methods, software tools that can randomly generate diverse functional patterns are necessary. Such computer programs are unavailable as far as we are aware, in spite of several R functions for generating other types of random contingency table. The function `r2dtable` in the base package **stats** creates random two-way tables with given marginals using Patefield's algorithm (Patefield, 1981) under product-multinomial sampling. The package **rTableICC** (Demirhan, 2016) produces random $2 \times 2 \times K$ tables and $R \times C$ tables based on either intraclass-correlated or uncorrelated individuals. No utility known to us can generate noisy, random, and nonparametric discrete patterns that satisfy requirements on functional relationships between the row and column variables.

To address this gap, we present a new simulator to generate discrete functional patterns. This simulator is publicly available and implemented as the R function `simulate_tables` within the R package **FunChisq** ($\geq$2.4.0) (Zhang et al., 2017), which also contains statistical hypothesis testing methods for non-parametric functional dependencies using asymptotic chi-square or exact distributions. Functional chi-squares are asymmetric and functionally optimal, unique from other related statistics. The simulator can generate associative two-way contingency tables to depict several types of relationships. A relationship type can be a combination of statistical dependency and mathematical functionality. In a dependent relationship, the outcomes of two random variables are statistically dependent; whereas in an independent relationship, the outcomes of two random variables are statistically independent of each other. Dependency can be further categorized as functional (Bartle, 1964) or non-functional. The simulator is also capable of generating many-to-one functional patterns whose inverse is non-functional. To emulate real data often tainted with noise, our simulator can apply a controllable level of house noise (Zhang et al., 2015) into the contingency tables. This option enables one to evaluate the robustness of a discrete pattern discovery algorithm subject to noise. We utilized the simulator to contrast FunChisq and Pearson's chi-square tests and demonstrate their expected difference in detecting discrete functional relationships. We also report the runtime of the simulator as a function of sample size with and without noise. Finally, we illustrated how to use this computer program to generate simulated data for weather forecasting, drug development, molecular biology,

and statistical algorithm design. This discrete data simulator thus can serve as a benchmarking tool for the development of new statistical methods for pattern discovery in various scientific fields.

Generating random contingency tables with required discrete patterns

The simulator randomly samples contingency tables with specified functional and statistical patterns and also has an option to apply additional noise on the tables. Let $X \in \{1, \ldots, r\}$ be a discrete random variable with r levels, and $Y \in \{1, \ldots, c\}$ be a second discrete random variable with c levels. We use $p_{XY}(i, j)$ to represent the joint probability mass function of X and Y, and $p_X(i)$ and $p_Y(j)$ for the marginal probability mass functions of X and Y, respectively. We use the notation $Y = f(X)$ to indicate that Y is some function of X, and the notation $X \neq g(Y)$ to indicate that X is not any function of Y. We use the notation $X \perp Y$ to indicate that X and Y are statistically independent.

We define four pattern types formed by X and Y as follows:

- *Functional*—Y is a function of X;
- *Many-to-one*—Y is a function of X, but not vice versa. A discrete non-monotonic function is not necessarily many-to-one;
- *Dependent non-functional*—Y and X are statistically dependent, but not a function of each other;
- *Independent*—Y and X are statistically independent.

For all functional patterns, we do not consider the special case where X or Y can take only one value ($r = 1$ or $c = 1$), as in constant functions. We do not include the pattern type where Y is not a function of X but X is a function of Y, as it can be generated easily by transposing a many-to-one functional pattern. These four patterns are characterized by the mathematical and statistical relations between X and Y in Table 1.

Pattern	Is $Y = f(X)$?	Is $X = g(Y)$?	Is $X \perp Y$?
Functional	True	True or False	False
Many-to-one	True	False	False
Dependent non-functional	False	False	False
Independent	False	False	True

Table 1: Four pattern types between two random variables X and Y. Here functional patterns do not include constant functional patterns.

The simulator will generate three related tables in order: a noise-free sampled contingency table, a pattern table, and a noisy contingency table. All tables are $r \times c$, where X and Y are the row and column variables, respectively. Here is an explanation of these tables:

- *The sampled contingency table*—With given sample size n and row and/or column marginal probability mass functions, this table satisfies both the functional and statistical requirements. This table, being noise free, can be used to simulate noisy versions and evaluate the performance of pattern discovery algorithms on their statistical effectiveness.
- *The binary pattern table*—This table is created by setting all non-zero entries in the sampled contingency table to 1. Thus values in the pattern table are either 0 or 1. The table strictly satisfies the mathematical relationship for a given pattern type requested by the user, but it does not meet the statistical requirements. It can be used as the ground truth or gold standard for benchmarking how well pattern discovery algorithms can uncover the mathematical relationships.
- *The noisy contingency table*—At a user-specified noise level, this table is the noisy version of the sampled contingency table. Due to the added noise, this table may no longer strictly satisfy the required functional or statistical relationships. This table is the main output to be used for the evaluation of a discrete pattern discovery algorithm.

Figure 1 illustrates the four pattern types by sampled contingency tables generated using the simulator. The tables are rotated so that the horizontal axis represents the row variable X.

Next, we describe how to generate the sampled contingency tables for each type of discrete pattern. All pattern types require the common input of sample size n and table size $r \times c$. They differ in which marginal probability mass functions, p_X, p_Y, or both, must be provided. Let n_{ij} be the count in the

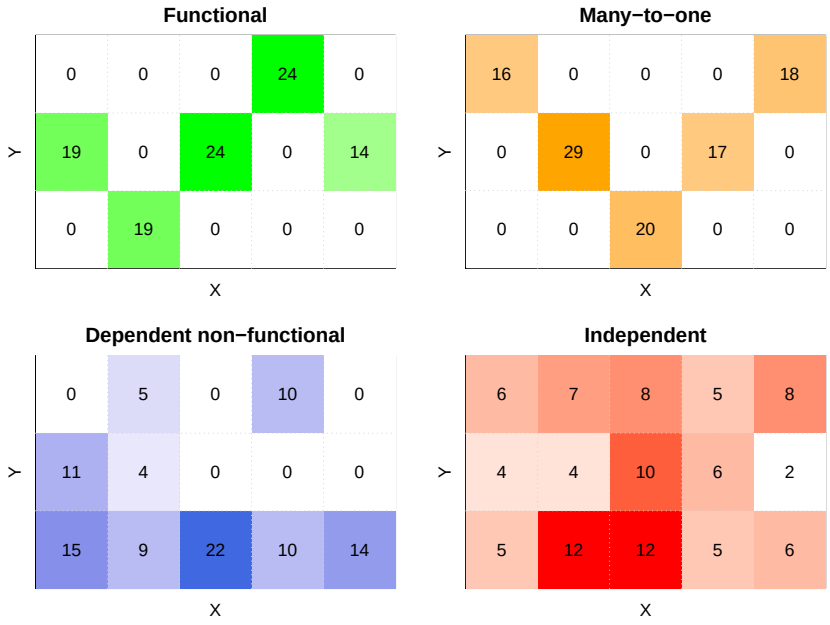


Figure 1: Four pattern types generated by the simulator. The contingency tables are rotated so that row variable X is the horizontal axis. The original tables are all 5×3 with a sample size of 100. The number in each square is the count in the corresponding table entry. No noise is applied. The color intensity of each entry is proportional to the sample count in that entry.

entry at row i and column j. We use the notation $n_{i\cdot}$ to indicate the sum of row i in a contingency table:

$$n_{i\cdot} = \sum_{j=1}^{c} n_{ij}, \quad i = 1, \ldots, r \tag{1}$$

and $n_{\cdot j}$ as the sum of column j:

$$n_{\cdot j} = \sum_{i=1}^{r} n_{ij}, \quad j = 1, \ldots, c \tag{2}$$

Generating functional tables

Functional patterns can be used to model causal relationships that are either linear or nonlinear. A contingency table reduces the burden of assuming a parametric form for the function. In a functional table representing $Y = f(X)$, every level $X = i$ maps to exactly one outcome $Y = j$ in the contingency table. The following steps generate a noise-free sampled table that satisfies all mathematical and statistical requirements:

> *Input*: row marginal probability function p_X, sample size $n \geq 2$, table size $r \times c$, $r, c \geq 2$.
>
> *Output*: a non-constant functional table.

1. Randomly generate the row sums $n_{i\cdot}$ $(i = 1, \ldots, r)$ by the multinomial distribution with success probability function p_X and the sample size n.
2. For each row i in the table, initialize the entire row to be 0, randomly pick a column $j \in \{1, \ldots, c\}$ with equal probability, and set $n_{ij} = n_{i\cdot}$.
3. Convert the function from constant to non-constant: If the function is constant—all nonzero values are on the same column j, randomly pick a row i' from $\{1, \ldots, r\}$ with equal probability and a column j' from $\{1, \ldots, c\} \backslash \{j\}$ with equal probability. Swap the values of n_{ij} and $n_{i'j'}$. This will guarantee function $Y = f(X)$ is a non-constant function.

Generating many-to-one functional tables

Many-to-one functions are special cases of functional relationships. They are increasingly relevant as they can expose complex patterns from data of large sample sizes. Being able to generate such patterns will facilitate the evaluation of complex pattern discovery methods. The following two main steps generate this type of patterns:

Input: row marginal probability function p_X, sample size $n \geq 2$, table size $r \times c$, $r \geq 3$, $c \geq 2$.

Output: a strictly many-to-one and non-constant functional table.

1. Generate a non-constant functional table using Step 1 to 3 above.
2. Convert the function from one-to-one to many-to-one: If the function is one-to-one—every column of the table has at most one non-zero entry, do the following. Randomly pick a row i from all those rows with non-zero totals with equal probability. Let j be the index to the only non-zero entry on row i. Pick j' from the indices of those columns with non-zero totals except j with equal probability. Swap the values of n_{ij} and $n_{ij'}$.

Generating dependent and non-functional tables

In non-functional dependent relationships between X and Y, X and Y are statistically dependent but Y is not a function of X and X is neither a function of Y. They are not as strong as functional patterns in revealing causal relationships, but can reveal strong associative relationships such as a circular pattern. Their joint probability function must satisfy

$$p_{XY}(i,j) \neq p_X(i) \cdot p_Y(j) \quad \text{for some } i,j \tag{3}$$

As we do not have the above true probability functions, we will directly estimate them using frequencies in the table. If the above inequality is satisfied, we call X and Y empirically statistically dependent; otherwise, X and Y are empirically statistically independent.

Lemma 1. Let p, P, Q, M, N be five positive numbers and q be a nonnegative number, where $P > p$ and $Q > q$. If these numbers satisfy two equalities

$$\frac{P}{N} \cdot \frac{M}{N} = \frac{p}{N} \tag{4}$$

and

$$\frac{Q}{N} \cdot \frac{M}{N} = \frac{q}{N} \tag{5}$$

and if we are given a number k such that $0 < k \leq p$, then the following two inequalities must be true:

$$\frac{P-k}{N} \cdot \frac{M}{N} > \frac{p-k}{N} \tag{6}$$

and

$$\frac{Q+k}{N} \cdot \frac{M}{N} < \frac{q+k}{N} \tag{7}$$

Proof. As we are given $P > p$, we have algebraically

$$\frac{P-k}{p-k} = \frac{P-p}{p-k} + 1 \tag{8}$$

monotonically increasing as k increases from 0 to p. Thus it follows

$$\frac{P-k}{p-k} > \frac{P}{p} = \frac{N}{M} \tag{9}$$

using the equality in Eq. (4). Multiplying both sides by $\frac{(p-k)M}{N^2}$, we immediately prove inequality (6). Similarly, with given $Q > q$ and as k increases from 0, we have a monotonic decreasing function of k

$$\frac{Q+k}{q+k} = \frac{Q-q}{q+k} + 1 \tag{10}$$

which implies

$$\frac{Q+k}{q+k} < \frac{Q}{q} = \frac{N}{M} \tag{11}$$

based on Eq. (5). Multiplying both sides by $\frac{(q+k)M}{N^2}$, we immediately prove inequality (7). Q.E.D.

Based on Lemma 1, we design Algorithm 1 to break the statistical independency. Let (i,j) represent the entry at row i and column j.

Algorithm 1. Convert an independent non-functional table to a dependent non-functional table.

Input: an independent non-functional table

Output: a dependent non-functional table

1. Randomly pick an entry (i, j) with at least one sample point ($n_{ij} > 0$).
2. Randomly select column $j' \neq j$ with equal probability from row i.
3. Move all n_{ij} samples from entry (i, j) to (i, j'), which finally have 0 and $n_{ij} + n_{ij'}$ sample points, respectively.

Theorem 1. An empirically statistically independent non-functional contingency table can be converted to dependent non-functional contingency table by Algorithm 1. The row marginal probability function remains the same after the conversion.

Proof. Table 2 illustrates that any independent non-functional table has at least two rows and two columns with non-zero entries. Non-functionality guarantees at least one row (row i_1 in Table 2) and one column (column j_1 in Table 2) each with two non-zero entries. The independent property ensures that at least another non-zero row (row i_2 in Table 2) and another non-zero column (column j_2 in Table 2) has the same distribution of numbers proportional to row i_1 and the column j_1, respectively. Additionally, all non-zero rows are proportional to each other by a constant, and so do the columns.

$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
$\cdots$	$n_{i_1 j_1} > 0$	$\cdots$	$n_{i_1 j_2} > 0$	$\cdots$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
$\cdots$	$n_{i_2 j_1} > 0$	$\cdots$	$n_{i_2 j_2} > 0$	$\cdots$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$

Table 2: Any independent non-functional table has at least two rows and two columns each with at least two non-zero entries.

In the input table to Algorithm 1, entry (i, j) satisfies the empirical independence equation

$$\frac{n_{i\cdot}}{n} \cdot \frac{n_{\cdot j}}{n} = \frac{n_{ij}}{n} \tag{12}$$

where $n_{i\cdot}$ is the sum of row i and $n_{\cdot j}$ is the sum of column j. After the move, we have by Lemma 1

$$\frac{n_{i\cdot}}{n} \cdot \frac{n_{\cdot j} - n_{ij}}{n} > \frac{0}{n} \tag{13}$$

where $n_{\cdot j} - n_{ij} > 0$ is guaranteed by the property shown in Table 2. As the left hand side of the inequality is the product of the new marginal probabilities and the right hand side is the new joint probability, it implies the loss of statistical independency in the modified table. Therefore the modified table is now statistically dependent.

Also by the property shown in Table 2, no matter where n_{ij} was moved to, we will indeed have another row and another column with at least two non-zero entries, which will guarantee the non-functionality of the modified table.

As the samples are moved within the same row i, the row sum is unchanged and thus the row marginal probability function is unchanged in the modified table.

Therefore, we conclude that the new table must be dependent non-functional. Q.E.D.

Such tables are only possible when $n \geq 4$. The following steps generate such tables by distributing samples to rows and then to columns and converting any resulting functions to non-functions:

Input: row marginal probability function p_X, sample size $n \geq 4$, table size $r \times c$, $r \geq 2$, and $c \geq 2$.

Output: a dependent non-functional table.

1. Randomly generate the row sums $n_{i\cdot}$ $(i = 1, \ldots, r)$ by the multinomial distribution with success probability function p_X and the sample size n.
2. For each row i in the table, initialize the entire row to be 0. Sample a value k from $\{1, \ldots, c\}$ with equal probability. Randomly pick k columns $j_1, \ldots, j_k$ from $\{1, \ldots, c\}$ with equal probability. Then randomly determine the values of $n_{ij_1}, \ldots, n_{ij_k}$ using the multinomial distribution with equal success probability of $1/k$.
3. If $Y = f(X)$, convert the function to a non-function: If the table represents a valid function—each row has at most one non-zero entry, do the following. Pick a row i from all rows with at least two samples with equal probability. Let j be the index to the only non-zero entry in row i. Pick j' from $\{1, \ldots, c\} \backslash \{j\}$ with equal probability. Redistribute the row sample $n_{i\cdot}$ into n_{ij} and $n_{ij'}$ with equal probability using a binomial distribution.

4. If $X = g(Y)$, convert the function to a non-function: follow the previous step but switch rows and columns. This operation also maintains the non-functionality $Y \neq f(X)$ as no non-zero entries are eliminated.
5. If empirically $X \perp Y$, we break the statistical independency by converting the table to a dependent non-functional table using Algorithm 1.

Generating independent tables

When X and Y are statistically independent, their joint probability mass function is

$$p_{XY}(i,j) = p_X(i) \cdot p_Y(j) \quad \text{for all } i \in \{1,\ldots,r\} \text{ and } j \in \{1,\ldots,c\} \tag{14}$$

where $p_X(i)$ and $p_Y(j)$ are marginal probabilities of occurrence for $X = i$ and $Y = j$, respectively. The tables formed by such X and Y are useful as a negative control, because they represent patterns that are typically considered uninteresting. To generate an independent pattern, the input needs both row and column marginal probability mass functions p_X and p_Y. As X and Y are statistically independent, the numbers of samples in each entry follow a multinomial distribution. The success probability in the entry at row i and column j is exactly $p_{XY}(i,j)$. Given the sample size n, we randomly generate samples from the multinomial distribution with $r \times c$ outcomes and map them to the sampled contingency table.

The house noise model

It is very common to observe noise in real data; noise may arise due to a multitude of reasons ranging from data preparation through the machinery involved in the entire data acquisition process. In statistical inference, noise needs to be handled to reduce type I and type II errors. Thus, our simulator also factors in noise to make the contingency tables resemble those constructed out of real data sets in order to additionally provide for a test of robustness.

By specifying the noise level parameter in the `simulate_tables` function, one can apply noise to a contingency table. We use the discrete house noise model (Zhang et al., 2015) that is controlled by the *noise* level parameter θ. θ is a continuous constant between 0 and 1, where 0 represents no noise and 1 represents maximum noise level. Let $Y \in \{1,\ldots,c\}$ be a dependent discrete random variable and $c > 1$ be the number of discrete levels of Y. Let $Y' \in \{1,\ldots,c\}$ be a random discrete variable which represents the noisy version of Y. Let y and y' represent values for Y and Y', respectively. The house noise model is defined through a probability function of noisy Y' conditioned on truth Y and noise level θ by

$$p_{Y'|Y}(y'|y,\theta) = \Pr(Y' = y'|Y = y,\theta) = \begin{cases} \left[\left(1 - \frac{|y-y'|}{\sum_{d=1}^{c}(|d-y|)}\right)\frac{\theta}{c-1}\right](1-\theta) + \frac{\theta}{c} & \text{if } y' \neq y \\ \left(\frac{\theta}{c-1} + 1 - \theta\right)(1-\theta) + \frac{\theta}{c} & \text{if } y' = y \end{cases} \tag{15}$$

The model works on the principle of making its value closer to the true value rather than being further away. By this model, the probability of a larger deviation of Y' from Y is smaller than that of a smaller deviation. At $\theta = 0$, we have $Y' = Y$ and it is thus noise free; at $\theta = 1$, the conditional probability function becomes uniformly distributed and the noise completely destroys the underlying patterns.

We implemented the model as an R function `add.house.noise` to apply noise to an input contingency table and return a noisy contingency table as the output. The syntax of the function is as follows:

```
add.house.noise(tables, u, margin = 1)
```

where `tables` is a list of input contingency tables and `u` is the noise level between 0 and 1. The noise can be applied along rows (`margin=1`), columns (`margin=2`), or both rows and columns (`margin=0`). In `simulate_tables`, the noise is always applied along the rows, which can be interpreted as applying noise to the dependent variable Y if the column variable is a function of the row variable X. The house noise model can be used independently of function `simulate_tables`.

Performance evaluation

To affirm each table type generated by the simulator indeed has the required characteristics, we compared two hypothesis testing methods: the `fun.chisq.test` (Zhang and Song, 2013) in the R package **FunChisq** and `chisq.test` in the R package **stats**.

We first simulated 1000 randomly generated tables for each of the four types at the noise level of 0.1. The numbers of rows and columns of the tables were randomly selected from 3 to 10. The sample size of each table was randomly drawn from 10 to 1000 and must be at least the table size.

Next we applied the two tests on the tables.The log p-value distributions of both tests are shown in Fig. 2. In Fig. 2a, the FunChisq test tends to yield smaller p-values compared to Pearson's chi-square on functional patterns. In Fig. 2b, both tests are approximately equal on dependent non-functional patterns. The p-values remarkably increased in comparison to functional patterns but are still showing significance due to statistical dependency in the pattern. Taken together, the FunChisq test has a higher sensitivity of functional over non-functional patterns than Pearson's chi-square test, as expected.

In Fig. 2c, the FunChisq test was performed first on noisy many-to-one functional patterns and second on the transposed table. There is an increase in the p-value by the latter which explains that the transposed noisy many-to-one tables are no longer functional. We did not show Pearson's chi-square p-values because they are equal on transposed contingency tables. Indeed, one expects to see such a difference in the distribution of FunChisq statistics when comparing functional tables versus their transposed non-functional tables, as FunChisq, unlike Pearson's chi squares, is asymmetric when testing functional dependency.

In Fig. 2d the log p-value distributions of both FunChisq and Pearson's chi-squares are nearly identical and the p-values are very close to 1, because both tests follow the same χ^2 distribution under the null hypothesis when using independent patterns.

These results confirm that our simulator was indeed able to generate functional, non-functional, many-to-one, and independent patterns.

We further benchmarked the runtime of the simulator over four table types, four sample sizes (100, 500, 1000, and 10000), and two noise levels (0 and 0.5) at a fixed table size of 5×5. This gives a total of 32 configurations. For each configuration, we repeated the simulation 100 times to account for random variations. We ran the simulation on iMac (27-inch, Mid 2010) with 2.93 GHz Intel Core i7 processor and 16 GB 1333 MHz DDR3 RAM. Figure 3 shows the runtime of generating noise-free and noisy versions of all four pattern types. It is apparent from the figure that table type, sample size, and noise application can all influence the runtime of table generation. Dependent non-functional tables took the most time because they are subject to more mathematical requirements. Applying noise incurred additional time for all table types. Sample size has an observable effect on the runtime when noise is applied.

Installation and examples of discrete pattern generation

Here we demonstrate the usage of the simulator by providing examples of all four pattern types including the description of important parameters. The R package **FunChisq** ($\geq$ 2.4.0) contains the function `simulate_tables` that implements the simulator. The package is publicly available from the Comprehensive R Archive Network (CRAN). The package can be installed and loaded by

```
install.packages("FunChisq")
library("FunChisq")
```

The signature of the function `simulate_tables` is given below:

```
simulate_tables(n=100, nrow=3, ncol=3, type=c("functional", "many.to.one",
    "independent", "dependent.non.functional"), noise=0.0, n.tables=1,
    row.marginal=rep(1/nrow, nrow), col.marginal=rep(1/ncol, ncol))
```

The arguments are

- `n`—an integer specifying the sample size to be distributed in the table. For `"functional"` and `"many.to.one"` tables, `n` must be no less than `nrow`. For `"independent"` tables, `n` must be no less than `nrow*ncol`. For `"dependent.non.functional"` tables, `n` must be greater than `nrow`.
- `nrow`—an integer specifying the number of rows of output tables. The value must be no less than 2. For `"many.to.one"` tables, `nrow` must be no less than 3.
- `ncol`—an integer value specifying the number of columns in desired table. The value for `ncol` must be no less than 2.
- `type`—a character string to specify the type of pattern underlying the table. The options are `"functional"` (default), `"many.to.one"`, `"independent"`, and `"dependent.non.functional"`.
- `noise`—a numeric value between 0 and 1 specifying the factor of noise to be added to the table using the house noise model (Zhang et al., 2015). The house noise is applied along the rows of the table. See `add.house.noise` for details.

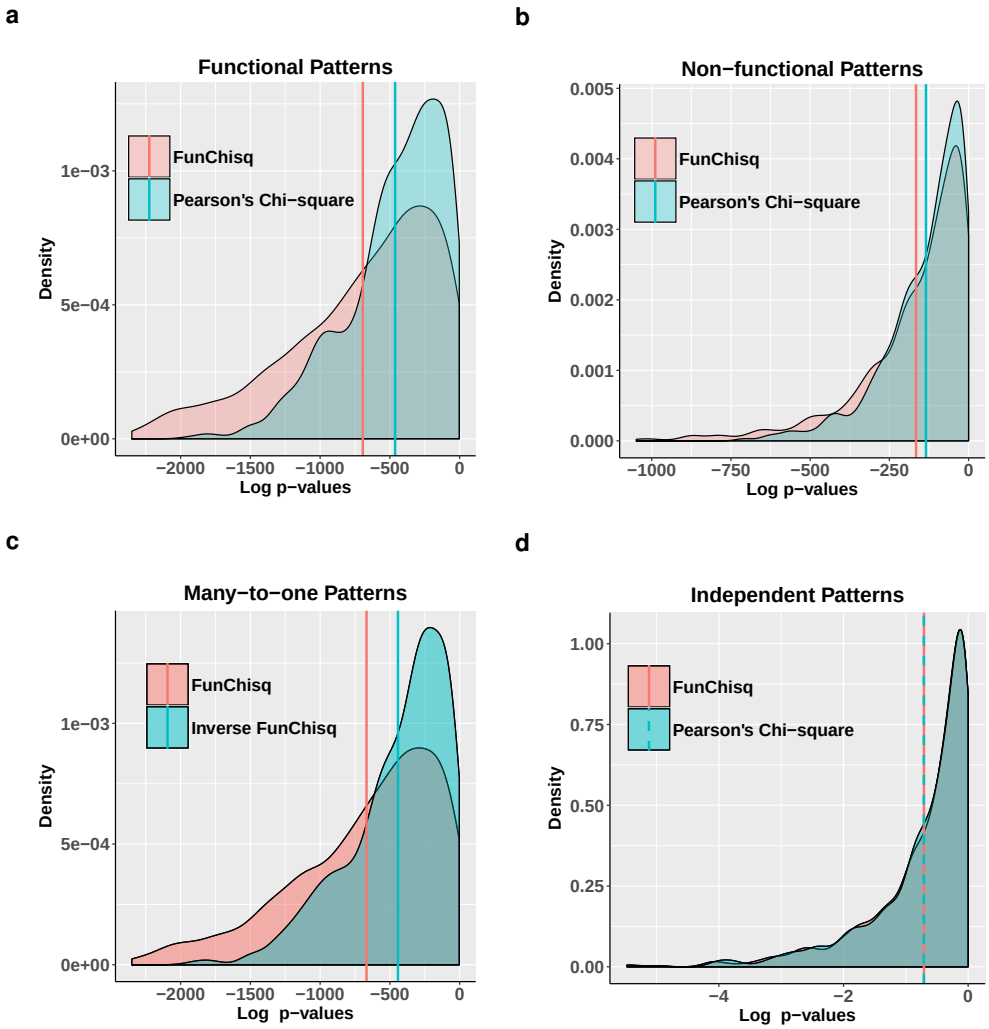


Figure 2: The FunChisq test is more sensitive to functional patterns than Pearson's chi-square test, while being equally sensitive to non-functional patterns. The distributions of log p-values for all four table types using the two tests at a noise level of 0.1 are shown. The means of the distributions are indicated by the vertical lines. (a) Noisy functional patterns. (b) Noisy dependent non-functional patterns. (c) Noisy many-to-one functional patterns. The FunChisq test was applied on both the original table and the transposed table, indicated by inverse FunChisq. (d) Noisy independent patterns.

- `n.tables`—an integer value specifying the number of tables to be generated.
- `row.marginal`—a numeric vector of length at least 2 specifying row marginal probabilities. For `"many.to.one"` tables, the length of `row.marginal` vector must be no less than 3.
- `col.marginal`—a numeric vector specifying column marginal probabilities. It is only applicable in generating independent tables.

The return value of the function `simulate_tables` is a list containing the following components:

- `pattern.list`—a list of tables containing 0-1 binary patterns. Each table is created by setting all non-zero entries in the corresponding sampled contingency table from `sample.list` to 1. Each table strictly satisfies the functional relationship for a given pattern `type` requested. This table does not meet the statistical requirements. As each table represents the truth regarding the mathematical relationship between the row and column variables, they can be used as the ground truth or gold standard for benchmarking.
- `sample.list`—a list of tables satisfying both the functional and statistical requirements. These tables are noise free.
- `noise.list`—a list of tables after applying noise to the corresponding tables in `sample.list`. Each table is the noisy version of the sampled contingency table. Due to the added noise, each table may no longer strictly satisfy the required functional or statistical relationships. These tables are the main output to be used for the evaluation of a discrete pattern discovery algorithm.

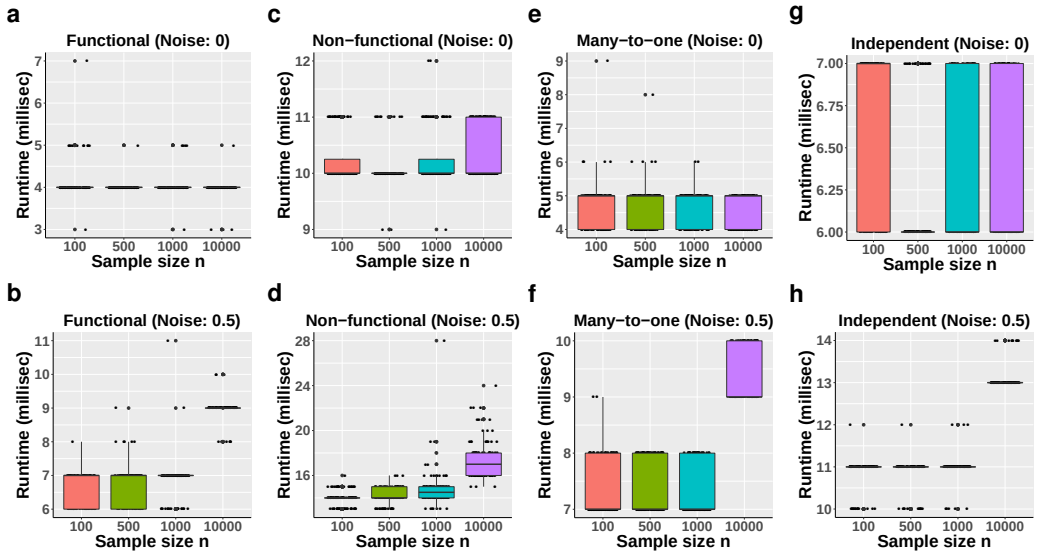


Figure 3: The empirical runtime of generating a contingency table as a function of sample size over four table types and two noise levels. We fix the table size to be 5×5. The runtime is measured in milliseconds and the box plots are colored by sample sizes. The runtime differs in sample size, table type, and whether noise was applied, as shown in (a) noise-free and (b) noisy functional patterns, (c) noise-free and (d) noisy dependent non-functional patterns, (e) noise-free and (f) noisy many-to-one functional patterns, and (g) noise-free and (h) noisy independent patterns.

- `pvalue.list`—a list of p-values reporting the statistical significance of the generated tables for the required type. When the pattern type specifies a functional relationship, the p-values are computed by the FunChisq test; otherwise, the Pearson's chi-square test of independence is used to calculate the p-value.

Example 1. A functional table. A scientist working at a weather forecasting agency came up with a novel method that can be trained for each geographical area to predict the amount of rainfall in that area given a month. While the agency is busy collecting real data for the last 10 years, the scientist wants to test his method and train the method to learn the following hypothesis: 'The amount of rainfall in an area is a function of time, expressed in months'. Using different probability distributions for each month (`row.marginal`) the scientist can simulate contingency tables with X being month with 12 levels and Y being the amount of rainfall with 3 levels 'Scant', 'Medium' and 'Heavy'. We assume that the amount of rainfall is equally likely in a given month. Noise can be added to imitate imperfect real-world scenario. The following code generates 12×3 functional contingency tables with 100 samples using multinomial distribution at the noise level of of 0.1:

```
simulate_tables(nrow=12, ncol=3, type='functional', n=100, n.tables=1,
                row.marginal = c(0.116, 0.109, 0.049, 0.142, 0.083,
                0.070, 0.140, 0.151, 0.037, 0.032, 0.050, 0.015), noise=0.1)
## Output:
##   pattern.table
      C1 C2 C3
## R1   0  1  0
## R2   0  0  1
## R3   0  0  1
## R4   0  1  0
## R5   0  1  0
## R6   0  0  1
## R7   1  0  0
## R8   0  1  0
## R9   0  0  1
## R10  1  0  0
## R11  0  0  1
## R12  0  0  1

sampled.table
    C1  C2  C3
R1   0   5   0
R2   0   0  17
R3   0   0   3
R4   0  15   0
R5   0   3   0
R6   0   0   5
R7  22   0   0
R8   0  15   0
R9   0   0   4
R10 5   0   0
R11 0   0   3
R12 0   0   3

noise.table
    C1 C2 C3
R1   0  5  0
R2   0  1 16
R3   0  0  3
R4   0 14  1
R5   1  2  0
R6   1  0  4
R7  20  1  1
R8   1 13  1
R9   0  1  3
R10 5  0  0
R11 0  0  3
R12 0  0  3
```

The scientist can also tune parameters and generate 100 or more such tables representing data for 100 or more years.

Example 2. A many-to-one functional table. This type of table is a special case of functional table. It can be used to model the relationship between the effectiveness Y and the dosage X of a drug. When the dosage of the drug is too low or too high, it becomes ineffective. The following code generates 4×5, noisy, many-to-one contingency tables with 100 samples distributed using multinomial distribution. The row marginals used are 0.1, 0.3, 0.2, 0.4. Noise factor of 0.5 is added to the sampled table.

```
   simulate_tables(nrow=4, ncol=5, type='many.to.one', n=100, n.tables=1, noise=0.5)
## Output:
##   pattern.table                    sampled.table                    noise.table
##    C1 C2 C3 C4 C5                    C1 C2 C3 C4 C5                    C1 C2 C3 C4 C5
## R1  0  0  0  1  0                 R1  0  0  0  19 0                 R1  2  4  0  11 2
## R2  0  1  0  0  0                 R2  0  32 0  0  0                 R2  5  12 7  3  5
## R3  0  1  0  0  0                 R3  0  31 0  0  0                 R3  5  10 4  7  5
## R4  0  0  1  0  0                 R4  0  0  18 0  0                 R4  2  3  7  4  2
```

Example 3. An independent table. This type of table is useful in generating null distribution or bootstrapping. To measure the statistical significance or probability of getting a certain empirical score reported by a method, one needs to randomly sample from an independent and identically distributed population. The following code generates 4×5 contingency tables with 100 samples distributed using multinomial distribution, where row and column variables are statistically independent. The row marginal probabilities are 0.1, 0.3, 0.2, 0.4 and column marginal probabilities are 0.3, 0.2, 0.1, 0.3, 0.1. No noise is applied.

```
 simulate_tables(nrow=4, ncol=5, type='independent', n=100, n.tables=1, noise=0.0,
                 row.marginal=c(0.1,0.3,0.2,0.4), col.marginal=c(0.3,0.2,0.1,0.3,0.1))
## Output:
##    pattern.table                    sampled.table                    noise.table
##    C1 C2 C3 C4 C5                    C1 C2 C3 C4 C5                    C1 C2 C3 C4 C5
## R1  1  1  1  1  1                 R1  4  3  2  4  1                 R1  4  3  2  4  1
## R2  1  1  1  1  1                 R2  8  4  1  5  3                 R2  8  4  1  5  3
## R2  1  1  1  1  1                 R3  6  3  1  6  5                 R3  6  3  1  6  5
## R2  1  1  1  1  1                 R4  16 8  5  7  8                 R4  16 8  5  7  8
```

Example 4. A dependent non-functional table. Two genes X and Y may be regulated by a common transcription factor. Here X and Y are categorized as low, medium and high. The two variables are dependent but not a function of each other. We simulate this type of data using dependent non-functional table type. The following code generates 3×3, noise free, non-functional contingency tables with 100 samples distributed using multinomial distribution. The row marginals used are 0.3, 0.5, 0.2.

```
   simulate_tables(nrow=3, ncol=3, type='dependent.non.functional', n=100, n.tables=1,
                   row.marginal=c(0.3,0.5,0.2), noise=0.1)
## Output:
##   pattern.table                  sampled.table                  noise.table
##    C1 C2 C3                        C1  C2  C3                     C1  C2  C3
## R1  1  1  0                     R1  12  11  0                   R1  8   13  2
## R2  0  0  1                     R2  0   0   49                  R2  3   1   45
## R3  1  1  1                     R3  7   10  11                  R3  9   9   10
```

Discussion

We have just introduced a new simulator which conveniently generates random noisy $r\times c$ contingency tables exhibiting four contrasting pattern types including functional, many-to-one functional, dependent non-functional, and independent. These contingency tables are advantageous for methods which are designed to discover patterns among discrete random variables.

Although other methods for constructing random contingency tables are available (Demirhan, 2016), the generation of functional tables is innovative. Having diverse functional tables meets a need to detect causal relationships from functional dependencies without using a parametric form.

For practical reasons, if the row marginal is non-zero, we will generate at least one sample for that row. Currently the simulator utilizes row marginal probabilities for the generation of dependent contingency tables; in the future we may provide an option to use the column marginal probabilities instead, to match some experimental design where the distribution of the effect variable is predefined, such as in case-control studies of cancer.

The generated dependent non-functional patterns may span a wide range of statistical dependency from being very weak to very strong. The p-values associated with these patterns can be used to filter out tables of weak statistical dependency.

Beyond generating tables of bivariate patterns between the row and column variables, one may consider the row variable as a combination of multiple discrete variables. Therefore one can immediately extend the procedures to generate contingency tables representing multivariate patterns.

In summary, we described algorithms and implemented a simulator to construct contingency tables of desired mathematical and statistical properties, and illustrated the use of this function with several examples. We validated the generation of all table types by the FunChisq and Pearson's chi-square tests. We evaluated the runtime of the function in generating various noisy patterns. This function offers a previously overlooked utility to generate diverse functional patterns to evaluate discrete pattern discovery methods increasingly important in data science research.

Acknowledgment

This work is supported by US NSF Advances in Biological Informatics Grant DBI 1661331.

Bibliography

G. A. Barnard. A new test for 2×2 tables. *Nature*, 156:177, 1945. URL https://doi.org/10.1038/156177a0. [p366]

R. G. Bartle. *The Elements of Real Analysis*, volume 2. John Wiley & Sons, 1964. [p366]

H. Demirhan. rTableICC: An R package for random generation of 2×2×K and R×C contingency tables. *The R Journal*, 8(1):48–63, 2016. [p366, 375]

R. A. Fisher. On the interpretation of χ^2 from contingency tables, and the calculation of P. *Journal of the Royal Statistical Society*, 85(1):87–94, 1922. URL https://doi.org/10.2307/2340521. [p366]

S. M. Hill, L. M. Heiser, T. Cokelaer, M. Unger, N. K. Nesser, D. E. Carlin, Y. Zhang, A. Sokolov, E. O. Paull, C. K. Wong, K. Graim, A. Bivol, H. Wang, F. Zhu, B. Afsari, L. V. Danilova, A. V. Favorov, W. S. Lee, D. Taylor, C. W. Hu, B. L. Long, D. P. Noren, A. J. Bisberg, The HPN-DREAM Consortium, G. B. Mills, J. W. Gray, M. Kellen, T. Norman, S. Friend, A. A. Qutub, E. J. Fertig, Y. Guan, M. Song, J. M. Stuart, P. T. Spellman, H. Koeppl, G. Stolovitzky, J. Saez-Rodriguez, and S. Mukherjee. Inferring causal molecular networks: Empirical assessment through a community-based effort. *Nature methods*, 13(4), 2016. URL https://doi.org/10.1038/nmeth.3773. [p366]

J. H. McDonald. *Handbook of Biological Statistics*, chapter G–test of goodness-of-fit, pages 53–58. Sparkly House Publishing, 3rd edition, 2014. [p366]

W. Patefield. Algorithm AS 159: An efficient method of generating random R×C tables with given row and column totals. *Journal of the Royal Statistical Society C*, 30(1):91–97, 1981. URL https://doi.org/10.2307/2346669. [p366]

K. Pearson. X. on the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 50 (302):157–175, 1900. URL https://doi.org/10.1080/14786440009463897. [p366]

H. A. Simon and N. Rescher. Cause and counterfactual. *Philosophy of Science*, 33(4):323–340, 1966. URL https://doi.org/10.1086/288105. [p366]

Y. Zhang and M. Song. Deciphering interactions in causal networks without parametric assumptions, 2013. URL https://arxiv.org/abs/1311.2707. [p366, 371]

Y. Zhang, Z. L. Liu, and M. Song. ChiNet uncovers rewired transcription subnetworks in tolerant yeast for advanced biofuels conversion. *Nucleic Acids Research*, 43(9):4393–4407, 2015. [p366, 371, 372]

Y. Zhang, H. Zhong, R. Sharma, S. Kumar, and J. Song. *FunChisq: Chi-Square and Exact Tests for Non-Parametric Functional Dependencies*, 2017. URL https://cran.r-project.org/package=FunChisq. R package version 2.4.0. [p366]

Ruby Sharma
Department of Computer Science
New Mexico State University
Las Cruces, NM
U.S.A.
ruby49@nmsu.edu

Sajal Kumar
Department of Computer Science
New Mexico State University
Las Cruces, NM
U.S.A.
sajal49@nmsu.edu

Hua Zhong
Department of Computer Science
New Mexico State University
Las Cruces, NM
U.S.A.
huazhong@nmsu.edu

Mingzhou Song
Department of Computer Science
New Mexico State University
Las Cruces, NM
U.S.A.
ORCiD: 0000-0002-6883-6547
joemsong@cs.nmsu.edu

glmmTMB Balances Speed and Flexibility Among Packages for Zero-inflated Generalized Linear Mixed Modeling

by Mollie E. Brooks, Kasper Kristensen, Koen J. van Benthem, Arni Magnusson, Casper W. Berg, Anders Nielsen, Hans J. Skaug, Martin Mächler, Benjamin M. Bolker

Abstract Count data can be analyzed using generalized linear mixed models when observations are correlated in ways that require random effects. However, count data are often *zero-inflated*, containing more zeros than would be expected from the typical error distributions. We present a new package, **glmmTMB**, and compare it to other R packages that fit zero-inflated mixed models. The **glmmTMB** package fits many types of GLMMs and extensions, including models with continuously distributed responses, but here we focus on count responses. **glmmTMB** is faster than **glmmADMB**, **MCMCglmm**, and **brms**, and more flexible than **INLA** and **mgcv** for zero-inflated modeling. One unique feature of **glmmTMB** (among packages that fit zero-inflated mixed models) is its ability to estimate the Conway-Maxwell-Poisson distribution parameterized by the mean. Overall, its most appealing features for new users may be the combination of speed, flexibility, and its interface's similarity to **lme4**.

Introduction

Observed response variables are often in the form of discrete count data, e.g., the number of times that owl nestlings beg for food (Roulin and Bersier, 2007), counts of salamanders in streams (Price et al., 2016), or counts of parasite eggs in fecal samples of sheep (Denwood et al., 2008). These counts are often analyzed using generalized linear models (GLMs) and their extensions (O'Hara and Kotze, 2010; Wilson and Grenfell, 1997). GLMs quantify how expected counts change as a function of predictor variables, e.g., nestlings change their behavior depending on which parent they interact with (Roulin and Bersier, 2007), salamander abundance decreases in streams affected by coal mining (Price et al., 2016), and helminth infection intensity in sheep decreases in response to treatment with anthelmintic drugs (Wang et al., 2017). Repeated measurements on the same individual, at the same location, or observations during the same time period are often correlated; this correlation can be accounted for using random effects in generalized linear mixed models (GLMMs; Bolker et al., 2009; Bolker, 2015).

These types of count data are commonly modeled with GLMs and GLMMs using either Poisson or negative binomial distributions. For the Poisson distribution, the variance is equal to the mean. When data are overdispersed — meaning the variance is larger than the mean — they are often instead modeled using the negative binomial distribution, which can be defined as a mixture of Poisson distributions with Gamma-distributed rates. Overdispersion can also be addressed by adding a random effect with one level for each observation, i.e., a log-normal Poisson distribution (Elston et al., 2001; Hadfield, 2010; Harrison, 2014, 2015). Ignoring overdispersion causes confidence intervals to be too narrow and inflates the rate of false positives in statistical tests (Rhodes, 2015). When data are either over- or underdispersed, they can be modeled with the lesser-known, Conway-Maxwell-Poisson distribution (Shmueli et al., 2005; Lynch et al., 2014; Barriga and Louzada, 2014). Depending on the dispersion, the upper tail of the Conway-Maxwell-Poisson distribution can be either longer or shorter than that of the Poisson (Sellers and Shmueli, 2010). With two parameters, the Conway-Maxwell-Poisson is a generalization of the Poisson distribution and, depending on the dispersion parameter, it also includes the Bernoulli and geometric distributions as special cases (Sellers and Shmueli, 2010). Among other generalizations, the Sichel and Delaporte distributions (Stasinopoulos et al., 2017) provide flexibility in skewness in addition to dispersion.

For these distributions, the expected number of zeros decreases as the mean increases. However, when multiple processes underlie the observed counts, the counts can contain many zeros even if the mean is much greater than zero. For example, an observation of a stream with zero salamanders could be a "structural" zero due to the stream being uninhabitable due to mining waste, or a "sampling" zero due to the combination of a low mean (due to poor ecological suitability and/or low detectability) and sampling variation (Price et al., 2016). Zero-inflated (more broadly zero-altered) GLMs allow us to model count data using a mixture of a Poisson, negative binomial, or Conway-Maxwell-Poisson distribution and a structural zero component. Models that ignore zero-inflation, or attempt to handle it in the same way as simple overdispersion, can yield biased parameter estimates (Harrison, 2014).

In this article, we outline the R packages available for fitting models to count data while introducing **glmmTMB**. We assume that the reader already has a basic understanding of GLMs (Buckley, 2015),

GLMMs (Bolker, 2015), and zero-altered models (Zeileis et al., 2008; Harrison, 2014)).

Several R packages are available for fitting zero-inflated models: **pscl**, **INLA**, **MCMCglmm**, **glmmADMB**, **mgcv**, **brms**, and **gamlss** (Table 1; Zeileis et al., 2008; Rue et al., 2009; Hadfield, 2010; Skaug et al., 2012; Wood et al., 2016; Bürkner, 2017; Stasinopoulos et al., 2017). The widely-used **pscl** package can fit zero-inflated and hurdle GLMs with predictor variables on the zero-inflation using maximum likelihood estimation (MLE: Zeileis et al., 2008). For example, **pscl** can be used to test the hypothesis that sheep fecal egg counts depend on age and structural zeros depend on genotype. However, **pscl** cannot model the correlation within sampling units caused by repeated samples; this requires random effects. Omitting random effects and thereby ignoring correlation makes statistical inference anti-conservative (Bolker et al., 2009; Bolker, 2015). Several other packages have similar capabilities for fitting zero-inflated GLMs (**flexmix**, **MXM**, **VGAM**: Grün and Leisch, 2008; Lagani et al., 2017; Yee, 2017), but in this paper we focus on packages that can also estimate random effects. One such package is **glmmADMB** which can fit zero-inflated GLMMs (Skaug et al., 2012). However, **glmmADMB** cannot fit models where the degree of zero-inflation varies across observational units; thus, it is only appropriate for models where all observational units have an equal probability of producing a structural zero. **INLA** has the same limitation as **glmmADMB** (Rue et al., 2009). The **mgcv** package can only fit zero-inflated GLMMs with predictors of zero-inflation when using a Poisson distribution (Wood et al., 2016). The **MCMCglmm** and **brms** packages can fit zero-inflated GLMMs with predictors of zero-inflation, but they are relatively slow (as we will show) because they rely on Markov chain Monte Carlo (MCMC) sampling (Bürkner, 2017; Hadfield, 2010). **gamlss** is a flexible package that fits generalized additive models with predictors on all parameters of a distribution; its scope includes several zero-inflated and zero-altered distributions (Stasinopoulos et al., 2017).

The list of features documented here is not exhaustive. It should be appreciated that **brms**, **gamlss** and **MCMCglmm** have additional features that go beyond the scope of zero-inflated GLMMs (Bürkner, 2017; Stasinopoulos et al., 2017; Hadfield, 2010). We focus on the process of fitting models, largely neglecting questions of statistical frameworks (frequentist vs. Bayesian) or post-fitting procedures such as inference and prediction. For example, having MCMC samples from a fitted model allows a wide range of inferential and predictive procedures.

Here we introduce a new R package, **glmmTMB**, that estimates GLMs, GLMMs and extensions of GLMMs including zero-inflated and hurdle GLMMs using ML. The ability to fit these types of models quickly and using a single package will make it easier to perform model selection. We focus on zero-inflated counts, but note that there are many other distributions available in **glmmTMB**, including continuous distributions. We demonstrate the package using two examples without going into details of the reasons why a user would want to fit these models. We use an example of salamander abundance to show how to fit and compare zero-inflated and hurdle GLMMs and then how to extract results from a model. We then use variations of the salamander data to compare how the timings of different packages scale with the number of observations and random effect levels. We use a classic example of owl nestling behavior to compare the timing and parameter estimates from **glmmTMB** with other R packages.

Implementation of glmmTMB

The design goal of **glmmTMB** is to extend the flexibility of GLMMs in R while maintaining a familiar interface. To maximize flexibility and speed, **glmmTMB**'s estimation is done using the **TMB** package (Kristensen et al., 2016), but users need not be familiar with **TMB**. We based **glmmTMB**'s interface (e.g., formula syntax) on the **lme4** package — one of the most widely used R packages for fitting GLMMs (Bates et al., 2015). Like **lme4**, **glmmTMB** uses MLE and the Laplace approximation to integrate over random effects; unlike **lme4**, **glmmTMB** does not have the alternative options of doing restricted maximum likelihood (REML) estimation nor using Gauss-Hermite quadrature to integrate over random effects (Bolker et al., 2009; Bolker, 2015). The Laplace approximation may perform poorly when there is little information available on each random effect level (Ogden, 2015). REML may be added to **glmmTMB** in the future. The underlying implementation using **TMB** is a fundamental difference compared to **lme4** and provides **glmmTMB** with a speed advantage when estimating non-Gaussian models (Figures 1 and 2) and gives it greater flexibility in the classes of distributions it can fit (Table 1).

The flexibility of **glmmTMB** enables users to fit and compare many varieties of models with assurance that the log-likelihood values are calculated in a consistent way. Comparing likelihoods of models fit by multiple packages must be done carefully because some packages drop constants from the log-likelihood calculations while others do not.

A **glmmTMB** model has four main components: a conditional model formula, a distribution for the conditional model, a dispersion model formula, and a zero-inflation model formula. Simple GLMs and GLMMs can be fit using the conditional model while leaving the zero-inflation and dispersion

Feature	glmmTMB	glmmADMB	MCMCglmm	brms	INLA	mgcv	gamlss
hurdle models[1]	✓		✓	✓	✓		✓
predictors of zero-inflation	✓		✓	✓			✓
predictors of dispersion	✓			✓			✓
zero-truncated distributions	✓	✓	✓	✓	✓		✓
`nbinom2` distribution	✓	✓		✓	✓	✓[2]	✓
`nbinom1` distribution	✓	✓					✓[2]
`compois` distribution	✓[3]						
Delaporte distribution							✓[2]
Sichel distribution							✓
geometric distribution			✓	✓			✓[2]
PIG distribution							✓
weights	✓[4]	✓		✓	✓	✓	✓
offsets	✓	✓	[5]	✓	✓	✓	
various RE structures	✓[6]		✓	✓	✓	✓	
RE cor across components [7]			✓	✓			
MLE	✓	✓				✓	✓
MCMC samples[8]	✓[9]	✓	✓	✓	✓[10]	✓	
multivariate responses [11]			✓	✓			
GAM [12]				✓		✓	✓

Table 1: Features of packages that are used for modeling zero-inflated count data. This table only contains packages that can at least fit zero-inflated Poisson GLMMs. **lme4** is omitted because it can only estimate zero-inflation when wrapped in an expectation maximization algorithm (Bolker et al., 2013). `nbinom2` and `nbinom1` are negative binomial distributions in which the variance increases quadratically and linearly (respectively) with the mean. `compois` is the Conway-Maxwell-Poisson distribution. PIG is the Poisson inverse Gaussian distribution. Notes: 1 We restrict this to mean a single function call, rather than two separate models. 2 Not available with zero-inflation. 3 Conway-Maxwell-Poisson distribution parameterized by the mean. 4 Weights are often used to reduce the influence of some observations over others, e.g., Gurevitch and Hedges, 1999; additionally, `glmmTMB`'s dispersion formula can be used to model heteroskedasticity. 5 Offsets can be implemented using priors. 6 See `vignette("covstruct")` for details. 7 Some packages allow for correlation across random effects from different components of the model (e.g., conditional and zero-inflation). 8 Here, we mean that MCMC samples can be obtained from an estimated model (i.e., joint samples from the full distribution, conditional on the data) whether it is estimated using MCMC sampling or MLE. In the case of MLE, flat priors are used for MCMC sampling and chains are initiated at the ML estimate. 9 See `vignette("mcmc")` for details. 10 This feature is available and widely used e.g., Chevin et al., 2015, but apparently unsupported by any formal evaluation. 11 We exclude the possibility that the correlation structure of random effects makes a model mathematically equivalent to a multivariate response. 12 Here, we mean generalized additive modeling with automatic smoothness selection. However, a spline can be included in any of these methods using the `bs` or `ns` functions from the **splines** package.

formulas at their default values. The mean of the conditional model is specified using a two-sided formula with the response variable on the left and predictors on the right, potentially including random effects and offsets. This formula uses the same syntax as **lme4**. For example, if salamander counts vary by species (`spp`) and vary randomly by site, then the formula for the dependence of mean count on species could be

```
count ~ spp + (1 | site)
```

The distribution around the mean of the conditional model is specified using the argument `family`. For the types of count data described in the introduction, the distribution will typically be either Poisson or negative binomial. The Conway-Maxwell-Poisson is a less commonly known distribution for count data that is flexible enough to fit both over- and underdispersed data. Following the standard of **glmmTMB** described above, the Conway-Maxwell-Poisson distribution (`family = compois`) is parameterized with the mean (Huang, 2017), which differs from the **COMPoissonReg package**(Sellers et al., 2017). The Poisson, Conway-Maxwell-Poisson, and negative binomial distributions use a log link by default, but other links can be specified as in `family = poisson(link = "identity")`. **glmmTMB** provides two parameterizations of the negative binomial which differ in the dependence of the variance (σ^2) on the mean (μ). For `family = nbinom1`, the variance increases linearly with the mean as $\sigma^2 = \mu(1+\alpha)$, with $\alpha > 0$; for `family = nbinom2`, the variance increases quadratically with the mean as $\sigma^2 = \mu(1+\mu/\theta)$, with $\theta > 0$ (Hardin and Hilbe, 2007). For the Conway-Maxwell-Poisson distribution, there is no closed form equation for the variance (Huang, 2017).

With the default dispersion model (`dispformula = ~ 1`), the dispersion parameter (e.g., α or θ for the negative binomial distribution) is identical for each observation. Alternatively, the dispersion parameter can vary with fixed effects; in this case, the dispersion model uses a log link. The dispersion model can be used to account for heteroskedasticity. For example, if the response is more variable (relative to the mean) as the year progresses, then a model with either negative binomial distribution might use the one-sided formula `dispformula = ~ DOY` where `DOY` is the day of the year. When the same variables are in the conditional and dispersion models, the mean-variance relationship can be manipulated, but this could potentially lead to non-convergence issues. A description of the dispersion

parameter for each distribution can be accessed by typing `?sigma.glmmTMB` in R.

The zero-inflation model describes the probability of observing an extra (i.e., structural) zero that is not generated by the conditional model. Zero-inflation creates an extra point mass of zeros in the response distribution; the overall distribution is a mixture of the conditional model and zero-inflation model (Lambert, 1992; Rhodes, 2015). The zero-inflation probability is bounded between zero and one by using a logit link. For example, if salamanders emerged seasonally at each site, such that a structural zero could occur either because a site was contaminated or because it was visited too early in the season, then the model could include the one-sided formula `ziformula = ~ DOY`. The probability of producing a structural zero can be modeled as equal for all observations with `ziformula = ~ 1`. In **glmmTMB**, it is possible to include random effects in the conditional and zero-inflation models, but not the dispersion model.

Installation of glmmTMB

The package is available from The Comprehensive R Archive Network (CRAN) via the command `install.packages("glmmTMB")`. The current version is 0.2.0. Development versions are available from GitHub and can be installed using **devtools** (Wickham and Chang, 2017). Current details for installing development versions should be accessed on the GitHub page `https://github.com/glmmTMB/glmmTMB`.

Examples and benchmarks

To illustrate how to use **glmmTMB** and to compare it to other packages, we applied it to two data sets that are distributed with **glmmTMB**. Additional code and graphs for these examples can be found in Appendicies A and B.

ABUNDANCE OF SALAMANDERS IN STREAMS

Salamander data We demonstrate how to use features of `glmmTMB` to do model selection and output model results using a data set of the abundance of salamanders (Figure 3). They were observed four times at 23 sites in streams, some of which were impacted by coal mining; multiple species and life stages of salamanders were observed. The data set contains covariates that may affect the habitat suitability of a site and the ability of researchers to capture salamanders that inhabit the site (Price et al., 2016, 2015). Price et al. analyzed the data using a Bayesian model with an ecological and a sampling component; abundance was estimated with a hurdle Poisson model and then observations were modeled as binomial samples from the abundance.

Model fitting and selection We fit GLMMs, zero-inflated GLMMs, and hurdle models to the salamander data with Poisson, Conway-Maxwell-Poisson, and negative binomial distributions on the conditional model. For simplicity, we neglected some possible covariates. As a null model, we assumed that counts varied by species (`spp`) and randomly by site (`site`). We fit models where the mean count additionally depended on mining status (`mined`). Our full zero-inflated GLMMs allowed both the conditional and zero-inflation models to differ between mined and unmined sites. Full zero-inflated and hurdle negative binomial GLMMs were fit using the following commands (respectively) :

```
zinb = glmmTMB(count~spp * mined + (1|site), zi=~spp * mined,
        data=Salamanders, family=nbinom2)
hnb = glmmTMB(count~spp * mined + (1|site), zi=~spp * mined,
        data=Salamanders, family=truncated_nbinom2)
```

As is generally the case for model formulas in R, the `*` indicates an interaction plus main effects. We used Akaike information criteria (AIC) to compare all models via the `AICtab` function from the `bbmle` package (Bolker and Team, 2017). For convenience, `glmmTMB` reports the log-likelihood of unconverged models as `NA` and version 1.0.19 of `bbmle` puts these models at the bottom of AIC tables. The code for fitting these models and doing model selection is presented in Appendix A.

Of the models we considered, the most parsimonious was a Conway-Maxwell-Poisson GLMM that allowed counts to vary with species, mining, and their interaction. It did not include zero-inflation, as is common in abundance models (Warton, 2005). Two zero-inflated negative binomial and one zero-inflated Conway-Maxwell-Poisson model did not converge. Failed convergence is typically caused by trying to estimate parameters for which the data do not contain information. Models that do not converge should not be considered in model comparison. General model convergence issues are discussed in `vignette("troubleshooting",package="glmmTMB")`.

Model summary The summary of simple GLMMs from **glmmTMB** is modeled on the familiar output format of **lme4**. To demonstrate the extra output from zero-inflation and dispersion models, we present the summary from a more complicated model.

```
glmmTMB(count ~ mined + (1|site), zi=~mined , disp=~DOY, Salamanders, family=nbinom2)
```

Following the arguments in the function call above, this model allows the conditional mean to depend on whether or not a site was mined and to vary randomly by site. It allows the number of structural (i.e., extra) zeros to depend on mining. Additionally, it allows the dispersion parameter to depend on the day of the year. This model can be represented by the following set of equations

$$\begin{aligned}
\mu &= E(\texttt{count}|u,\texttt{NSZ}) = \exp(\beta_0 + \beta_{\texttt{minedno}} + u), && (1)\\
u &\sim N\left(0, \sigma_u^2\right), && (2)\\
\sigma^2 &= \mathrm{Var}(\texttt{count}|u,\texttt{NSZ}) = \mu(1+\mu/\theta), && (3)\\
\mathrm{logit}(p) &= \beta_0^{(\texttt{zi})} + \beta_{\texttt{minedno}}^{(\texttt{zi})}, && (4)\\
\log(\theta) &= \beta_0^{(\texttt{disp})} + \beta_{\texttt{DOY}}^{(\texttt{disp})} \cdot \texttt{DOY}, && (5)
\end{aligned}$$

where u is a site specific random effect, NSZ is the event "non-structural zero", $p = 1 - \Pr(\texttt{NSZ})$ is the zero inflation probability, and β's are regression coefficients with subscript denoting covariate/level (with 0 denoting intercept).

```
summary(glmmTMB(count~mined+(1|site), zi=~mined , disp=~DOY, Salamanders, family=nbinom2))
```

```
#>  Family: nbinom2  ( log )
#> Formula:          count ~ mined + (1 | site)
#> Zero inflation:         ~mined
#> Dispersion:             ~DOY
#> Data: Salamanders
#>
#>      AIC      BIC   logLik deviance df.resid
#>     1735     1767     -861     1721      637
#>
#> Random effects:
#>
#> Conditional model:
#>  Groups Name        Variance Std.Dev.
#>  site   (Intercept) 0.134    0.366
#> Number of obs: 644, groups:  site, 23
#>
#> Conditional model:
#>             Estimate Std. Error z value Pr(>|z|)
#> (Intercept)   -0.540      0.376   -1.44     0.15
#> minedno        1.424      0.365    3.90 0.000098 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Zero-inflation model:
#>             Estimate Std. Error z value Pr(>|z|)
#> (Intercept)    0.256      0.487    0.53   0.5988
#> minedno       -2.244      0.745   -3.01   0.0026 **
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Dispersion model:
#>             Estimate Std. Error z value Pr(>|z|)
#> (Intercept)   0.0278     0.3177    0.09     0.93
#> DOY          -0.3947     0.1537   -2.57     0.01 *
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

This summary can be broken down into five sections. The top section is a general overview containing a description of the model specification (`Family`, `Formula`, `Zero inflation`, `Dispersion`,

Data) and resulting information criteria. The information criteria can only be compared to models fitted by packages that, like, **glmmTMB**, compute the full form of the log-likelihood without dropping constant terms. The second section describes the variability of the Random effects. In this model, we only had random effects in the conditional model (equation 1), but random effects from the zero-inflation model (equation 4) could also appear here. The estimated variance 0.134 is σ_u^2 in equation 2. The third section describes the coefficients of the Conditional model (β_0 and $\beta_{\texttt{minedno}}$ in equation 1) including Wald Z statistics and p-values. Apart from the intercept, the estimates are all contrasts as is standard in regression models. This model has a log link as stated in the top line of the summary. The fourth section describes the Zero-inflation model similarly to the Conditional model except that this model has a logit link. The zero-inflation model estimates the probability of an extra zero such that a positive contrast indicates a higher chance of absence (e.g. minedno <0 means fewer absences in sites unaffected by mining); this is the opposite of the conditional model where a positive contrast indicates a higher abundance (e.g., minedno >0 means higher abundances in sites unaffected by mining). The estimates in this section correspond to $\beta_0^{(\texttt{zi})}$ and $\beta_{\texttt{minedno}}^{(\texttt{zi})}$ in equation 4. The last section provides estimated coefficients from the Dispersion model (equation 5), which uses a log link to keep the dispersion parameter θ positive. In contrast, a model with the default (simple) dispersion model would report the single dispersion parameter on the natural (rather than log) scale. To interpret the dispersion parameters of any distribution, see ?sigma.glmmTMB.

The current version of the summary function does not display uncertainty estimates for the random effects nor for single dispersion parameters, but confidence intervals can be calculated using the confint function. See ?confint.glmmTMB for details. All confidence intervals produced by the current version of the confint function are Wald intervals, based on the standard errors calculated using the delta method for the parameter on the scale of the internal parameterization (which varies by family).

Additional model output using the predict and simulate functions from glmmTMB is demonstrated in appendix A (Figures 4, 5, 6, and 7).

Timing comparisons Because **glmmTMB** is the only package that can fit Conway-Maxwell-Poisson GLMMs, it was not possible to do benchmarking with the most parsimonious model. Therefore, for benchmarking, we used the second best model (Δ AIC=0.5) which substituted a negative binomial response for the Conway-Maxwell-Poisson response. We measured the time required to fit this model using multiple packages. We performed three sets of timing benchmarks: (1) on simulated data with the same structure as the original salamander data, (2) on the original data replicated to create more observations per random effect level and the same number of random effect levels, (3) on simulated data with increasing numbers of random effect levels and the same number of observations per random effect level. Benchmarks were run in parallel using parLapply on a high performance computing cluster with 12 cores. This performance should match running on a single core. We used default values for all packages except with **glmmADMB** which required an additional argument (extra.args="-ndi 1000000") to allocate additional memory. By default, **brms** runs four MCMC chains while **MCMCglmm** runs one, which greatly affects their estimation time. However, it would be simple to speed up fitting of **brms** models by running the chains in parallel. For Bayesian methods, the important aspect of timing is sampling efficiency (minimum effective samples per unit time, Bürkner, 2017), but this is not compatible with the MLE methods, so we limit our presentation of the timings of the Bayesian methods.

Benchmarking showed that fitting the negative binomial model to simulated data with the same structure as the original data was, on average, equally fast in glmmTMB and INLA, 14 times slower with glmmADMB, 29 times slower with **lme4**, and 190 times slower with brms. **mgcv** fit the model the fastest, taking 0.03 times as long as **glmmTMB**. **gamlss** took 0.24 times as long as **glmmTMB**. With increasing numbers of observations, the estimation times of all packages appeared to follow power law functions (Figure 1). For simulated data sets with increasing numbers of random effect levels, estimation time increased as a power-law function for all packages except **INLA** which had estimation times that accelerated (Figure 2). The speed of glmmTMB for models with more random effect levels is due to the sparseness handling by TMB. Benchmarking nuances such as memory usage and how timings scale with model complexity could be investigated in more detail in future studies.

BEGGING BEHAVIOR OF OWL NESTLINGS

To further compare R packages for fitting zero-inflated GLMMs, we analyzed counts of begging behavior by owl nestlings. The full analyses can be found in Appendix B. This example previously appeared in Zuur et al. (2009) and Bolker et al. (2013) and was originally published by Roulin and Bersier (2007). We compared the estimates of fixed effects and the amount of time required for fitting the same model in **INLA**, **MCMCglmm**, **glmmADMB**, **mgcv**, and **brms** (Rue et al., 2009; Hadfield, 2010; Skaug et al., 2012; Wood et al., 2016; Bürkner, 2017). For **brms** and **MCMCglmm**, we used the

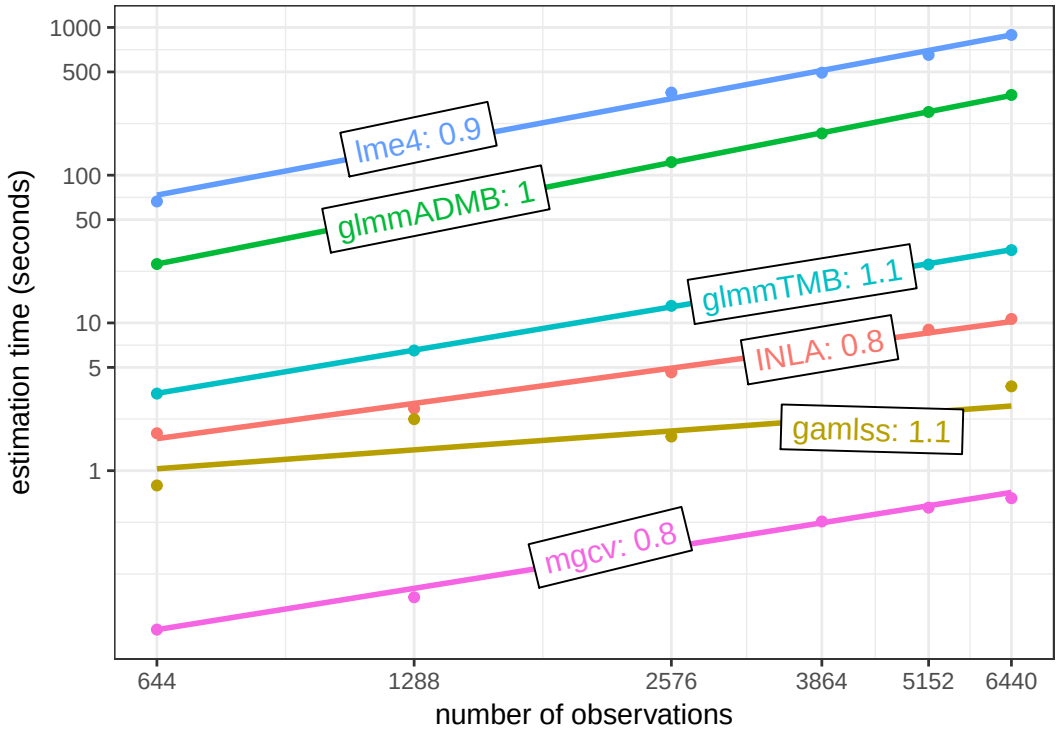


Figure 1: The Salamander data set was replicated by 1, 2, 4, 6, 8, and 10 times to create larger data sets. The time required to fit the same model using functions `glmmadmb`,`glmmTMB`,`inla`, `glmer.nb`, and `gam` was recorded. That model can be represented as `glmmTMB(count ~ spp * mined + (1 | site), Salamanders, family="nbinom2")`. All models had the same number of parameters including random effect levels. Lines represent linear models fit on the log-log scale. With increasing numbers of observations (n), estimation times increased as a power-law function (n^x) with exponents (x) reported next to model names.

default number of iterations, burn-in samples, and thinning. In each package, we fit zero-inflated Poisson models with six fixed effects, one random effect; we also accounted for overdispersion, although sometimes (of necessity) in slightly different ways with different packages (e.g., negative binomial vs. log-normal-Poisson models). We allowed zero-inflation to vary with food treatment and vary randomly with nest. See Appendix B for details of these methods, including code.

Estimates and confidence (or credible) intervals (CI) from **brms**, **mgcv**, `MCMCglmm`, and **INLA** were nearly identical to those of **glmmTMB**, when running the Bayesian models with flat priors (Figures 8 and 9).

Conclusions

We have introduced an R package that can quickly estimate a variety of models including GLMs, GLMMs, zero-inflated GLMMs, and hurdle models. By providing this flexibility in a single package, we reduce the need for researchers to learn multiple packages. Another benefit is that models estimated with a single package can be compared using likelihood-based methods including information criteria. Using information criteria to select a model for the salamander data, we found that (among the models we considered) zero-inflation did not improve the fit; we expect that this will be a common result. While **glmmTMB** allows users to easily fit complicated models, a maximally complex model might not be necessary and might not converge, as we saw here. Other packages have many of the features implemented in **glmmTMB**, but none have the ability to fit Conway-Maxwell-Poisson GLMMs. Overall, **glmmTMB** is a very flexible package for modeling count data with zero-inflated GLMMs while still ranking highly in speed comparisons.

Acknowledgements

Thanks to S Price for providing the data on salamanders and for helpful comments on the manuscript. Thanks to P-C Bürkner, J Hadfield, M Bekker-Nielsen Dunbar, and two anonymous reviewers for helpful comments on the manuscript. Thanks to A Zuur for providing the data on owl nestlings.

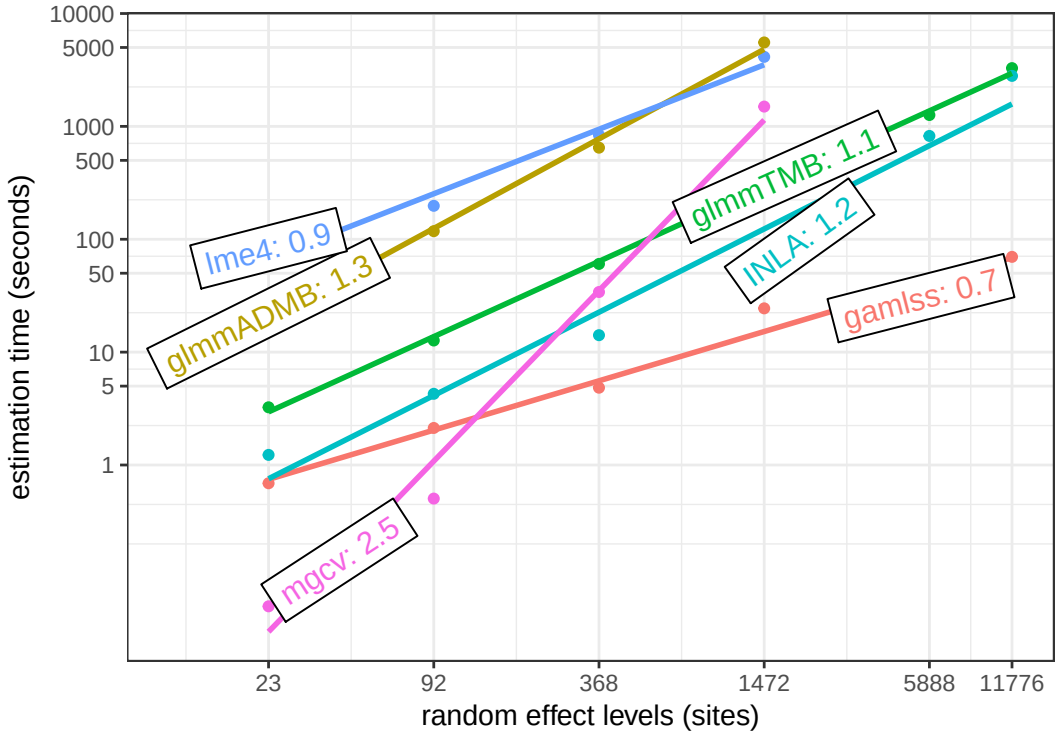


Figure 2: Data sets with increasing numbers of levels of the random effect were simulated based on a negative binomial model fit to the salamander data, `glmmTMB(count ~ spp * mined + (1 | site), Salamanders, family="nbinom2")`. The time required to fit the same model using functions `glmmadmb`, `glmmTMB`, `inla`, `glmer.nb`, and `gam` was recorded. Each simulated data set had the same number of observations per random effect level — the same ratio as in the original data. Lines represent linear models fit on the log-log scale.With increasing numbers of random effect levels (n), estimation times increased as a power-law function (n^x), except for INLA which accelerates. The exponents (x) are reported here as labels over the lines.

Thanks to G Simpson for advice on capabilities of the **mgcv** package. This work was supported by grants from the Swiss National Science Foundation (#IZ320Z0_161670) and from the AD Model Builder Foundation to MEB.

Bibliography

G. D. Barriga and F. Louzada. The Zero-Inflated Conway-Maxwell-Poisson Distribution: Bayesian Inference, Regression Modeling and Influence Diagnostic. *Statistical Methodology*, 21:23–34, 2014. URL https://doi.org/10.1016/j.stamet.2013.11.003. [p378]

D. Bates, M. Mächler, B. M. Bolker, and S. Walker. Fitting Linear Mixed-Effects Models Using Lme4. *Journal of Statistical Software*, 67(1):1–48, 2015. URL https://doi.org/10.18637/jss.v067.i01. [p379]

B. Bolker and R. D. C. Team. *bbmle: Tools for General Maximum Likelihood Estimation*, 2017. URL https://CRAN.R-project.org/package=bbmle. R package version 1.0.20. [p381]

B. M. Bolker. Linear and Generalized Linear Mixed Models. In G. A. Fox, S. Negrete-Yankelevich, and V. J. Sosa, editors, *Ecological Statistics*. Oxford University Press, Oxford, UK, 2015. [p378, 379]

B. M. Bolker, M. E. Brooks, C. J. Clark, S. W. Geange, J. R. Poulsen, M. H. H. Stevens, and J.-S. S. White. Generalized Linear Mixed Models: a Practical Guide for Ecology and Evolution. *Trends in Ecology & Evolution*, 24(3):127–135, 2009. URL https://doi.org/10.1016/j.tree.2008.10.008. [p378, 379]

B. M. Bolker, B. Gardner, M. Maunder, C. W. Berg, M. Brooks, L. Comita, E. Crone, S. Cubaynes, T. Davies, P. de Valpine, J. Ford, O. Gimenez, M. Kéry, E. J. Kim, C. Lennert-Cody, A. Magnusson, S. Martell, J. Nash, A. Nielsen, J. Regetz, H. Skaug, and E. Zipkin. Strategies for Fitting Nonlinear Ecological Models in R, AD Model Builder, and BUGS. *Methods in Ecology and Evolution*, 4(6): 501–512, 2013. URL https://doi.org/10.1111/2041-210x.12044. [p380, 383, 393, 395]

Y. Buckley. Generalized Linear Models. In G. A. Fox, S. Negrete-Yankelevich, and V. J. Sosa, editors, *Ecological Statistics*. Oxford University Press, Oxford, UK, 2015. [p378]

P.-C. Bürkner. brms: An R package for Bayesian multilevel models using Stan. *Journal of Statistical Software*, 80(1):1–28, 2017. URL https://doi.org/10.18637/jss.v080.i01. [p379, 383]

L. Chevin, M. E. Visser, and J. Tufto. Estimating the variation, autocorrelation, and environmental sensitivity of phenotypic selection. *Evolution*, 69(9):2319–2332, 2015. URL https://doi.org/10.1111/evo.12741. [p380]

M. J. Denwood, M. J. Stear, L. Matthews, S. W. J. Reid, N. Toft, and G. T. Innocent. The Distribution of the Pathogenic Nematode *Nematodirus Battus* in Lambs is Zero-Inflated. *Parasitology*, 135(10): 1225–1235, 2008. URL https://doi.org/10.1017/s0031182008004708. [p378]

D. A. Elston, R. Moss, T. Boulinier, C. Arrowsmith, and X. Lambin. Analysis of aggregation, a worked example: Numbers of ticks on red grouse chicks. *Parasitology*, 122(5):563–569, 2001. URL https://doi.org/10.1017/s0031182001007740. [p378, 394]

B. Grün and F. Leisch. FlexMix Version 2: Finite Mixtures with Concomitant Variables and Varying and Constant Parameters. *Journal of Statistical Software*, 28(4):1–35, 2008. URL https://doi.org/10.18637/jss.v028.i04. [p379]

J. Gurevitch and L. V. Hedges. Statistical Issues in Ecological Meta-Analyses. *Ecology*, 80(4):1142–1149, 1999. ISSN 00129658, 19399170. URL https://doi.org/10.2307/177061. [p380]

J. D. Hadfield. MCMC Methods for Multi-Response Generalized Linear Mixed Models: The MCMCglmm R Package. *Journal of Statistical Software*, 33(2):1–22, 2010. URL https://doi.org/10.18637/jss.v033.i02. [p378, 379, 383, 394]

J. W. Hardin and J. M. Hilbe. Generalized Linear Models and Extensions, 2007. [p380]

X. A. Harrison. Using observation-level random effects to model overdispersion in count data in ecology and evolution. *PeerJ*, 2:e616, 2014. ISSN 2167-8359. URL https://doi.org/10.7717/peerj.616. [p378, 379]

X. A. Harrison. A comparison of observation-level random effect and Beta-Binomial models for modelling overdispersion in Binomial data in ecology & evolution. *PeerJ*, 3:e1114, 2015. ISSN 2167-8359. URL https://doi.org/10.7717/peerj.1114. [p378]

A. Huang. Mean-Parametrized Conway–Maxwell–Poisson Regression Models for Dispersed Counts. *Statistical Modelling*, 17(6):1–22, 2017. URL https://doi.org/10.1177/1471082x17697749. [p380]

K. Kristensen, A. Nielsen, C. W. Berg, H. Skaug, and B. Bell. TMB: Automatic Differentiation and Laplace Approximation. *Journal of Statistical Software*, 70(1):1–21, 2016. URL https://doi.org/10.18637/jss.v070.i05. [p379]

V. Lagani, G. Athineou, A. Farcomeni, M. Tsagris, and I. Tsamardinos. Feature selection with the r package mxm: Discovering statistically equivalent feature subsets. *Journal of Statistical Software, Articles*, 80(7):1–25, 2017. URL https://doi.org/10.18637/jss.v080.i07. [p379]

D. Lambert. Zero-Inflated Poisson Regression, with an Application to Defects in Manufacturing. *Technometrics*, 34(1):1–14, 1992. URL https://doi.org/10.2307/1269547. [p381]

H. J. Lynch, J. T. Thorson, and A. O. Shelton. Dealing with under- and over-dispersed count data in life history, spatial, and community ecology. *Ecology*, 95(11):3173–3180, 2014. URL https://doi.org/10.1890/13-1912.1. [p378]

H. E. Ogden. A sequential reduction method for inference in generalized linear mixed models. *Electronic Journal of Statistics*, 9(1):135–152, 2015. URL https://doi.org/10.1214/15-ejs991. [p379]

R. B. O'Hara and D. J. Kotze. Do Not Log-Transform Count Data. *Methods in Ecology and Evolution*, 1 (2):118–122, 2010. URL https://doi.org/10.1111/j.2041-210x.2010.00021.x. [p378]

S. J. Price, B. L. Muncy, S. J. Bonner, A. N. Drayer, and C. D. Barton. Data From: Effects of Mountaintop Removal Mining and Valley Filling on the Occupancy and Abundance of Stream Salamanders, 2015. URL https://doi.org/10.5061/dryad.5m8f6. [p381, 387]

S. J. Price, B. L. Muncy, S. J. Bonner, A. N. Drayer, and C. D. Barton. Effects of Mountaintop Removal Mining and Valley Filling on the Occupancy and Abundance of Stream Salamanders. *Journal of Applied Ecology*, 53(2):459–468, 2016. URL https://doi.org/10.1111/1365-2664.12585. [p378, 381, 387]

J. R. Rhodes. Mixture Models for Overdispersed Data. In G. A. Fox, S. Negrete-Yankelevich, and V. J. Sosa, editors, *Ecological Statistics*. Oxford University Press, Oxford, UK, 2015. [p378, 381]

A. Roulin and L.-F. Bersier. Nestling Barn Owls Beg More Intensely in the Presence of Their Mother than in the Presence of Their Father. *Animal Behaviour*, 74(4):1099–1106, 2007. URL https://doi.org/10.1016/j.anbehav.2007.01.027. [p378, 383, 393]

H. Rue, S. Martino, and N. Chopin. Approximate Bayesian Inference for Latent Gaussian Models by Using Integrated Nested Laplace Approximations. *Journal of the royal statistical society: Series B (statistical methodology)*, 71(2):319–392, 2009. URL https://doi.org/10.1111/j.1467-9868.2008.00700.x. [p379, 383]

K. Sellers, T. Lotze, and A. Raim. *COMPoissonReg: Conway-Maxwell Poisson (COM-Poisson) Regression*, 2017. URL https://CRAN.R-project.org/package=COMPoissonReg. R package version 0.4.1. [p380]

K. F. Sellers and G. Shmueli. A Flexible Regression Model for Count Data. *The Annals of Applied Statistics*, 4(2):943–961, 2010. URL https://doi.org/10.1214/09-aoas306. [p378]

G. Shmueli, T. P. Minka, J. B. Kadane, S. Borle, and P. Boatwright. A Useful Distribution for Fitting Discrete Data: Revival of the Conway-Maxwell-Poisson Distribution. *Journal of the Royal Statistical Society C*, 54(1):127–142, 2005. URL https://doi.org/10.1111/j.1467-9876.2005.00474.x. [p378]

H. Skaug, D. Fournier, A. Nielsen, A. Magnusson, and B. M. Bolker. *glmmADMB: Generalized Linear Mixed Models Using AD Model Builder*, 2012. [p379, 383]

M. D. Stasinopoulos, R. A. Rigby, G. Z. Heller, V. Voudouris, and F. De Bastiani. *Flexible Regression and Smoothing: Using GAMLSS in R*. CRC Press, 2017. [p378, 379]

C. Wang, P. R. Torgerson, J. Höglund, and R. Furrer. Zero-inflated hierarchical models for faecal egg counts to assess anthelmintic efficacy. *Veterinary Parasitology*, 235:20 – 28, 2017. ISSN 0304-4017. URL https://doi.org/10.1016/j.vetpar.2016.12.007. [p378]

D. I. Warton. Many Zeros Does Not Mean Zero Inflation: Comparing the Goodness-of-Fit of Parametric Models to Multivariate Abundance Data. *Environmetrics*, 16(3):275–289, 2005. URL https://doi.org/10.1002/env.702. [p381]

H. Wickham and W. Chang. *devtools: Tools to Make Developing R Packages Easier*, 2017. URL https://CRAN.R-project.org/package=devtools. R package version 1.13.4. [p381]

K. Wilson and B. T. Grenfell. Generalized Linear Modelling for Parasitologists. *Parasitology Today*, 13 (1):33–38, 1997. [p378]

S. N. Wood, N. Pya, and B. Säfken. Smoothing Parameter and Model Selection for General Smooth Models. *Journal of the American Statistical Association*, 111(516):1548–1563, 2016. URL https://doi.org/10.1080/01621459.2016.1180986. [p379, 383]

T. W. Yee. *VGAM: Vector Generalized Linear and Additive Models*, 2017. URL https://CRAN.R-project.org/package=VGAM. R package version 1.0-4. [p379]

A. Zeileis, C. Kleiber, and S. Jackman. Regression Models for Count Data in R. *Journal of Statistical Software*, 27(8):1–25, 2008. URL https://doi.org/10.18637/jss.v027.i08. [p379]

A. F. Zuur, E. N. Ieno, N. J. Walker, A. A. Saveliev, and G. M. Smith. *Mixed Effects Models and Extensions in Ecology with R*. Springer-Verlag, 2009. [p383, 393]

Appendix A: Salamander example comparing GLMMs, zero-inflated GLMMs, and hurdle models using glmmTMB

In this appendix, we reanalyze counts of salamanders in streams. Repeated samples of salamanders were taken at 23 sites. Some of the sites were affected by mountain top removal coal mining. The data was originally published in (Price et al., 2016) and was acquired from Dryad (Price et al., 2015). These analyses are intended to be a simple demonstration of how to use some features of the **glmmTMB** package, so we do not attempt to fit all of the models that could be reasonable to try with the covariates that were collected.

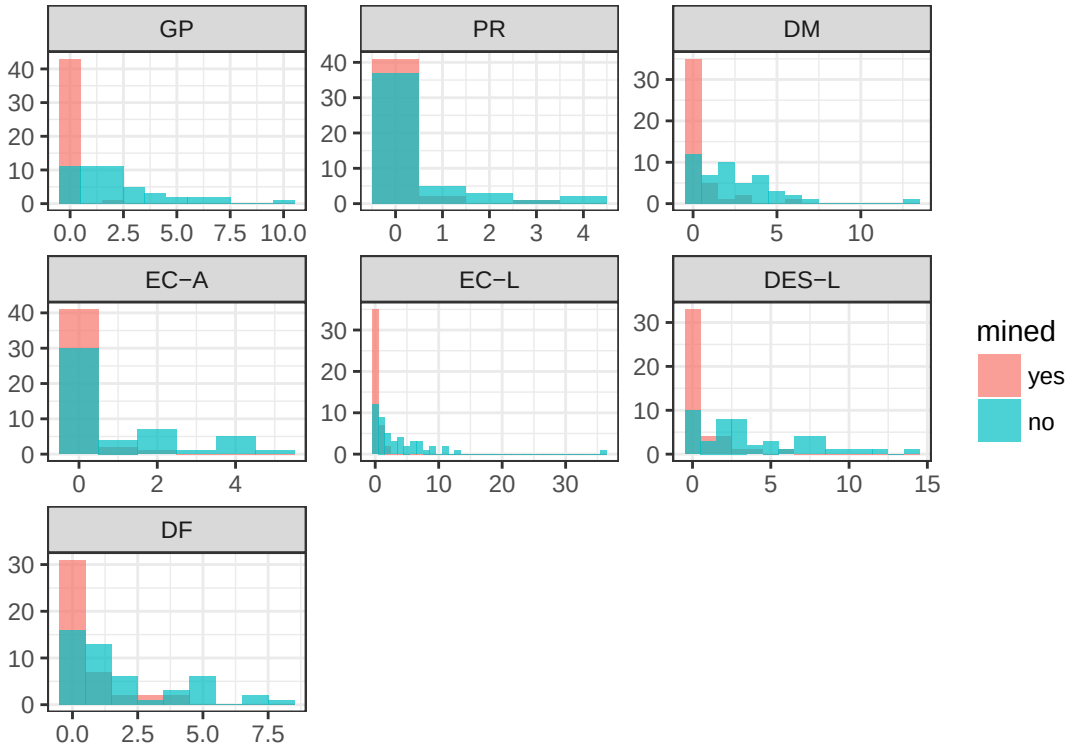


Figure 3: Observed numbers of salamanders. Histograms show count data split into separate panels for each salamander species or life stage. Each panel contains two overlaid histograms in which color represents whether the site was affected by mining.

Poisson models

The syntax for fitting GLMMs with **glmmTMB** is quite similar to using **glmer**. In the first model, the formula, `count ~ spp + (1 | site)`, says that counts depend on species and vary randomly by site. We also pass it the data frame, `Salamanders`, and specify a Poisson distribution using the `family` argument. **glmmTMB** assumes that we want a log link with the Poisson distribution because that's the standard.

```
pm0 = glmmTMB(count~spp + (1|site), Salamanders, family=poisson)
pm1 = glmmTMB(count~spp + mined + (1|site), Salamanders, family=poisson)
pm2 = glmmTMB(count~spp * mined + (1|site), Salamanders, family=poisson)
```

Conway-Maxwell-Poisson models

To fit Conway-Maxwell-Poisson models, we use `family=compois` instead of `poisson`.

```
cmpm0 = glmmTMB(count~spp + (1|site), Salamanders, family=compois)
cmpm1 = glmmTMB(count~spp + mined + (1|site), Salamanders, family=compois)
cmpm2 = glmmTMB(count~spp * mined + (1|site), Salamanders, family=compois)
```

Negative binomial models

```
nbm0 = glmmTMB(count~spp + (1|site), Salamanders, family=nbinom2)
nbm1 = glmmTMB(count~spp + mined + (1|site), Salamanders, family=nbinom2)
nbm2 = glmmTMB(count~spp * mined + (1|site), Salamanders, family=nbinom2)
```

Unlike the Poisson, the negative binomial distribution has a dispersion parameter. If we expected the counts to become more dispersed (relative to the mean) as the year progresses, then we could use the dispersion formula to model how the dispersion changes with the day of the year (`DOY`) using `disp= ~ DOY`.

```
nbdm0 = glmmTMB(count~spp + (1|site), disp=~DOY, Salamanders, family=nbinom2)
nbdm1 = glmmTMB(count~spp + mined + (1|site), disp=~DOY, Salamanders, family=nbinom2)
nbdm2 = glmmTMB(count~spp * mined + (1|site), disp=~DOY, Salamanders, family=nbinom2)
```

Zero-inflated models

To fit zero-inflated models, we use the `ziformula` argument, or **glmmTMB** will also recognize `zi`. This is a formula that describes how the probability of an extra zero (i.e., structural zero) will vary with predictors. In this example, we might assume that absences will at least vary by species (`spp`), so we write `zi= ~ spp`. This formula only has a right side because the left side is always the probability of having a structural zero in the response that was specified in the first formula. The zero-inflation probability is always modeled with a logit link to keep it between 0 and 1.

Zero-inflation can be used with any of the distributions in **glmmTMB**, so we compare the same conditional and zero-inflation models with Poisson, Conway-Maxwell-Poisson, and negative binomial distributions.

Warning messages tell us that `zicmpm3`, `zinbm0`, and `zinbm1` do not converge. The convergence warning refers to `vignette("troubleshooting")`; this vignette will expand as advice for troubleshooting convergence issues develops. It seems that `zicmpm3` is overparameterized, but the problem with `zinbm0` and `zinbm1` is that they only have `mined` in the conditional model and not the zero-inflation model, so that they do not agree well with the data. Plotting the data and thinking carefully about the models should help to avoid convergence issues.

```
zipm0 = glmmTMB(count~spp +(1|site), zi=~spp, Salamanders, family=poisson)
zipm1 = glmmTMB(count~spp + mined +(1|site), zi=~spp, Salamanders, family=poisson)
zipm2 = glmmTMB(count~spp + mined +(1|site), zi=~spp + mined, Salamanders, family=poisson)
zipm3 = glmmTMB(count~spp * mined +(1|site), zi=~spp * mined, Salamanders, family=poisson)

zicmpm0 = glmmTMB(count~spp +(1|site), zi=~spp, Salamanders, family=compois)
zicmpm1 = glmmTMB(count~spp + mined +(1|site), zi=~spp, Salamanders, family=compois)
zicmpm2 = glmmTMB(count~spp + mined +(1|site), zi=~spp + mined, Salamanders, family=compois)
zicmpm3 = glmmTMB(count~spp * mined +(1|site), zi=~spp * mined, Salamanders, family=compois)

#> Warning in fitTMB(TMBStruc): Model convergence problem; non-positive-
#> definite Hessian matrix. See vignette('troubleshooting')

zinbm0 = glmmTMB(count~spp +(1|site), zi=~spp, Salamanders, family=nbinom2)

#> Warning in fitTMB(TMBStruc): Model convergence problem; non-positive-
#> definite Hessian matrix. See vignette('troubleshooting')

zinbm1 = glmmTMB(count~spp + mined +(1|site), zi=~spp, Salamanders, family=nbinom2)

#> Warning in fitTMB(TMBStruc): Model convergence problem; non-positive-
#> definite Hessian matrix. See vignette('troubleshooting')

zinbm2 = glmmTMB(count~spp + mined +(1|site), zi=~spp + mined, Salamanders, family=nbinom2)
zinbm3 = glmmTMB(count~spp * mined +(1|site), zi=~spp * mined, Salamanders, family=nbinom2)

#> Warning in fitTMB(TMBStruc): Model convergence problem; singular
#> convergence (7). See vignette('troubleshooting')
```

Hurdle models

We can also fit hurdle models in a single model by using a truncated distribution for the conditional model and adding zero-inflation.

```
hpm0 = glmmTMB(count~spp + (1|site), zi=~spp, Salamanders, family=truncated_poisson)
hpm1 = glmmTMB(count~spp + mined + (1|site), zi=~spp + mined, Salamanders,
               family=truncated_poisson)
hpm2 = glmmTMB(count~spp * mined + (1|site), zi=~spp + mined, Salamanders,
               family=truncated_poisson)
hnbm0 = glmmTMB(count~spp + (1|site), zi=~spp, Salamanders, family=truncated_nbinom2)
hnbm1 = glmmTMB(count~spp + mined + (1|site), zi=~spp + mined, Salamanders,
               family=truncated_nbinom2)
hnbm2 = glmmTMB(count~spp * mined + (1|site), zi=~spp + mined, Salamanders,
               family=truncated_nbinom2)
```

Model comparison using AIC

We can use `AICtab` to compare all the GLMMs, including zero-inflated and hurdle models. Here, to save space, we only output the AICtable for the top four and bottom four models. The most parsimonious model has a Conway-Maxwell-Poisson distribution with effects of species, mining, and their interaction.

```
AICtab(pm0, pm1, pm2,
       cmpm0, cmpm1, cmpm2,
       nbm0, nbm1, nbm2,
       nbdm0, nbdm1, nbdm2,
       zipm0, zipm1, zipm2, zipm3,
       zicmpm0, zicmpm1, zicmpm2, zicmpm3,
       zinbm0, zinbm1, zinbm2, zinbm3,
       hpm0, hpm1, hpm2,
       hnbm0, hnbm1, hnbm2)
```

The top of the table

model	df	dAIC
cmpm2	16	0.00
nbm2	16	0.48
nbdm2	17	1.99
zicmpm2	18	2.09
zinbm3	30	6.80

and the bottom

model	df	dAIC
hpm0	15	314
pm0	8	330
zicmpm3	30	NA
zinbm0	16	NA
zinbm1	17	NA

The log-likelihood of the unconverged models is reported as `NA` so that these models appear at the end of the AIC table. The negative log-likelihood could be extracted with `zinbm1$fit$objective` if it was needed.

Plotting model results

There are many decisions to make about marginalizing over or conditioning on the random effects. See discussion at https://cran.r-project.org/web/packages/merTools/vignettes/Using_predictInterval.html.

For demonstration purposes, we plot results from the top zero-inflated model `zinbm3`.

Quick and dirty plot

It's easiest to see the pattern by using the `predict` function. To avoid marginalizing over or conditioning on random effects, we can refit the best model without the random effect of site; however, this is not ideal because it ignores the correlation within sites. We present a more rigorous version next.

The `predict` function has a parameter `zitype` that specifies whether you want predictions from the conditional model, the zero-inflation model, or the expected response that combines both parts of the model.

```
zinbm3FE = glmmTMB(count~spp * mined, zi=~spp * mined, Salamanders, family=nbinom2)
newdata0 = newdata = unique(Salamanders[,c("mined","spp")])
temp = predict(zinbm3FE, newdata, se.fit=TRUE, zitype="response")
newdata$predFE = temp$fit
newdata$predFE.min = temp$fit-1.96*temp$se.fit
newdata$predFE.max = temp$fit+1.96*temp$se.fit

real=ddply(Salamanders, ~site+spp+mined, summarize, m=mean(count))

ggplot(newdata, aes(spp, predFE, colour=mined))+geom_point()+
```

```
geom_errorbar(aes(ymin=predFE.min, ymax=predFE.max))+
geom_point(data=real, aes(x=spp, y=m) )+
ylab("Average abundance \n including presences and absences")+
xlab("Species")
```

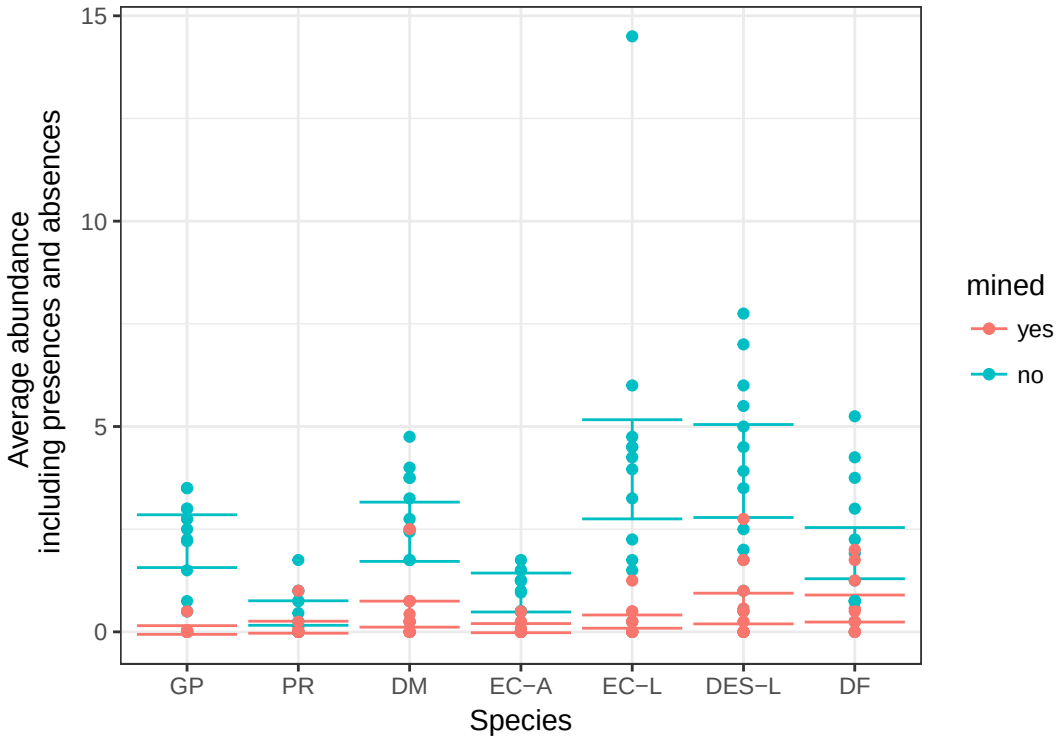


Figure 4: Estimated abundance ignoring correlation. Points represent site-specific average counts. Error bars represent the 95% Wald-type confidence intervals for the predicted average count.

Alternative prediction method

We can predict at the population mode, by setting the random effects to zero.

```
X.cond = model.matrix(lme4::nobars(formula(zinbm3)[-2]), newdata0)
beta.cond = fixef(zinbm3)$cond
pred.cond = X.cond %*% beta.cond

ziformula = zinbm3$modelInfo$allForm$ziformula #$
X.zi = model.matrix(lme4::nobars(ziformula), newdata0)
beta.zi = fixef(zinbm3)$zi
pred.zi = X.zi %*% beta.zi
```

These are estimates of the linear predictors (i.e., predictions on the link scale: logit(prob) and log(cond)), not the predictions themselves. The easiest thing to do for the point estimates of the unconditional count (ucount) is to transform to the response scale and multiply:

```
pred.ucount = exp(pred.cond)*(1-plogis(pred.zi))
```

For the standard errors/confidence intervals, we could use posterior predictive simulations (i.e., draw multivariate normal samples from the parameter for the fixed effects). This conditions on/ignores uncertainty in the random-effect parameters.

```
library(MASS)
set.seed(101)
pred.condpar.psim = mvrnorm(1000,mu=beta.cond,Sigma=vcov(zinbm3)$cond)
pred.cond.psim = X.cond %*% t(pred.condpar.psim)
pred.zipar.psim = mvrnorm(1000,mu=beta.zi,Sigma=vcov(zinbm3)$zi)
pred.zi.psim = X.zi %*% t(pred.zipar.psim)
pred.ucount.psim = exp(pred.cond.psim)*(1-plogis(pred.zi.psim))
```

```
ci.ucount = t(apply(pred.ucount.psim,1,quantile,c(0.025,0.975)))
ci.ucount = data.frame(ci.ucount)
names(ci.ucount) = c("ucount.low","ucount.high")
pred.ucount = data.frame(newdata0, pred.ucount, ci.ucount)
```

These predicted counts should be close to the median counts, so we plot them together to compare.

```
real.count = ddply(Salamanders, ~spp+mined, summarize, m=median(count), mu=mean(count))
ggplot(pred.ucount, aes(x=spp, y=pred.ucount, colour=mined))+geom_point(shape=1, size=2)+
  geom_errorbar(aes(ymin=ucount.low, ymax=ucount.high))+
  geom_point(data=real.count,  aes(x=spp, y=m, colour=mined), shape=0, size=2)+
  geom_point(data=real.count,  aes(x=spp, y=mu, colour=mined), shape=5, size=2)+
  ylab("Abundance \n including presences and absences")+
  xlab("Species")
```

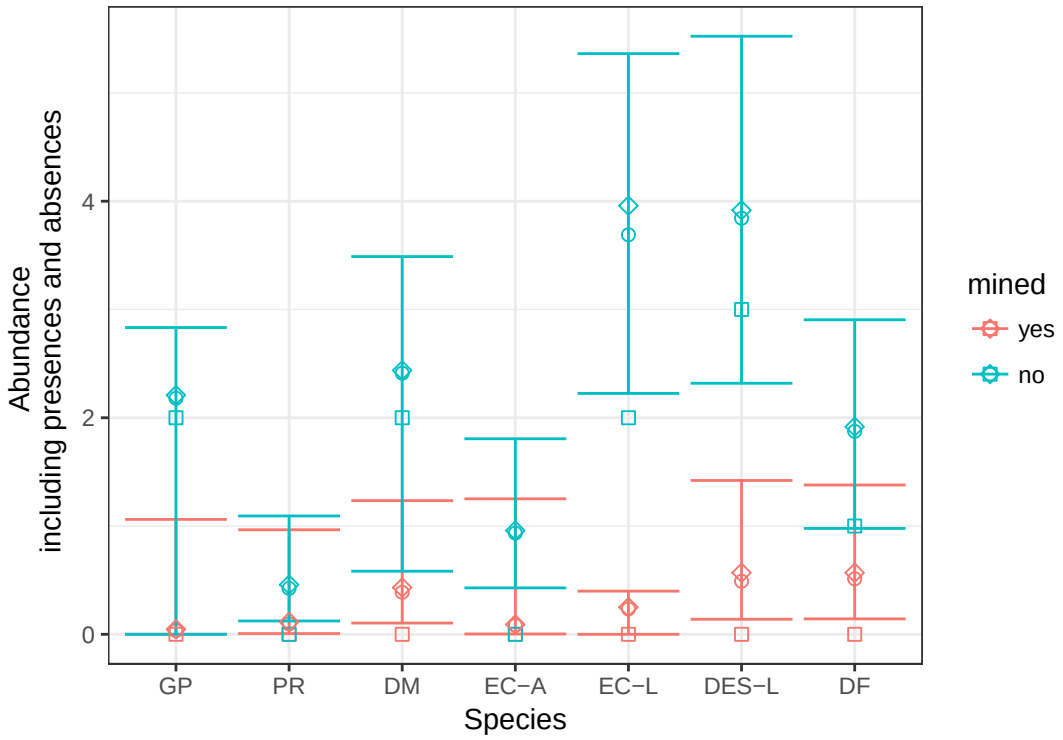


Figure 5: Estimated abundance at mode. Circles represent predicted unconditional counts at the mode (i.e., site effect = 0) and error bars represent the 95% confidence intervals for that mode. Squares represent the observed median and diamonds represent observed means calculated across samples and sites. In this highly skewed data, the mode is closer to the mean than the median.

Simulating from a fitted model

We could also examine the distribution of simulated values from the best fitted model. For this we use the function `simulate.glmmTMB`. This function works for zero-inflated and hurdle models as well as less complex models.

```
sims=simulate(nbm2, seed = 1, nsim = 1000)
```

This function returns a list of vectors. The list has one element for each simulation (`nsim`) and the vectors are the same shape as our response variable.

```
simdatlist=lapply(sims, function(count){
  cbind(count, Salamanders[,c('site', 'mined', 'spp')])
})
simdatsums=lapply(simdatlist, function(x){
  ddply(x, ~spp+mined, summarize,
        absence=mean(count==0),
        mu=mean(count))
```

```
})
ssd=do.call(rbind, simdatsums)
```

Then we can plot them with the observations summarized in the same way.

```
real = ddply(Salamanders, ~spp+mined, summarize,
             absence=mean(count==0),
             mu=mean(count))
ggplot(ssd, aes(x=absence, color=mined))+
  geom_density(adjust=4)+
  facet_wrap(~spp)+
  geom_point(data=real, aes(x=absence, y=1, color=mined), size=2)+
  xlab("Probability that salamanders are not observed")+ylab(NULL)
```

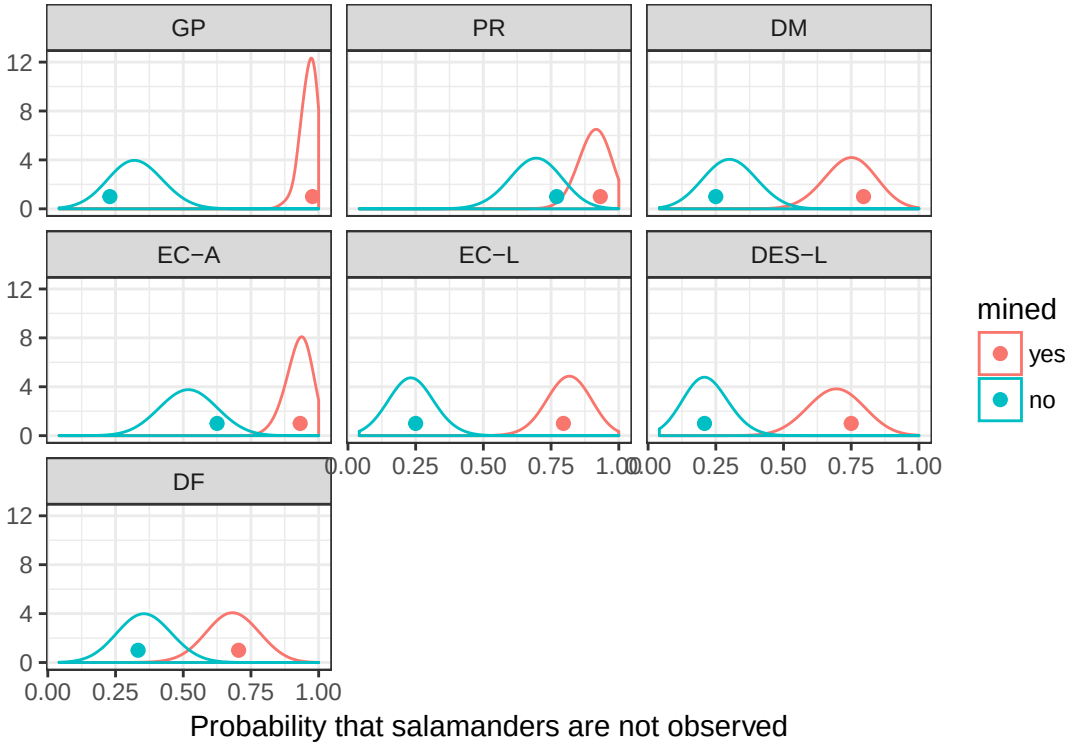


Figure 6: Simulated zero counts. Each panel represents a different species or life stage of a species. Densities are values from 1000 data sets simulated from our best fit model. Points represent the observed data.

We can see that this model does a good job of capturing the observed zero counts.

```
ggplot(ssd, aes(x=mu, color=mined))+
  geom_density(adjust=4)+
  facet_wrap(~spp)+
  geom_point(data=real, aes(x=mu, y=.5, color=mined), size=2)+
  xlab("Abundance including presences and absences")+ylab(NULL)
```

Appendix B: Compare zero-inflated mixed models across R packages

In this appendix, we analyze counts of begging behavior by owl nestlings. This example previously appeared in similar forms (Zuur et al., 2009) and (Bolker et al., 2013); the data were originally published by (Roulin and Bersier, 2007). The response variable is the number of calls from chicks (`NCalls`) in a nest. Two changes from the example published in (Bolker et al., 2013) are that (1) we use an observation-level random effect to account for overdispersion; (2) instead of assuming that the number of calls is strictly proportional to brood size (i.e., using an offset of log(brood size)), we fit the model with log(brood size) as a predictor, equivalent to assuming that calls $\propto$ (brood size)$^\gamma$, with γ not necessarily equal to 1.

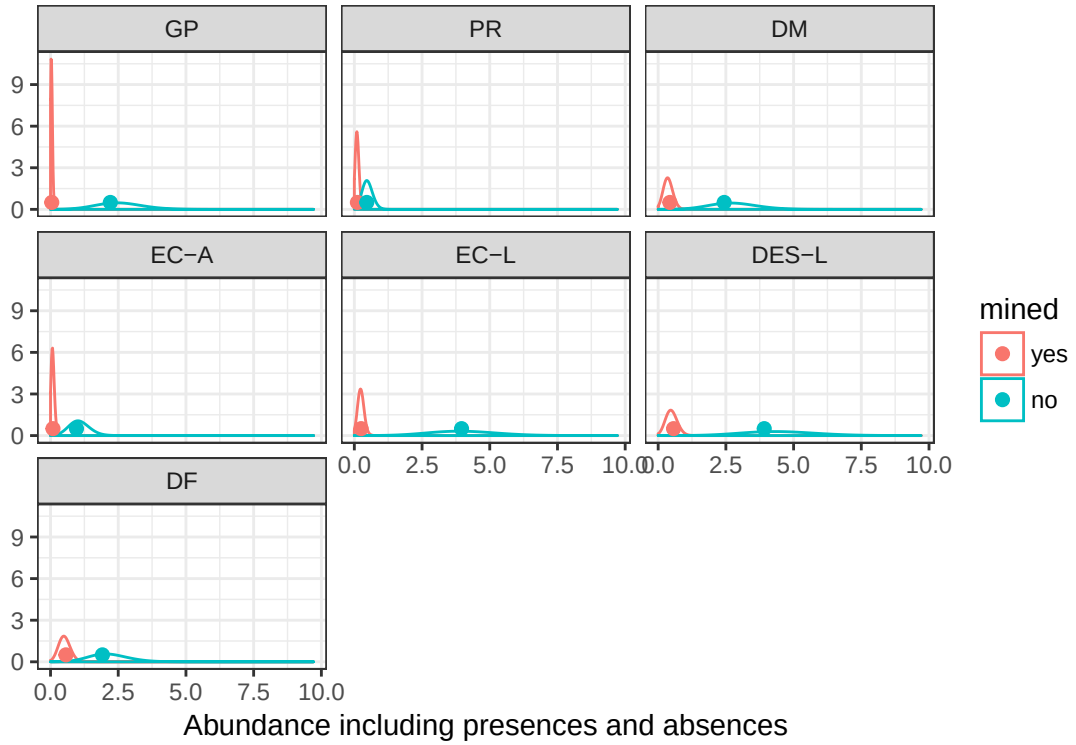


Figure 7: Simulated unconditional abundances. Each panel represents a different species or life stage of a species. Densities are values from 1000 data sets simulated from our best fit model. Points represent the observed data.

Since nests were repeatedly measured, `Nest` is included as a random effect; observation-level random effects are incorporated to allow for overdispersion (Elston et al., 2001; Hadfield, 2010). Covariates of interest include the sex of the parent visiting the nest (`SexParent`), whether the chicks were satiated or not (`FoodTreatment`), and the timing of the parent's arrival (`ArrivalTime`).

Preliminaries

Load packages

```
library(glmmTMB)
library(glmmADMB)
library(MCMCglmm)
library(brms)
library(INLA)
library(mgcv)
library(broom) #for tidy devtools::install_github("bbolker/broom")
library(plyr)
library(dplyr) #tidyverse
library(ggplot2); theme_set(theme_bw())
library(ggstance)#for position_dodgev
```

Data organization and helper functions (hidden)

```
data(Owls,package="glmmTMB")
Owls = plyr::rename(Owls, c(SiblingNegotiation="NCalls"))
Owls = transform(Owls,
                 ArrivalTime=scale(ArrivalTime, center=TRUE, scale=FALSE),
                 obs=factor(seq(nrow(Owls))))
```

Constant zero-inflation

Here we fit the model with zero-inflation assumed to be constant across the data set, i.e., zero-inflation is independent of the predictor variables.

glmmTMB

```
fixef1 = NCalls~(FoodTreatment + ArrivalTime) * SexParent + logBroodSize
form1 =  update(fixef1, . ~ . + (1|Nest)+(1|obs))
time.tmb = tfun(m1.tmb <<- glmmTMB(form1,
                                   ziformula=~1, data = Owls,
                                   family=poisson))
```

glmmADMB

With the additions to the model (`logBroodSize` as a covariate and observation-level random effects), we were unsuccessful in fitting the model with **glmmADMB**. Some variants (e.g., with observation-level random effects, but with `logBroodSize` as an offset) were possible by modifying control settings (i.e. `admb.opts=admbControl(shess=FALSE,noinit=FALSE)`), but even when successful these fits were very slow (>10 minutes).

MCMCglmm

Code for this example was modified from Bolker et al. (2013); a more complete description appears in the supplementary material for that paper.

```
fixef2 = NCalls~trait-1+ # intercept terms for both count and binary terms
    # other fixed-effect terms only apply to count term
    at.level(trait,1):logBroodSize+
    at.level(trait,1):((FoodTreatment+ArrivalTime)*SexParent)
nfix = 8
# residual variances independent for count and binary terms;
#    fixed to 1 for binary term
# random-effects variances independent for count and binary terms;
#    fixed very small (1e-6) for binary term
prior_overdisp  = list(R=list(V=diag(c(1,1)),nu=0.002,fix=2),
                       G=list(list(V=diag(c(1,1e-6)),nu=0.002,fix=2)))
prior_overdisp_broodoff = c(prior_overdisp,
                            list(B=list(mu=rep(0,nfix),
                                        V=diag(rep(1e4,nfix)))))
set.seed(101)
time.MCMCglmm = tfun(m1.MCMCglmm <<- MCMCglmm(fixef2,
                                    rcov=~idh(trait):units,
                                    random=~idh(trait):Nest,
                                    prior=prior_overdisp_broodoff,
                                    data=Owls,
                                    nitt=103000,
                                    thin=100,
                                    family="zipoisson",
                                    verbose=FALSE))
```

We adjusted the number of samples until the effective sample size of the most poorly sampled parameter (the zero-inflation parameter, in this case) was greater than 500; with 100,000 samples after a burn-in of 3,000, we achieved a minimum effective sample size of 568 (for the zero-inflation parameter).

brms

```
time.brms = tfun(m1.brms <<- brm(form1, data = Owls,
                                 iter=1000,
                                 family="zero_inflated_poisson"))

#> Compiling the C++ model
```

```
#> Start sampling
```

One of the known advantages of Hamiltonian Monte Carlo, as implemented in Stan (on which **brms** is built), is that it achieves high effective sample size per MCMC step; we were able to cut down the number of samples considerably from the default of 2000 and still achieve a minimum effective sample size of approximately 500 (the minimum effective sample size achieved was 460).

We resample to estimate the time required for sampling only (i.e., not including model compilation

```
time.brms2 = tfun(m1.brms2 <<- update(m1.brms))
```

```
#> Start sampling
```

INLA

```
form2 = update(fixef1, . ~ . + f(Nest, model="iid") + f(obs, model="iid"))
time.inla = tfun(m1.inla <<-
       inla(form2,
            family= "zeroinflatedpoisson1",
            data=Owls))
```

mgcv

To the best of our knowledge, `mgcv::gam` is currently unable to fit models with the combination of zero-inflation and overdispersion (the `ziplss()` famiy that handles zero-inflation handles only a ZIP case rather than zero-inflated negative binomial or other extensions, and observation-level random effects cannot be fit with `gam`'s random-effect approach). To address this issue, we fitted a GAM with zero-inflated Poisson responses and used a quasi-likelihood approach: i.e., estimating overdispersion as [sum of squared Pearson residuals/residual degrees of freedom] and inflating the parameter standard errors by the square root of the overdispersion.

```
form3 = update(fixef1, . ~ . + s(Nest, bs="re"))
time.mgcv = tfun(m1.gam <<- gam(list(form3, ~ 1),
                                data = Owls,
                                family = ziplss(), method = "REML"))
```

Comparing the results

Timings:

	time (sec)
mgcv	0.2
glmmTMB	2.5
INLA	4.2
MCMCglmm	53.0
brms (no compilation)	55.2
brms (with compilation)	115.6

The deterministic methods (`gam`, `glmmTMB` and `inla`) were all fast; `gam` was fastest, because we fitted a simpler model (see above). The stochastic methods (`MCMCglmm` and `brm`) were about an order of magnitude slower.

Because we ran **brms** with flat priors, the estimates are very close to the ML estimates of **glmmTMB**. The most different results are from `gam`, because we used a quasi-likelihood model with a slightly different implicit variance scaling.

Complex zero-inflation

Here we fit the model with zero-inflation depending on some of the predictor variables. We can no longer use **glmmADMB** or **INLA** (**INLA** allows the zero-inflation probabilities to depend on covariates in hurdle models — "type 0" in the INLA documentation — but not for zero-inflated models).

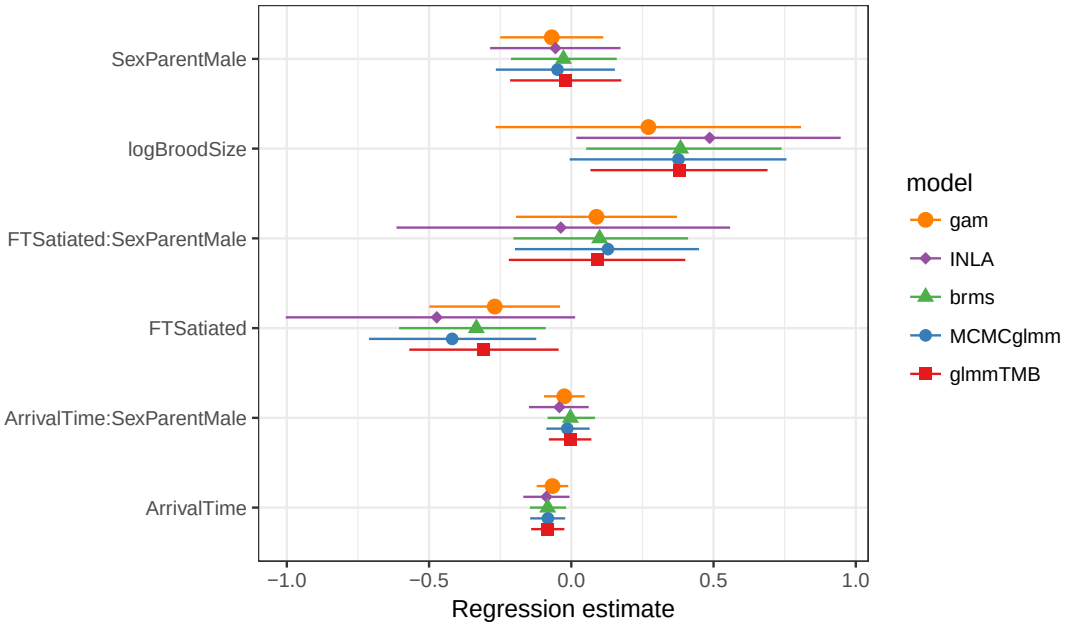


Figure 8: Estimated fixed-effect coefficients: Estimates are from similar zero-inflated Poisson models fit using functions `glmmTMB`, `MCMCglmm`, `brm`, `inla`, and `gam`.

glmmTMB

```
form0 = update(fixef1, . ~ . + (1|Nest))
ziform = ~FoodTreatment+(1|Nest)
time.tmb_czi = tfun(m1.tmb_czi <<- glmmTMB(form0,
                                           ziformula=ziform,
                                           data = Owls, family=nbinom2))
```

Attempting to fit the **glmmTMB** model with a log Normal-Poisson model (i.e., a Poisson model with observation-level random effects) and covariate-dependent zero-inflation led to convergence failure, so we substituted a similar model (a negative-binomial model without observation-level random effects).

MCMCglmm

```
fixef3 = NCalls~trait-1+ # intercept terms for both count and binary terms
    # fixed-effect terms for count term
    at.level(trait,1):logBroodSize+
    at.level(trait,1):((FoodTreatment+ArrivalTime)*SexParent)+
    # fixed-effect terms for binary term
    at.level(trait,2):FoodTreatment
nfix = 9
# residual variances independent for count and binary terms;
#    fixed to 1 for binary term
# random-effects variances now allow estimated variance for binary term
#   as well
prior_overdisp_czi  = list(R=list(V=diag(c(1,1)),nu=0.002,fix=2),
                       G=list(list(V=diag(c(1,1)),nu=0.002)))
prior_overdisp_broodoff_czi = c(prior_overdisp_czi,
                                list(B=list(mu=rep(0,nfix),
                                            V=diag(rep(1e4,nfix)))))
set.seed(101)
time.mcmc_czi=tfun(m1.mcmc_czi <<- MCMCglmm(fixef3,
                                    rcov=~idh(trait):units,
                                    random=~idh(trait):Nest,
                                    prior=prior_overdisp_broodoff_czi,
                                    data=Owls,
                                    nitt=153000,
```

```
                                  family="zipoisson",
                                  verbose=FALSE))
## warning message suppressed ...
```

Minimum effective sample size: 884

brms

```
time.brms_czi = tfun(m1.brms_czi <<- brm(brmsformula(form1, zi=ziform),
                                 data = Owls,
                                 family="zero_inflated_poisson"))

#> Compiling the C++ model

#> Start sampling

time.brms_czi2 = tfun(m1.brms_czi2 <<- update(m1.brms_czi))

#> Start sampling
```

mgcv

```
time.mgcv_czi = tfun(m1.gam_czi <<- gam(list(form3,
                                             ~FoodTreatment+ s(Nest, bs = "re")),
                                        data = Owls, family = ziplss(), method = "REML"))
```

Comparison

Timings:

	time (sec)
mgcv	0.3
TMB	3.9
brms (no compilation)	53.1
MCMCglmm	88.5
brms (with compilation)	174.4

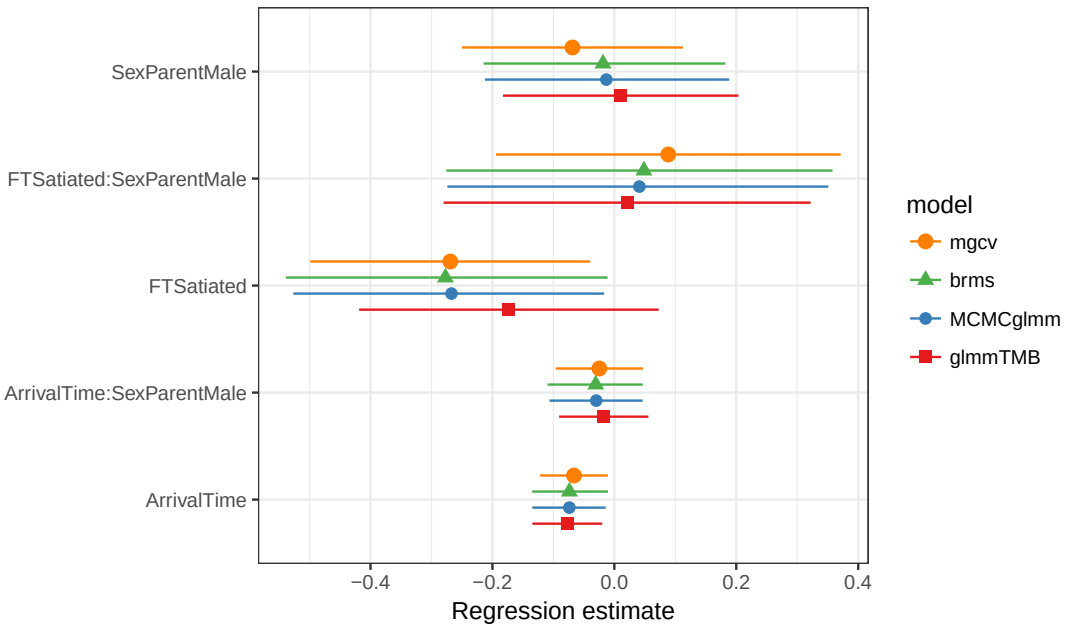


Figure 9: Estimated fixed-effect coefficients: Estimates are from the same zero-inflated Poisson model with predictors on zero-inflation fit using functions `glmmTMB`, `MCMCglmm`, `brm`, and `gam`.

Package versions:

```
#>     brms glmmADMB  glmmTMB     INLA MCMCglmm     mgcv
#>   1.10.2    0.8.5    0.2.0  17.6.20     2.25   1.8.20
```

Mollie E. Brooks
National Institute of Aquatic Resources, Technical University of Denmark
Building 201, 2800 Lyngby
Denmark
Department of Evolutionary Biology and Environmental Studies, University of Zurich
Winterthurerstrasse 190, 8057 Zurich
Switzerland
orcid.org/0000-0001-6963-8326
MollieEBrooks@gmail.com

Kasper Kristensen
National Institute of Aquatic Resources, Technical University of Denmark
Building 201, 2800 Lyngby
Denmark
kaskr@dtu.dk

Koen J. van Benthem
Department of Evolutionary Biology and Environmental Studies, University of Zurich
Winterthurerstrasse 190, 8057 Zurich
Switzerland
koenvanbenthem@gmail.com

Arni Magnusson
International Council for the Exploration of the Sea
H.C. Andersens Boulevard 44-46, 1553 Copenhagen
Denmark
arni.magnusson@ices.dk

Casper W. Berg
National Institute of Aquatic Resources, Technical University of Denmark
Building 201, 2800 Lyngby
Denmark
cbe@aqua.dtu.dk

Anders Nielsen
National Institute of Aquatic Resources, Technical University of Denmark
Building 201, 2800 Lyngby
Denmark
an@aqua.dtu.dk

Hans J. Skaug
Department of Mathematics, University of Bergen
P.O. Box 7803, 5020 Bergen
Norway
Hans.Skaug@uib.no

Martin Mächler
Seminar für Statistik, ETH Zurich
8092 Zurich
Switzerland
orcid.org/0000-0002-8685-9910
Maechler@stat.math.ethz.ch

Benjamin M. Bolker
Department of Mathematics and Statistics, McMaster University
1280 Main St W, L8S4L8 Hamilton, Ontario
Canada
Department of Biology, McMaster University
1280 Main St W, L8S4L8 Hamilton, Ontario

Canada
orcid.org/0000-0002-2127-0443
Bolker@mcmaster.ca

Simulating Probabilistic Long-Term Effects in Models with Temporal Dependence

by Christopher Gandrud, Laron K. Williams

Abstract The R package **pltesim** calculates and depicts probabilistic long-term effects in binary models with temporal dependence variables. The package performs two tasks. First, it calculates the change in the probability of the event occurring given a change in a theoretical variable. Second, it calculates the rolling difference in the future probability of the event for two scenarios: one where the event occurred at a given time and one where the event does not occur. The package is consistent with the recent movement to depict meaningful and easy-to-interpret quantities of interest with the requisite measures of uncertainty. It is the first to make it easy for researchers to interpret short- and long-term effects of explanatory variables in binary autoregressive models, which can have important implications for the correct interpretation of these models.

Introduction

Scholars from a wide variety of academic disciplines study phenomena with binary outcomes. This includes the study of war or peace (Beck et al., 1998), civil war or stability (Collier et al., 2003), wildlife habitat selection (Keating and Cherry, 2004), automobile accident severity (Al-Ghamdi, 2002), banking decisions (Maddala and Trost, 1982), labor force participation (Mroz, 1987), individual decisions about drinking water sources (Gelman et al., 2004), conflicts over water resources (Gleditsch et al., 2006), and education policy (Bailey et al., 2016), just to name a diverse few.

The desire to generalize produces the incentive for scholars to incorporate information both over time and across units, which results in time-series cross-sectional data. While helpful from an inferential standpoint, modeling processes that vary across time and space increase the number of potential estimation and interpretation problems facing scholars.

One problem that is unique to scholars examining binary time-series cross-sectional (BTSCS) data is the role of temporal dependence, or the notion that the probability of the occurrence of the event (i.e., the dependent variable) depends in part on how much time has passed since the previous occurrence. Whenever scholars estimate BTSCS models where there are omitted (or potentially unobservable) variables that are also correlated with time, there is a substantial risk of incorrect standard errors and highly misleading results (Beck et al., 1998).

Beck et al. (1998, 1261) offered a ground-breaking solution to this inferential obstacle by noting that "BTSCS data are grouped duration data", which implies that one can borrow techniques from duration analysis to properly model the influence of time since the previous event at some time t. This discovery led to a drastic increase in the number of scholars, political scientists in particular, employing duration modeling techniques with BTSCS data. Notable alternatives include dummy variables representing each value of t, splines, and cubic polynomials (Beck et al., 1998; Carter and Signorino, 2010). Put simply, these approaches assume that, for possibly un-modeled reasons, the probability of the event occurring at time t is a function of how much time has elapsed since the event previously occurred.

At the same time, another movement has produced meaningful improvements in the interpretation of dynamic models in the social sciences. Over the last decade or so, scholars have improved our understanding of the various short- and long-term effects that arise from dynamic models. These long-term effects can take the form of long-range multipliers in autoregressive distributed lag models (De Boef and Keele, 2008) or dynamic simulations in models with lagged dependent variables (Williams and Whitten, 2012). In addition to providing a more complete picture of the inferences of key theoretical variables, one reason for the explosion in scholarly attention is the emphasis on providing appropriate measures of uncertainty with easy-to-implement software packages (e.g. Williams and Whitten, 2011; Gandrud et al., 2016; Choirat et al., 2017).

To this end, this article introduces the R (R Core Team, 2017) package **pltesim**, which utilizes simulation methods to depict **p**robabilistic **l**ong-**t**erm **e**ffects in binary models with temporal dependence (PLTE). The package is available from the Comprehensive R Archive Network (CRAN). The package follows the methodology introduced in Williams (2016). In the remainder of the paper, we will first discuss the methodological principles at work, then the process **pltesim** uses to calculate probabilistic long-term effects, and finally an example with various visualization approaches.

Long-term effects in models with temporal dependence

Probabilistic long-term effects are the product of the intersection of two methodological trends: controlling for unmodeled duration dependence by including temporal dependence variables and interpreting short- and long-term effects of explanatory variables in autoregressive models. For example, if one is interested in the effects of X on the probability of Y at time t, and one controls for temporal dependence in any of the ways stated above, then X will have both a short-term effect (interpreted in the traditional manner based on the link function) and a long-term effect. However, contrary to the calculation of long-term effects in dynamic models of continuous dependent variables (De Boef and Keele, 2008), the long-term effects in BTSCS models are probabilistic. As Williams (2016, 247) notes, "modifying the values of any of the independent variables at time t potentially influences the predicted probabilities of the outcome in future time periods by forcing *time since previous event* to revert back to 0, which itself affects the probability of observing the event".

Calculating probabilistic long-term effects involves a two-step process. The first step finds the change in the predicted probability of the outcome, given a change in the independent variable (X_K), and a particular configuration of values of the other independent variables (or simulation scenario, $\mathbf{X}_C$). More formally, $\Delta\Pr(\hat{y} = 1|\mathbf{X}_C, \Delta X_K)$. A long-term effect occurs (by changing the values of the temporal dependence variables at future observations) if the observed outcome, $\hat{y} = 1|\mathbf{X}_C$ changes as a result of the change in X_K. The problem is that since this is a counterfactual, we never observe the actual outcome (just its probability). The change in the predicted probability of the event is typically the quantity of interest, and often is the point of emphasis when researchers interpret their results. In the calculation of PLTE, this quantity has a secondary interpretation as the likelihood of a variable having a PLTE. This is the change in the probability that $\hat{y} = 1|\mathbf{X}_C$, which also reflects the change in the probability that the time since previous event variables are reset to 0 at time $t + 1$.[1]

The second step is to calculate the long-term effect (LTE). Assume that we have modeled temporal dependence in a simple fashion, with *time* representing a counter based on how many time periods have elapsed since the last event. The long-term effect, then, is the difference in the probability of the event occurring at time $t + 1$ to $t + k$, given that an event occurred at time t, compared to the probability, given that the event did not occur at time t. Put another way, the LTE is a sequence of moving differences in the probability for two points along the hazard rate: one that assumes the event occurred at time t and one that does not. If we use the notation that we establish above, we first set up a simulation scenario ($\mathbf{X}_C$) containing the values of the independent variables (typically this would be the mean or median values) including *time* ($\bar{t}$). We then compare the probabilities of the event for this scenario—assuming that the *time* variable increases at each time period—to the scenario where the event occurred at time t and the value of time resets to 0 at time $t + 1$. The long-term effect at time $t + 1$ is the following (L. K. Williams 2016, 248):

$$\text{LTE}_{\mathbf{X_C}}^{t+1} = \Pr(\hat{y} = 1|\mathbf{X_C}, time = 0) - \Pr(\hat{y} = 1|\mathbf{X_C}, time = \bar{t}).$$

Then the LTE is calculated at time $t + 2$ by updating the values of *time* in both scenarios:

$$\text{LTE}_{\mathbf{X_C}}^{t+2} = \Pr(\hat{y} = 1|\mathbf{X_C}, time = 1) - \Pr(\hat{y} = 1|\mathbf{X_C}, time = \bar{t} + 1).$$

And so on, up to a value of k, which represents the maximum or some other intuitive value of *time*. It is important to note that *time*—in addition to all the other temporal dependence variables derived from *time* such as splines or cubic polynomials—must be updated at each time period.

These probabilistic long-term effects can be modified so that they reflect a wide variety of quantities of interest. For example, scholars can easily depict the PLTE of a short-term change in X_K (such as a one-unit change) or more lasting or permanent shocks in X_K. Figures can also depict the possibility of compounded effects, or the fact that having an event occur in one counterfactual increases the probability of future events. Finally, PLTE can provide interesting illustrations of the lasting effects of X_K in models that explicitly model non-proportional hazards, such as when time is interacted with X_K. In the next section we provide an overview of the process of estimating PLTE using **pltesim**.

pltesim Process

pltesim is the only tool we know of that makes it easy to calculate and visualize probabilistic long-term effects in binary models with temporal dependence. **pltesim** has four steps:

[1] Two other considerations are important here. First, since these quantities are all based on estimates, then scholars interested in hypothesis testing must use the appropriate measures of uncertainty. Second, the quality of these quantities depends on the model's fit, so the interests of transparency requires that scholars provide measures of model fit.

1. Find the parameter estimates. Currently **pltesim** works with binary outcome models, e.g. logit. So use a binary response with the `glm` function included with the default R installation.
2. Create a counterfactual scenario in a `data.frame` class object. This should have a row with the fitted counterfactual values and columns with names matching variables in your fitted model. All variables without values will be treated as having values of 0 in the counterfactual.
3. Simulate the long-term effects with **pltesim**'s `plte_builder` function.
4. Plot the results with **pltesim**'s `plte_plot` function.

In the next section we use simulated data from Williams (2016) to illustrate these steps.

Examples

The following examples replicate panels from Figure 1 in Williams (2016, 249). We start by loading the necessary packages:

```
library(pltesim)
library(ggplot2)
```

Notice that **ggplot2** is loaded. It will be used later in this section to customize the plots created by **pltesim**.

The simulated data we will use in these examples is packaged with **pltesim**. It is called `negative_year`. The name refers to the simulated data having negative duration dependence. It has the following form:

```
data("negative_year", package = "pltesim")

head(negative_year)
```

```
#>   group year y           x
#> 1     1 1991 0 -1.04320703
#> 2     1 1992 0  0.56581828
#> 3     1 1993 0 -1.21016176
#> 4     1 1994 0  0.07632362
#> 5     1 1995 0 -0.40669992
#> 6     1 1996 1  0.44269959
```

where `y` is the binary response, `x` is the non-*time* independent variable, `year` is the *time* variable, and `group` identifies each section of the panel.

Before finding the parameter estimates from this data, we need to create a standardized *time* variable that is in terms of time periods from the last spell (or the beginning of the observation period if left-censored), rather than years. **pltesim** includes the `btscs` function to accomplish this:[2]

```
neg_set <- pltesim::btscs(df = negative_year, event = "y", t_var = "year",
                 cs_unit = "group")
```

where `df` specifies the data frame. `event` is the binary response variable where `1` indicates an event, `0` otherwise. `t_var` specifies the time variable, `cs_unit` specifies the cross-sectional unit. The resulting data frame has the form:

```
head(neg_set, n = 10)
```

```
#>    group year y           x spell_time
#> 1      1 1991 0 -1.04320703          1
#> 2      1 1992 0  0.56581828          2
#> 3      1 1993 0 -1.21016176          3
#> 4      1 1994 0  0.07632362          4
#> 5      1 1995 0 -0.40669992          5
#> 6      1 1996 1  0.44269959          6
#> 7      2 1991 0 -1.25522659          1
#> 8      2 1992 0  0.29738988          2
#> 9      2 1993 0  1.00741250          3
#> 10     2 1994 0 -0.42211204          4
```

[2] `btscs` is based on a function by the same name from the R package **DAMisc** which itself is based on the Stata (StataCorp, 2009) command implementing the procedure in Beck et al. (1998). `btscs` was included in **pltesim** to (a) allow improvements for handling single period spells, (b) match **pltesim**'s syntax for ease of use within one workflow, and (c) to reduce **pltesim**'s dependencies. It also starts the spell time counter at 1 rather than 0.

This is the same data frame as before with the addition of a `spell_time` column containing a counter of time periods within each spell.

Now estimate the parameters:

```
m1 <- glm(y ~ x + spell_time + I(spell_time^2) + I(spell_time^3),
          family = binomial(link = "logit"), data = neg_set)
```

Note the inclusion of the `I` interpretation function to create the squared and cubed versions of `spell_time`.[3] Additionally, the `bs` function from the splines package, included with R, allows similar inclusion of polynomial splines for *time* using the B-spline basis.

The change in x is specified with:

```
x_change <- data.frame(x = 0.5)
```

which can be passed to `plte_builder` along with the fitted model object (`m1`). The counterfactual must be in the form of a data frame with column names matching each variable in the model and one row of fitted values. Variables from the model not included in the fitted value data frame will be treated as 0.

The counterfactual is passed to `plte_builder` with the `cf` argument. The fitted model object is specified with `obj`. The time variable is identified with `obj_tvar`. Information about how long the change in x persists in the simulation is given with `cf_duration`. It is `permanent` by default. The time period from the last spell over which to simulate the effects is given with the `t_points` argument.

The first simulation example finds the estimated impact of the counterfactual lasting for one time period. To do this, the `plte_builder` function's `cf_duration` argument is set to `"one-time"`.

```
sim1 <- plte_builder(obj = m1, obj_tvar = "spell_time",
                     cf_duration = "one-time",
                     cf = x_change, t_points = c(13, 25))
```

Running this code simulates a one period increase in `x` by 0.5 that occurs at 13 time points from the last spell. By default the central 95 percent interval of 1,000 simulations is returned. The extent of the returned central interval can be specified with `plte_builder`'s `ci` argument and the number of simulations can be adjusted with the `nsim` argument.

We can now plot the results with the `plte_plot` function:

```
plte_plot(sim1) +
    scale_y_continuous(limits = c(0, 0.4))
```

The first dot from the left in Figure 1 (and vertical dashed line) represents the median simulated baseline probability (and central 95 percent simulation interval) of the event occurring given the simulation scenario $\Pr(\hat{y} = 1|\mathbf{X}_C)$. The second dot from the left represents the updated probability of the event occurring given a one-time change in the variable of interest (or $\Pr(\hat{y} = 1|\mathbf{X}_C, \Delta X_K)$). In this case, the probability of the event occurring at time t given $x = 0.5$ is about 0.09. The number labels next to the dots represent the values of t in both scenarios.

One can assess whether the change in X_K produces a statistically significant change in the probability of an LTE by determining whether the confidence intervals overlap. In this case, the increase in X_K does not produce a statistically significant change in the probability for that time period. The remainder of Figure 1, however, reveals that the change in X_K has a meaningful impact on the probability in future periods by changing the probability that the t variable resets to 0. The dashed lines from $t + 1$ onwards are the 95 percent central simulation intervals for the probability of the event, given that the event did not occur at time t: $\Pr(\hat{y} = 1|\mathbf{X}_C, \textit{time} = 14 \ldots 25)$. The solid lines represent the counterfactual where $\hat{y}_t = 1$. The two vertical lines at time $t + 1$ illustrate how the value of t either resets to 0 (if $Y_t = 1$) or continues beyond its current value (if $Y_t = 0$). Of the two scenarios, the counterfactual where the event does not occur ($Y_t = 0$) is much more likely given its small probability (0.09). The difference between these two vertical lines is the visual representation of the LTE from equation 1. The intervals show that there is a statistically significant LTE from $t + 1$ until $t + 9$, at which point the central intervals overlap and there is no statistical difference between the two probabilities.

Note that because the output of `plte_plot` is a `gg` class **ggplot2** object, we can modify it using the full set of **ggplot2** functions, including in this case, the plot's y-axis limits with `scale_y_continuous`. This modification makes the plot more easily comparable with the ones that follow in this section.

To examine the effects of changes to X_K that last for the entire simulation period, we set `cf_duration = "permanent"`. The results are show in Figure 2.

[3] This will allow `plte_builder` to identify the polynomials given just the base `spell_time` variable name.

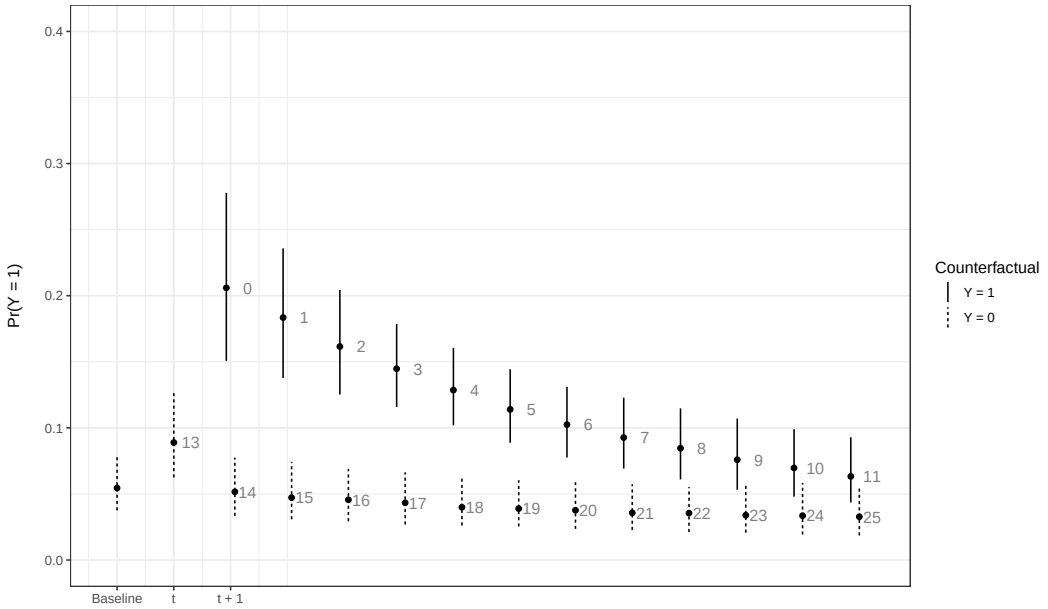


Figure 1: Simulated LTE with a one-period change in x by 0.5.

```
sim2 <- plte_builder(obj = m1, obj_tvar = "spell_time",
                     cf = x_change,
                     cf_duration = "permanent",
                     t_points = c(13, 25))

plte_plot(sim2) +
    scale_y_continuous(limits = c(0, 0.4))
```

Users can also specify changes that last for periods shorter than the entire simulation period, but longer than one-period by supplying a numeric value to `cf_duration`. For example, to have the 0.5 increase in `x` last for 4 time periods use:

```
sim3 <- plte_builder(obj = m1, obj_tvar = "spell_time",
                     cf_duration = 4,
                     cf = x_change,
                     t_points = c(13, 25))

plte_plot(sim3) +
    scale_y_continuous(limits = c(0, 0.4))
```

The results are shown in Figure 3.

Finally, we can use **pltesim** to examine not only the effects of changes in `x`, but also the compound effect of experiencing multiple events. To specify multiple events, supply an additional value to `t_points`. For example, to simulate and visualize the compound effect of an event at simulated *time* 20 use:

```
sim4 <- plte_builder(obj = m1, obj_tvar = "spell_time",
                     cf = x_change,
                     t_points = c(13, 20, 25))
plte_plot(sim4) +
    scale_y_continuous(limits = c(0, 0.4))
```

We can see in Figure 4 that in addition to the LTE given the event at time t (with probability 0.09), there is a compounding effect that results in an even larger LTE because of the event at time $t + 6$ (with probability of approximately 0.20).

Conclusion

The goal of **pltesim** is to allow researchers to easily explore and present the short- and long-term effects of models estimated with temporal dependence. These variables can have a massive influence on the

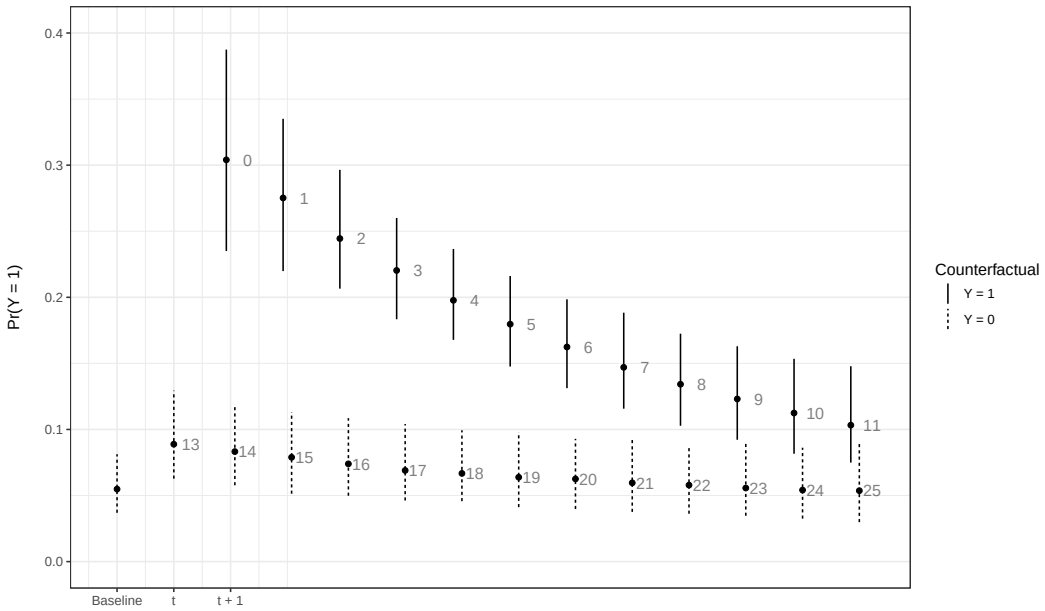


Figure 2: Simulated LTE with a permanent change in x to 0.5.

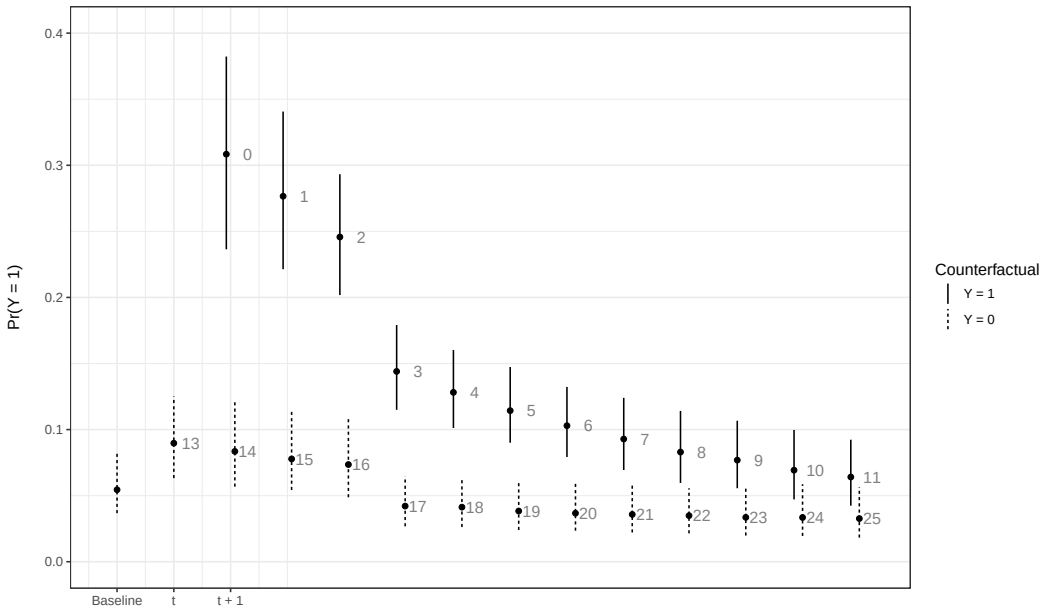


Figure 3: Simulated LTE with a four period increases of x by 0.5.

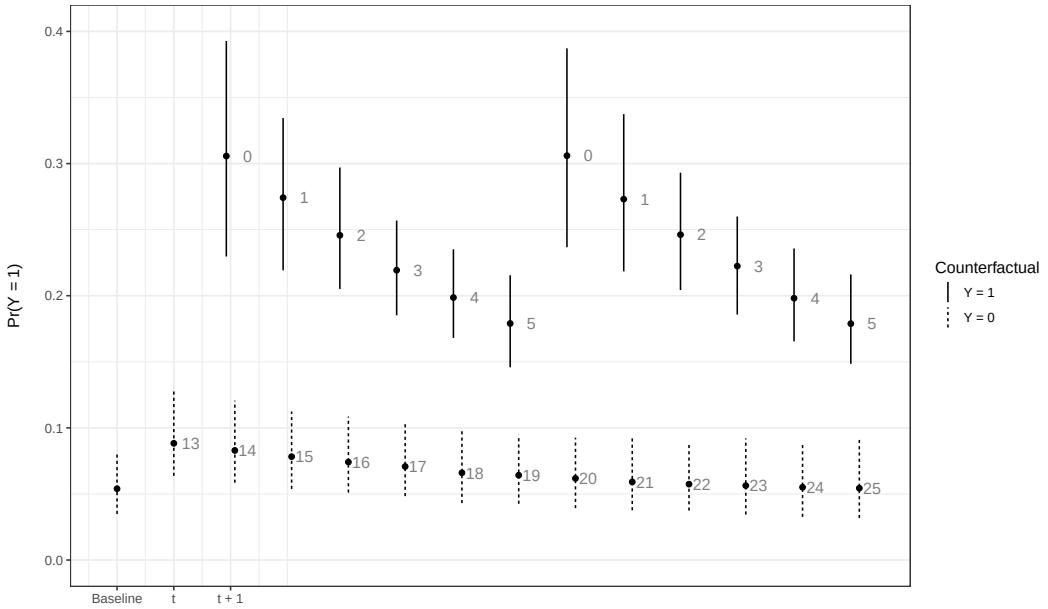


Figure 4: Simulated LTE with multiple events and a permanent increases of x by 0.5.

outcome of interest and can change the substantive effects of key theoretical variables. Prominent theories (such as the conflict trap; see Collier et al., 2003) often have expectations that their variables have long-lasting effects, or that a variable's influence grows with each recurring event. Yet, up until the introduction of **pltesim**, scholars have been unable to estimate and graphically depict these theoretically interesting long-run dynamics from BTSCS models.

Bibliography

A. Al-Ghamdi. Using logistic regression to estimate the influence of accident factors on accident severity. *Accident Analysis and Prevention*, 34(6):729–741, 2002. [p401]

M. A. Bailey, J. S. Rosenthal, and A. H. Yoon. Grades and incentives: Assessing competing grade point average measures and postgraduate outcomes. *Studies in Higher Education*, 41(9):1548–1562, 2016. [p401]

N. Beck, J. Katz, and R. Tucker. Taking time seriously: Time-series-cross-section analysis with a binary dependent variable. *American Journal of Political Science*, 42(4):1260–1288, 1998. [p401, 403]

D. B. Carter and C. S. Signorino. Back to the future: Modeling time dependence in binary data. *Political Analysis*, 18:271–292, 2010. [p401]

C. Choirat, C. Gandrud, J. Honaker, K. Imai, G. King, and O. Lau. **Zelig**: *Everyone's Statistical Software*, 2017. URL https://cran.r-project.org/package=Zelig. R package version 5.0-18. [p401]

P. Collier, V. L. Elliot, H. Hegre, A. Hoeffler, M. Reynal-Querol, and N. Sambanis. *Breaking the Conflict Trap*. Oxford University Press, New York, 2003. [p401, 407]

S. De Boef and L. Keele. Taking time seriously. *American Journal of Political Science*, 52(1):184–200, 2008. [p401, 402]

C. Gandrud, L. K. Williams, and G. D. Whitten. Visualize dynamic simulations of autoregressive relationships in r. *The Political Methodologist*, 23(2):6–10, 2016. [p401]

A. Gelman, M. Trevisani, H. Lu, and A. Van Geen. Direct data manipulation for local decision analysis as applied to the problem of arsenic in drinking water from tube wells in bangladesh. *Risk Analysis*, 24(6):1597–1612, 2004. [p401]

N. P. Gleditsch, K. Furlong, H. Hegre, B. Lacina, and T. Owen. Conflicts over shared rivers: Resource scarcity or fuzzy boundaries? *Political Geography*, 25(4):361–382, 2006. [p401]

K. A. Keating and S. Cherry. Use and interpretation of logistic regression in habitat-selection studies. *Journal of Wildlife Management*, 68(4):774–789, 2004. [p401]

G. S. Maddala and R. P. Trost. On measuring discrimination in loan markets. *Housing Finance Review*, 1 (3):245–268, 1982. [p401]

T. A. Mroz. The sensitivity of an empirical model of married women's hours of work to economic and statistical assumptions. *Econometrica: Journal of the Econometric Society*, 55(4):765–799, 1987. [p401]

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2017. URL https://www.R-project.org/. Version 3.4.0. [p401]

StataCorp. *Stata Statistical Software: Release 11*. College Station, 2009. URL http://www.stata.com/. [p403]

L. K. Williams. Long-term effects in models with temporal dependence. *Political Analysis*, 24:243–262, 2016. [p401, 402, 403]

L. K. Williams and G. D. Whitten. Dynamic Simulations of Autoregressive Relationships. *The Stata Journal*, 11(4):577–588, 2011. [p401]

L. K. Williams and G. D. Whitten. But Wait, There's More! Maximizing Substantive Inferences from TSCS Models. *Journal of Politics*, 74(03):685–693, 2012. [p401]

Christopher Gandrud
Harvard University & Zalando SE
Institute for Quantitative Social Science 1737 Cambridge Street Cambridge, MA, 02138, USA
ORCID: 0000-0003-4723-7585
christopher.gandrud@gmail.com

Laron K. Williams
University of Missouri

williamslaro@missouri.edu

RQGIS: Integrating R with QGIS for Statistical Geocomputing

by Jannes Muenchow, Patrick Schratz, Alexander Brenning

Abstract Integrating R with Geographic Information Systems (GIS) extends R's statistical capabilities with numerous geoprocessing and data handling tools available in a GIS. QGIS is one of the most popular open-source GIS, and it furthermore integrates other GIS programs such as the System for Automated Geoscientific Analyses (SAGA) GIS and the Geographic Resources Analysis Support System (GRASS) GIS within a single software environment. This and its QGIS Python API makes it a perfect candidate for console-based geoprocessing. By establishing an interface, the R package **RQGIS** makes it possible to use QGIS as a geoprocessing workhorse from within R. Compared to other packages building a bridge to GIS (e.g., **rgrass7**, **RSAGA**, **RPyGeo**), **RQGIS** offers a wider range of geoalgorithms, and is often easier to use due to various convenience functions. Finally, **RQGIS** supports the seamless integration of Python code using **reticulate** from within R for improved extendability.

Introduction

Defining a GIS as a system for the analysis, manipulation and visualization of geographical data (Longley et al., 2011), one could argue that R has become a GIS (Bivand et al., 2013). In great part this is thanks to packages that provide spatial classes and algorithms coded in and for R (despite this these packages might also link to other software outside of R). These include **maptools** (Bivand and Lewin-Koh, 2017), **raster** (Hijmans, 2017), **sp** (Bivand et al., 2013) and **sf** (Pebesma, 2017). Further packages even extend R's GIS capabilities through advanced mapping, e.g., **mapview** (Appelhans et al., 2017) and **mapmisc** (Brown, 2016), and routing, e.g., **osmar** (Eugster and Schlesinger, 2013) and **dodgr** (Padgham and Peutschnig, 2017), among others. Despite this, native R (in the sense of coded in and for R) lacks fundamental GIS capabilities. GIS topology and topological operations are only partially (**RArcInfo**, Gómez-Rubio and López-Quílez, 2005) or indirectly available via **rgrass7** (Bivand, 2017). Furthermore, R is neither a spatial database management system nor especially good at the manipulation of large data sets (Ripley et al., 2016). Hence, computationally demanding GIS operations (point cloud processing, overlay operations on 'big' spatial data) executed in R may be rather slow. Performance and scalability, of course, depend on the computer hardware, and cloud computing may eventually alleviate or even settle this problem. Yet, most R users most likely still work on a local machine. What is more, R is lacking a number of fundamental GIS operations such as the derivation of various terrain attributes from a digital elevation model (DEM). And the same is true for 3D data visualization and voxel processing (Hengl et al., 2015). Finally, though interactive tasks such as digitizing of geodata have become possible within R very recently (**mapedit**, Appelhans and Russell, 2017), extensive manual editing is better done with the help of a GIS.

Many of R's geospatial shortcomings could potentially be addressed through R programming directly. However, R was designed from the very beginning as an interactive interface to the algorithms

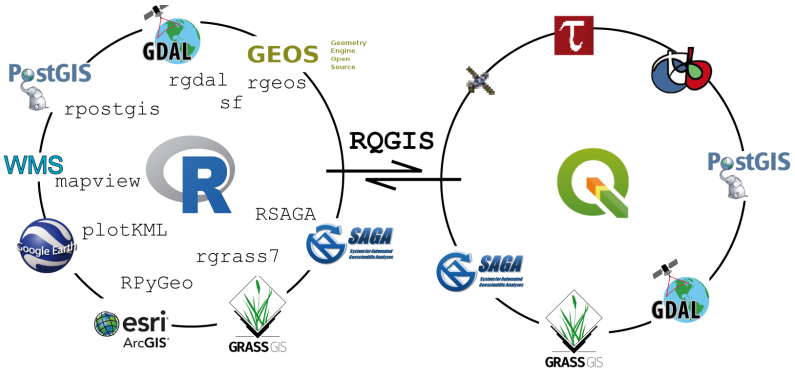


Figure 1: The R-interface to geospatial software - geospatial libraries, Desktop GIS, geobrowsers as well as web mapping and the position of **RQGIS** (left circle; WMS: Web Mapping Service). QGIS and corresponding third-party providers (right circle, the upper three symbols correspond to (from left to right): LiDAR tools, TauDEM, Orfeo Toolbox.

of other software (Chambers, 2016). Hence, it is unnecessary and even counterproductive to duplicate the functionality provided by an existing dedicated software with an expert developer and user community as long as there is a way to access it from within R. Therefore, it is barely surprising that numerous R packages provide access to third-party geoprocessing tools (Figure 1), only some of which will be discussed here. **rgdal** (Bivand et al., 2017) accesses the geospatial data abstraction library (GDAL/OGR) (GDAL Development Team, 2017). **rgeos** (Bivand and Rundel, 2017) is an interface to geometry engine - open source (GEOS, GEOS Development Team, 2017), which opens the way to GIS vector operations. However, GEOS performance is somewhat limited. Think, for instance, of the spatial union of all US American census tracts and postal code layers, and it may be quite possible that `rgeos::gUnion` may take a very long time. The successor of the **sp** package, package **sf** combines the functionality of **sp** (spatial classes), **rgdal** (here: import/export of spatial vector data) and **rgeos** (geometrical operations) in just one package. Note also that GEOS is a C API for topology operations on geometries. Consequently, it expects topologically correct data. To make sure that our geodata lives up to topological expectations in general, our best approach is probably through another third-party integration, namely R-GRASS (Bivand, 2007, 2017). Additionally, GRASS GIS comprises a large suite of vector and raster functions. Basically, the user has to set up a spatial database before being able to use GRASS's geoprocessing utilities (Neteler and Mitasova, 2008). Hence, less experienced GIS users will likely prefer faster-to-use GIS interfaces also providing extensive geoprocessing capabilities. In particular, **RSAGA** (Brenning et al., 2008) integrates R with SAGA (Conrad et al., 2015) and **RPyGeo** (Brenning, 2012b) provides an interface to ArcGIS (ESRI, 2017), which is probably still the most popular GIS environment in the world with >1 million users and the greatest market share among proprietary GIS (Longley et al., 2011).

What has been missing, however, is an R interface to one of the most widely used open-source GIS, QGIS (QGIS Development Team, 2017; Graser and Olaya, 2015). So far, the QGIS processing toolbox provided only the opposite interface by letting the user integrate R scripts as a user-defined 'tool' in QGIS. This is fine for people unwilling to use R directly. However, interfacing from R to QGIS has multiple benefits to the R user community. First and foremost, native QGIS geoalgorithms are now available from within R for the first time. Moreover, it is a special feature of QGIS that it acts as an umbrella integrating various other GIS power houses under its hood. These include SAGA, GRASS, GDAL, the Orfeo Toolbox (Inglada and Christophe, 2009), TauDEM (Tarboton and Mohammed, 2017) and additional tools for light detection and ranging (LIDAR) data (Rapidlasso, 2017). **RQGIS** (Muenchow and Schratz, 2017) brings this incredibly powerful geoprocessing environment to the R console in just one package. This, however, does not mean that specialized packages such as **RSAGA** and **rgrass7** (Bivand, 2007) will become obsolete, as discussed later. **RQGIS** also aims to be user-friendly by automatically retrieving GIS function parameter names and corresponding default values as well as supporting R named arguments for geoalgorithm parameters through the ellipsis argument.

In general, R–GIS interfaces open the way to extremely powerful and innovative statistical geoprocessing as for example shown by Brenning (2008), Hengl et al. (2010), Muenchow et al. (2012), Vanselow and Samimi (2014), Brenning et al. (2015) Mergili et al. (2015), Mergili and Kerschner (2015), Poggio and Gimona (2015) and Zandler et al. (2015). In this paper we will first introduce the general architecture and main features of the **RQGIS** package. We will then demonstrate the application of this integrated scientific programming approach with an ecological example. Subsequently, we will show how to easily complement and extend **RQGIS** with Python programming, especially PyQGIS (Sherman, 2014). In our discussion, we will finally compare and contrast **RQGIS** with other approaches to R–GIS integration, and provide an outlook and motivation for future developments.

Introducing the RQGIS package

Basic concepts

The **RQGIS** package utilizes the QGIS Python API in order to access QGIS modules. To successfully run the QGIS Python API, **RQGIS** first sets up all required environment variables (Figure 2). And secondly, it establishes a tunnel to Python using **reticulate** (Allaire et al., 2017) - a package providing an R interface to Python. The older package **rPython** (Bellosta, 2015) is similar to **reticulate**, however, it is only available for Unix-based systems which is why we had to dismiss it as an option for **RQGIS**. With **reticulate**, we set up the Python environment only once, and use the resulting tunnel to exchange functions and objects between R and Python seamlessly.

We can divide **RQGIS** roughly into two major components:

- The Python code ('python_funs.py' located in 'inst/python' of **RQGIS**) defines a Python class named `"RQGIS"` with methods to be called during the geoprocessing. Defining an own class has

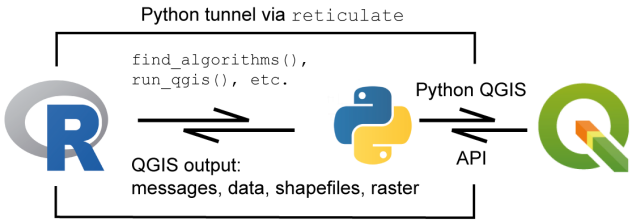


Figure 2: Conceptual model of how RQGIS calls QGIS from within R.

the additional benefit that it becomes highly unlikely that (advanced) users interacting with the QGIS Python API accidentally overwrite some of our predefined methods.

- The 'processing.R' file (found in the 'R' folder of **RQGIS**) actually establishes the QGIS Python interface, and lets the user run QGIS from within R. The most important functions are (see also Usage for a detailed description):
    1. `open_app()` to establish a tunnel to Python and a QGIS custom application
    2. `find_algorithms()` to retrieve the QGIS command-line names for all available geoalgorithms
    3. `open_help()` and `get_args_man()` to access help resources as well as function arguments and default values
    4. `run_qgis()` to call QGIS geoalgorithms from within R

The most notable features of **RQGIS** are:

- For the first time, native QGIS algorithms are available from within R.
- Additionally, **RQGIS** provides access to hundreds of third party geoalgorithms including GDAL, GRASS GIS and SAGA GIS. In the future many more integrations can be expected. For instance, there is already a plugin providing access to PostGIS geoprocessing tools (clip, dissolve, distance, etc.) available in the QGIS processing toolbox (https://plugins.qgis.org/plugins/postgis_geoprocessing/).
- R users can stay in their preferred programming language without having to touch Python.
- Convenient access to QGIS help resources facilitates the geoprocessing work flow. While `open_help` accesses the QGIS online help for a specific geoalgorithm, `get_args_man()` retrieves function arguments and their default values.
- `run_qgis()` also accepts `"sf"`, `"sp"` and `"raster"` objects as arguments. Similarly, users may directly load the QGIS output into R by setting `load_output` to `TRUE` when using `run_qgis()`.

Usage

Since **RQGIS** is an interface to various GIS software packages, the user needs to install this software beforehand. To facilitate the installation process we have written an installation guide, see http://jannes-m.github.io/RQGIS/articles/install_guide.html. Or after having installed the package, one can also access the corresponding vignette by typing:

```
vignette("install_guide", package = "RQGIS")
```

We will demonstrate the usage of **RQGIS** by showing how to compute the plan and tangential curvatures of a digital elevation model (DEM). The first thing to do is to make sure, that all paths are set correctly to successfully run the Python API from within R. Function `set_env()` facilitates this since the user only needs to specify the root path to the QGIS installation. If the root path remains unspecified, `set_env()` tries to be smart by checking the default QGIS installation directories. If this is unsuccessful, `set_env()` will try to find the QGIS installation on the computer which may be time-consuming especially on Windows machines. A much faster way is to explicitly indicate the root path. For Windows this might look like this '`qgis_env <-set_env(root = 'C:/OSGEO4~1')`'.

Subsequently, `set_env()` finds all required paths. Virtually all subsequent **RQGIS** functions require the output list of `set_env()`. This is why, **RQGIS** automatically caches the output of `set_env()`, and reuses it when required by another function later on. To establish a tunnel to the QGIS Python API, we run `open_app()`. Explicitly, the function sets all necessary paths (e.g., path to the QGIS Python binary) to successfully run QGIS, and secondly opens a QGIS custom application (i.e., outside of the QGIS GUI interface) while importing necessary Python modules.

```
library("RQGIS")
set_env()
open_app()
```

Running `set_env()` and `open_app()` is optional here since all subsequent functions dependent on their output will run them automatically in case they have not been executed before. To work in a reproducible manner, and to find out which QGIS and third-party GIS versions we are using, we execute:

```
## $root
## [1] "C:/OSGeo4W64"
##
## $qgis_prefix_path
## [1] "C:/OSGeo4W64/apps/qgis"
##
## $python_plugins
## [1] "C:/OSGeo4W64/apps/qgis/python/plugins"

info_r <- version
info_qgis <- qgis_session_info()
c(platform = info_r$platform, R = info_r$version.string, info_qgis)
## $platform
## [1] "x86_64-w64-mingw32"
##
## $R
## [1] "R version 3.4.2 (2017-09-28)"
##
## $qgis_version
## [1] "2.18.14"
##
## $gdal
## [1] "2.2.2"
##
## $grass6
## [1] "6.4.3"
##
## $grass7
## [1] "7.2.2"
##
## $saga
## [1] "2.3.2"
```

Continuing with our analysis, we need to find out the command-line name of a geoalgorithm available in QGIS that computes the curvatures from a DEM. `find_algorithms()` lets the user use regular expressions to search for a function which contains the search terms in its short description. Leaving the `search_term`-argument empty, will return all available geoalgorithms. Here, we assume that the function we are looking for contains the word 'curvature' in its short description. Setting `name_only` to `TRUE` gives back the name of the geoalgorithm instead of its name plus the corresponding short description.

```
find_algorithms(search_term = "curvature",
                name_only = TRUE)
## [1] "grass7:r.slope.aspect"             "saga:curvatureclassification"
## [3] "saga:slopeaspectcurvature"         "saga:upslopeanddownslopecurvature"
## [5] "grass:r.slope.aspect"
```

Several functions are available for our task, we will go on with function `grass7:r.slope.aspect`. To familiarize ourselves with the function, we can access its online help by calling (not shown):

```
open_help(alg = "grass7:r.slope.aspect")
```

Next, we would like to know how to use a specific geoalgorithm. `get_usage()` prints the parameters and default values for a given geoalgorithm to the console.

```
get_usage(alg = "grass7:r.slope.aspect")
## ALGORITHM: r.slope.aspect - Generates raster layers of slope
```

```
## aspect
## curvatures and partial derivatives from a elevation raster layer.
##         elevation <ParameterRaster>
##         format <ParameterSelection>
##         precision <ParameterSelection>
##         -a <ParameterBoolean>
##         zscale <ParameterNumber>
##         min_slope <ParameterNumber>
##         GRASS_REGION_PARAMETER <ParameterExtent>
##         GRASS_REGION_CELLSIZE_PARAMETER <ParameterNumber>
##         slope <OutputRaster>
##         aspect <OutputRaster>
##         pcurvature <OutputRaster>
##         tcurvature <OutputRaster>
##         dx <OutputRaster>
##         dy <OutputRaster>
##         dxx <OutputRaster>
##         dyy <OutputRaster>
##         dxy <OutputRaster>
##
##
## format(Format for reporting the slope)
##         0 - degrees
##         1 - percent
## precision(Type of output aspect and slope layer)
##         0 - FCELL
##         1 - CELL
##         2 - DCELL
```

`get_args_man()` lets us retrieve automatically a corresponding parameter-argument list. Setting the `options` parameter to `TRUE` (the default) automatically chooses the default value from a list of possible options for a parameter. This is always the first option which is in accordance with the QGIS GUI behavior. To make the user aware of the automatically chosen options, `get_args_man()` prints the corresponding values to the console.

```
params <- get_args_man(alg = "grass7:r.slope.aspect", options = TRUE)
## Choosing default values for following parameters:
## format: 0
## precision: 0
## See get_options('grass7:r.slope.aspect') for all available options.
```

`grass7:r.slope.aspect` has 17 parameters. Here, we only show the first six parameters for brevity.

```
head(params)
## $elevation
## [1] "None"
##
## $format
## [1] "0"
##
## $precision
## [1] "0"
##
## $`-a`
## [1] "True"
##
## $zscale
## [1] "1.0"
##
## $min_slope
## [1] "0.0"
```

Next, we specify the required arguments. Of course, `grass7:r.slope.aspect` expects a spatial input file, here a DEM. Conveniently, `run_qgis()` accepts as input both a path to a spatial file stored on disk or a spatial object residing in R's environment (specifically `"raster"`-, `"sp"`- and `"sf"`-objects).

Here, we use a DEM that comes as an example file with the **RQGIS** package. Note that `run_qgis()` simply saves spatial input files to a temporary output location. Hence, if the file already exists on disk, it is much more efficient to indicate the path to the file instead of loading it into R and letting `run_qgis()` export it again. By contrast, indicating output paths is not strictly necessary. If an output path parameter equals `None` (QGIS default), QGIS automatically creates an output file which it saves to a temporary processing output folder. In the code chunk below, we specifically indicate curvature outputs while keeping the default, i.e. `None`, for the remaining output parameters `slope`, `aspect`, `dx`, `dy`, `dxx`, `dyy` and `dxy` (check out the GRASS function documentation for more information, i.e. run 'open_help("grass7:r.slope.aspect")'). `run_qgis()` prints all output paths to the console if `show_output_paths` is set to TRUE. Here, we turn this behavior off for two reasons. First, we are not interested in the output paths of the seven terrain attributes we left unspecified. Secondly, we have explicitly specified the curvature output paths, i.e., we already know the corresponding output locations. We recommend to specify those output paths which are relevant to the analysis. Manual specification has the additional benefit that we can indicate a specific file format (QGIS default for raster data sets is in most cases .tif, but we might want to use e.g., .asc or SAGA's .sdat). Additionally, loading QGIS output directly back into R (`load_output`-parameter) only works with output paths specified by the user.

```
data("dem", package = "RQGIS")
out <- run_qgis(alg = "grass7:r.slope.aspect",
                elevation = dem,
                pcurvature = file.path(tempdir(), "pcurv.tif"),
                tcurvature = file.path(tempdir(), "tcurv.tif"),
                show_output_paths = FALSE,
                load_output = TRUE)
```

Note that we used R named arguments in `run_qgis()`, i.e., we assigned values or objects to the parameters whose names we have identified with the help of `get_args_man()` or `get_usage()`. We could replace the R named arguments also by a parameter-argument list. Remember that we have already created a parameter-argument list named `params` using `get_args_man()` (see above):

```
params$elevation <- dem
params$pcurvature <- file.path(tempdir(), "pcurv.tif")
params$tcurvature <- file.path(tempdir(), "tcurv.tif")
out <- run_qgis(alg = "grass7:r.slope.aspect",
                params = params,
                load_output = TRUE,
                show_output_paths = FALSE)
class(out)
## [1] "list"
names(out)
## [1] "pcurvature" "tcurvature"
```

However, providing R named arguments and a parameter-argument list is not possible, and `run_qgis()` will complain telling the user to user either one of these but not a mixture. Since we have set `load_output` to `TRUE`, `run_qgis()` automatically loads the QGIS output into R. In this case, the object `out` is a list with two `"raster"` objects. If we only had specified one output raster, `out` would have been a `"RasterLayer"` object. In case the output is a vector layer, `run_qgis()` will load it as an `"sf"` object. To have a look at the output, we can execute following code (not shown):

```
library("raster")
plot(stack(out))
```

Concerning the handling of parameter-argument pairs, `run_qgis()` uses `get_args_man()` through `pass_args()` in the background to access the default values of all missing arguments if available. If the user accidentally omits a required argument, `run_qgis()` will return an error message that informs about the missing argument. The help documentation of `pass_args()` presents a detailed list of argument checks that are run before executing `run_qgis()`.

Experienced GRASS users may wonder if there is a need to specify the `GRASS_REGION_PARAMETER`. `pass_args()` determines this parameter automatically based on the spatial layers provided as input by the user (in our example above this is `dem`). However, the `GRASS_REGION_PARAMETER` can also be set manually (see the `pass_args()` documentation for details).

Figure 3: The study area Mount Mongón in northern Peru (Landsat image: path 9, row 67, acquisition date 09/22/2000; USGS 2016).

Ecological example: combining geocomputing and statistics

To show the utility of **RQGIS** in real-world applications, we combine QGIS functionality with R's modeling and (geo-)statistical capabilities in an ecological study in the coastal desert of northern Peru (Figure 3). Despite the extreme aridity of the Mount Mongón region (200-1100 m asl), this area is the habitat of a distinct flora and fauna (Dillon et al., 2003). The unique vegetation, locally termed *lomas*, mainly survives due to heavy fog during the austral winter months (Muenchow et al., 2013b,c).

Linking species richness to environmental predictors along gradients is a key topic of community ecology and biogeography (Muenchow et al., 2017), and the fundamental basis for conservation planning (Pomara et al., 2012). In our use case, we model vascular plant species richness along an altitudinal gradient as a function of topographic and remotely sensed variables by means of count regression. In the following, we will show a simplified version of an analysis by Muenchow et al. (2013b).

Before running a Poisson model, we need to compute terrain attributes from a DEM. These will serve as predictors to model species richness. To account for the unimodal relationship between elevation and species richness, we use a second-order orthogonal polynomial function (Figure 4, Panel Elevation). In the original paper, we dropped the least significant variables one at a time until only significant predictors remained (elevation and its squared term, catchment slope, catchment area and the normalized difference vegetation index, NDVI). Due to space constraints and demonstration purposes, we will simply use the final model here (for details see Muenchow et al., 2013b). Instead of calculating all predictors used in the original paper, we only show how to derive selected geospatial predictors using **RQGIS**, namely tangential and profile curvature, catchment slope and catchment area.

Terrain attributes

The numerical representation as well as the analysis of the land surface is frequently referred to as terrain analysis and terrain modeling. The corresponding surface-characterizing measures are known as terrain attributes (Pike et al., 2008). Terrain attributes play an important role, for example, in pedometrics (McBratney et al., 2000), precision agriculture (Kühn et al., 2009), geomorphometry (Pike et al., 2008) and ecology (Muenchow et al., 2013a). They are frequently related to slope stability (Montgomery and Dietrich, 1994; Muenchow et al., 2012). Additionally, they are proxies for variables representing water availability such as soil moisture, soil texture or moisture-holding capacity, among others (Brenning, 2008; Franklin et al., 2000; Muenchow et al., 2013c). Especially the latter is of utmost importance regarding plant distribution in a desert environment. While GIS can easily calculate terrain attributes, R is rather limited in this respect. However, without terrain attributes, we would neither be

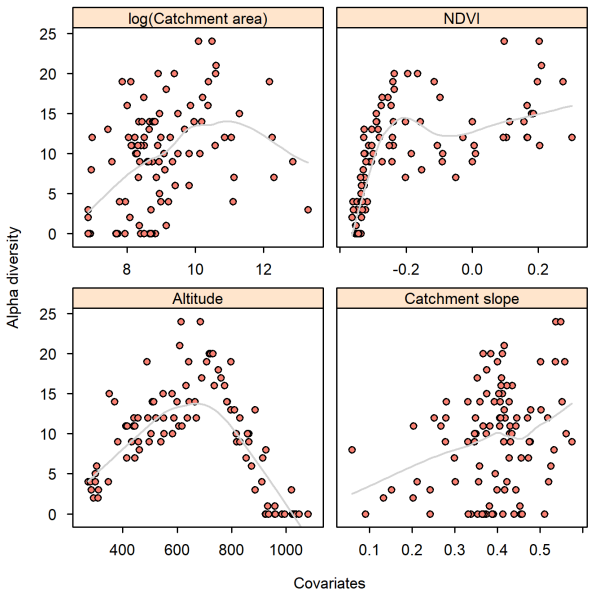


Figure 4: Scatterplot of all predictors used in the Poisson model against the response variable. Each dot represents a visited plot on Mount Mongón. The gray line smoother should aid visual inspection.

able to model nor predict species richness appropriately.

First we would like to use SAGA to remove local depressions from the DEM, since these may be artifacts. For this, we use the `[1] Fill Sinks` method. Note that you also may use numbers for specifying the option, here, the fill sinks method corresponds to 1.

```
get_usage("saga:sinkremoval")
## ALGORITHM: Sink removal
##         DEM <ParameterRaster>
##         SINKROUTE <ParameterRaster>
##         METHOD <ParameterSelection>
##         THRESHOLD         <ParameterBoolean>
##         THRSHEIGHT <ParameterNumber>
##         _RESAMPLING <ParameterSelection>
##         DEM_PREPROC <OutputRaster>
##
##
## METHOD(Method)
##         0 - [0] Deepen Drainage Routes
##         1 - [1] Fill Sinks
## _RESAMPLING(Resampling method)
##         0 - Nearest Neighbour
##         1 - Bilinear Interpolation
##         2 - Bicubic Spline Interpolation
##         3 - B-Spline Interpolation
run_qgis(alg = "saga:sinkremoval",
         DEM = dem,
         METHOD = "[1] Fill Sinks",
         DEM_PREPROC = file.path(tempdir(), "sdem.sdat"),
         show_output_paths = FALSE)
```

Next, we compute the catchment area (or upslope contributing area) and its mean slope from the preprocessed DEM. These raster datasets are calculated while calculating the 'Saga wetness index', which is also frequently used as a predictor in ecological studies. To calculate the catchment slope instead of the local slope we set the `SLOPE_TYPE` argument to 1. The names of the output rasters are 'carea.sdat' and 'cslope.sdat' both of which will be stored in `tempdir()`.

```
get_usage("saga:sagawetnessindex")
## ALGORITHM: Saga wetness index
```

```
##         DEM <ParameterRaster>
##         SUCTION <ParameterNumber>
##         AREA_TYPE <ParameterSelection>
##         SLOPE_TYPE <ParameterSelection>
##         SLOPE_MIN <ParameterNumber>
##         SLOPE_OFF <ParameterNumber>
##         SLOPE_WEIGHT <ParameterNumber>
##         _RESAMPLING <ParameterSelection>
##         AREA <OutputRaster>
##         SLOPE <OutputRaster>
##         AREA_MOD <OutputRaster>
##         TWI <OutputRaster>
##
##
## AREA_TYPE(Type of Area)
##         0 - [0] absolute catchment area
##         1 - [1] square root of catchment area
##         2 - [2] specific catchment area
## SLOPE_TYPE(Type of Slope)
##         0 - [0] local slope
##         1 - [1] catchment slope
## _RESAMPLING(Resampling method)
##         0 - Nearest Neighbour
##         1 - Bilinear Interpolation
##         2 - Bicubic Spline Interpolation
##         3 - B-Spline Interpolation
run_qgis(alg = "saga:sagawetnessindex",
         DEM = file.path(tempdir(), "sdem.sdat"),
         SLOPE_TYPE = 1,
         SLOPE = file.path(tempdir(), "cslope.sdat"),
         AREA = file.path(tempdir(), "carea.sdat"),
         show_output_paths = FALSE)
```

To have a look at the output rasters, we can run (not shown):

```
library("dplyr")
library("raster")
file.path(tempdir(), c("cslope.sdat", "carea.sdat")) %>%
  raster::stack(.) %>%
  plot
```

We furthermore need to apply some transformations to these raster files for improved interpretability. On the one hand, we convert slope angle from radians to degrees.

```
library("raster")
cslope <- raster(file.path(tempdir(), "cslope.sdat"))
cslope <- cslope * 180 /pi
```

On the other hand, we divide the catchment area variable by one million to change the unit from m^2 to km^2. Furthermore, we transform it logarithmically since it is strongly skewed to the right.

```
carea <- raster(file.path(tempdir(), "carea.sdat"))
log_carea <- log(carea / 1e+06)
```

Apart from the catchment area and catchment slope, our final model requires the NDVI as an indicator of vegetation properties. To calculate it, we would have to perform pixel-by-pixel arithmetic operations on raster data sets representing the Landsat satellite's spectral bands three (red) and four (near infrared). These so-called map algebra operations are available through **raster** and the QGIS modules `saga:rastercalculator` and `grass:r.mapcalculator`; however, the **RQGIS** package already provides the NDVI raster as an example dataset. Therefore, we merely have to attach the data to our workspace.

```
data("ndvi", package = "RQGIS")
```

To account for the nonlinear unimodal relationship between elevation and species richness, we need two rasters representing the second-order orthogonal polynomials of the original DEM. First, we

convert the elevation unit from m to km by dividing the original DEM raster by 1000. Next, we use R's built-in `poly()` function to calculate the orthogonal polynomials for each pixel in our DEM raster to avoid collinearity among the predictors. Before saving the resulting rasters and the catchment slope, the catchment area and the NDVI to a temporary output location, we apply the `crop()` function from the **raster** package to ensure that all rasters share the same extent.

```
data("dem", package = "RQGIS")
dem <- dem / 1000
my_poly <- poly(values(dem), degree = 2)
dem1 <- dem2 <- dem
values(dem1) <- my_poly[, 1]
values(dem2) <- my_poly[, 2]
for (i in c("dem1", "dem2", "log_carea", "cslope", "ndvi")) {
  tmp <- crop(get(i), dem)
  writeRaster(x = tmp,
              filename = file.path(tempdir(), paste0(i, ".asc")),
              format = "ascii",
              prj = TRUE,
              overwrite = TRUE)
}
```

After creating these raster datasets, we would like to extract their attributes to the randomly sampled points. Conveniently, RSAGA's `pick.from.ascii.grids()` function accepts multiple rasters for parallel attribute extraction. Note that `pick.from.ascii.grids()` is a pure R function, i.e., it runs without accessing SAGA. Here, we only extract the predictor variables needed in the final model (see above). The `"sf"` object `random_points` refers to randomly sampled points we visited in the field. Unfortunately, `pick.from.ascii.grids()` does not accept spatial point objects as input; instead we have to provide it with a `"data.frame"` and indicate which columns refer to the coordinates. In the `file` argument we specify the raster files from which we would like to extract values to our points. The output columns in `vals` will be named like the input rasters.

```
library("dplyr")
data("random_points", package = "RQGIS")
random_points[, c("x", "y")] <- sf::st_coordinates(random_points)
raster_names <- c("dem1", "dem2", "log_carea", "cslope", "ndvi")
vals <- RSAGA::pick.from.ascii.grids(data = as.data.frame(random_points),
                                     X.name = "x",
                                     Y.name = "y",
                                     file = file.path(tempdir(), raster_names),
                                     varname = raster_names)
dplyr::select(vals, -geometry) %>%
  head(., 3)
##   id spri      x       y     dem1    dem2 log_carea cslope    ndvi
## 1  1    4 797179 8932755 -0.01049 0.01268  -1.21800  21.18 -0.3603
## 2  2    4 796749 8932621 -0.01019 0.01163   0.04145  13.02 -0.3488
## 3  3    3 796816 8932739 -0.01008 0.01124  -0.48148  23.70 -0.3396
```

Modeling species richness and predictive mapping

To model species richness, which is a count variable, we naturally opt for a Poisson regression. Overall, spatial count regression models are popular across many fields including epidemiology (Fernandez et al., 2012), demography (Chien et al., 2016), criminology (Jones-Webb and Wall, 2008), remote sensing (Comber et al., 2016) and ecology (Moreno-Fernández et al., 2015). Since we observed a unimodal nonlinear relationship between species richness and elevation, elevation enters the model as a second-order polynomial function.

```
fit <- glm(formula = spri ~ dem1 + dem2 + cslope + ndvi + log_carea,
           data = vals,
           family = "poisson")
```

To spatially predict species richness (Figure 5), we apply the estimated beta coefficients to the input rasters. **raster**'s `predict` function does this by accepting the fitted model and the predictor rasters as input. Finally, the `crop`-function ensures that the predictions are restricted to the study area.

```
library("sf")
library("raster")
```

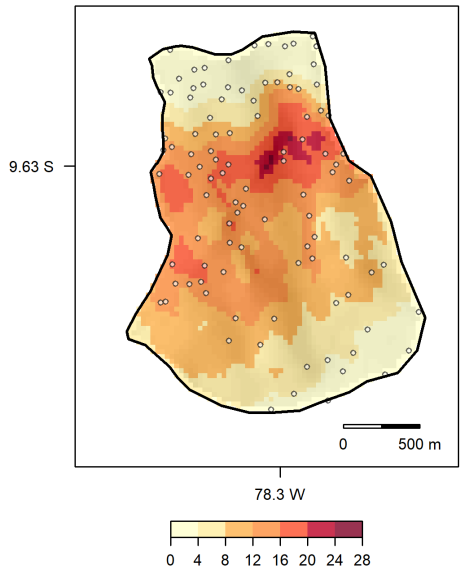


Figure 5: Prediction map of species richness. The points represent the visited plots.

```
raster_names <- c("dem1.asc", "dem2.asc", "log_carea.asc", "cslope.asc",
                  "ndvi.asc")
s <- stack(x = file.path(tempdir(), raster_names))
pred <- predict(object = s,
                model = fit,
                fun = predict,
                type = "response")
pred <- crop(x = pred,
             y = as(random_points, "Spatial"))
# plot the output (shown in Figure 5)
plot(pred)
plot(st_geometry(random_points), add = TRUE)
```

Note that an alternative to `raster::predict()` is **RSAGA**'s `multi.local.function()` in conjunction with `grid.predict()` (see Brenning, 2008).

The model reached a good goodness-of-fit (explained deviance divided by null deviance) of 0.78. The most important variable in predicting species richness was elevation and its squared term. In our interpretation, variation with elevation mainly relates to differences in water availability. Humidity, and thus species richness, is greatest just below the temperature inversion (ca. 750–850 m). For a more detailed interpretation of the model and its predictors, refer to Muenchow et al. (2013b).

Extending RQGIS through Python and PyQGIS

In this section we would like to show examplarily how one can easily extend **RQGIS** through Python and especially PyQGIS. As explained in sections Basic concepts and Usage, **RQGIS** uses the **reticulate** package to establish a tunnel to the QGIS API. To find out which Python binary is in use we run the `py_config()` function of the **reticulate** package. When using Windows, please do so only after having run `open_app` before, since this sets all necessary paths to run the QGIS Python binary. Otherwise `py_config()` will be unable to find it, in which case it most likely would use another Python binary (e.g., the one shipped with Anaconda) if available. Additionally, `open_app()` imports various necessary Python libraries (among others `osgeo`, `processing` and `qgis`), and attaches our Python RQGIS class (see section Usage).

```
library("reticulate")
py_config()
## python:         C:/OSGeo4W64/bin/python.exe
## libpython:      C:/OSGeo4W64/bin/python27.dll
## pythonhome:     C:\OSGeo4W64\apps\Python27
```

```
## version:        2.7.5 (default, May 15 2013, 22:44:16) [MSC v.1500 64 bit (AMD64)]
## Architecture:   64bit
## numpy:          C:\OSGeo4W64\apps\Python27\lib\site-packages\numpy
## numpy_version:  1.12.1
##
## NOTE: Python version was forced by use_python function
```

We have defined the Python class RQGIS in `python_funs.py` which is part of 'inst/python'. Aside from `set_env()` all functions found in 'R/processing' make extensive use of the Python RQGIS methods. Most of the Python methods have the same names as their counterparts in R. We can access them with `py_run_string()` of the **reticulate**-package. This sends a call to Python, and converts the Python into R output if desired.

```
py_run_string("methods = dir(RQGIS)")$methods
## [1] "__doc__"           "__init__"          "__module__"
## [4] "check_args"        "get_args_man"      "get_options"
## [7] "open_help"         "qgis_session_info"
```

Of course, we can use the Python RQGIS methods directly via **reticulate** which is exactly what the **RQGIS**-package is doing. For example, to find out what the options are for a QGIS geoalgorithm named `qgis:randompointsinsidepolygonsvariable`, we can run:

```
py_cmd <- "opts = RQGIS.get_options('qgis:randompointsinsidepolygonsvariable')"
py_run_string(py_cmd)$opts
## $STRATEGY
## [1] "Points count"   "Points density"
```

Or we can use PyQGIS-functionality directly. For instance, assuming we would like to find out the function parameters of `qgis:randompointsinsidepolygonsvariable`, we can use `alghelp()` from the QGIS Python processing framework (Graser and Olaya, 2015, and see also https://docs.qgis.org/2.8/en/docs/user_manual/processing/console.html). Here, we only show the first 40 characters of the output.

```
py_cmd <- "processing.alghelp('qgis:randompointsinsidepolygonsvariable')"
py_capture_output(py_run_string(py_cmd)) %>%
  substring(., 1, 40)
## [1] "ALGORITHM: Random points inside polygons"
```

Users can easily extend the RQGIS class with additional methods, or they could write their own classes and methods. Also if there is a need to write Python one-liners or make use of some Python functionality, this can easily be done using **RQGIS** in conjunction with **reticulate**. Here, we will present one last example. Frequently, users have asked us if it was possible to also use the QGIS map canvas from within R. Therefore, we provide a proof-of-concept how this could be achieved (see also http://docs.qgis.org/testing/en/docs/pyqgis_developer_cookbook/canvas.html). First of all, we save `random_points` to a temporary output location.

```
data("random_points", package = "RQGIS")
file <- normalizePath(file.path(tempdir(), "points.shp"), winslash = "/",
                      mustWork = FALSE)
sf::st_write(random_points, dsn = file)
```

Before running the subsequent code, make sure that you have already attached the packages **RQGIS** and **reticulate**. Additionally, you need to run `open_app()` first. Then we can create the map canvas, add the previously saved shapefile to it, set the extent to the extent of our shapefile, set the map canvas layer, and finally open a standalone map window (Figure 6). If this does not open a standalone window you might have to run `py_run_string('app.exec_()')` to initialize a Qt event loop which in turn renders the points in a standalone window (pers. comm. Barry Rowlingson). We emphasize that this is only a proof-of-concept, and a rather unstable solution (see section Current and future developments).

```
# create the map canvas
py_run_string("canvas = QgsMapCanvas()")
# import point shapefile
py_run_string(sprintf("layer = QgsVectorLayer('%s', 'points', 'ogr')", file))
# add imported point layer to the map canvas
py_run_string("QgsMapLayerRegistry.instance().addMapLayer(layer)")
```

Figure 6: A very simple example how to access the QGIS map canvas from within R.

```
# set the extent of the map canvas to the extent of the imported shapefile
py_run_string("canvas.setExtent(layer.extent())")
# set the map canvas layer
py_run_string("canvas.setLayerSet([QgsMapCanvasLayer(layer)])")
# open a standalone window
py_run_string("canvas.show()")
# if a standalone window has not already opened, run the next line
# py_run_string("app.exec_()")
```

Discussion

R/GIS-integration: Combining the best of two worlds

In our use case (see section Ecological example: combining geocomputing and statistics) we mainly used QGIS for raster preprocessing and the generation of terrain attributes. Naturally, there are many more geospatial analysis problems that can be solved using the thousands of geoalgorithms that are accessible through **RQGIS** and other R–GIS packages. For instance, SAGA provides more than 600 (Conrad et al., 2015) and GRASS more than 500 functions (`http://grass.osgeo.org/grass72/manuals/`).

For example, apart from relatively simple DEM derivatives (slope angle and orientation), a GIS can also calculate more complex, process-oriented terrain attributes such as the relative slope position or the topographic wetness index (Conrad et al., 2015). Additionally, most GIS can easily extract stream networks (Hengl et al., 2010) and surface roughness (Grohmann, 2004). Somewhat related are geospatial calculations concerned with terrain classification and landform identification (Brenning, 2012a; Rocchini et al., 2013). Physically-based models (e.g., SHALSTAB or Factor of Safety, both available in SAGA GIS) may provide additional insights (Goetz et al., 2011). Of course, GIS also provide an extensive suite of vector processing tools as required, for example, in geomarketing.

As pointed out in the beginning R has its limitations regarding GIS capabilities, but when it comes to statistical analyses, R is the uncontested champion in its field. For instance, instead of using a polynomial function in our use case (see section Modeling species richness and predictive mapping), we could have used a generalized additive model (GAM) with a logarithmic link function to allow for nonlinear relationships between various predictors and species richness (Figure 4). A GAM is the nonlinear extension of a generalized linear model (GLM), and uses smoothing functions to deal with nonlinearity (Hastie, 2017). In ecology, coupling ordination techniques and GAMs is a fruitful approach to spatially predict ecological communities (Muenchow et al., 2013a).

Equally, machine learning algorithms (support vector machines, random forests, etc.) are readily available in R, although these methods may tend to overfit the training data (Brenning, 2005). Overfitting in turn limits a model's ability to spatially predict the response variable. Here, spatial cross-validation through the **sperrorest** package (Brenning et al., 2012) provides an opportunity to assess spatial predictive capabilities.

Frequently, the residuals of spatial models show some form of spatial autocorrelation. This violates the assumption of independent model residuals made by most statistical models (Dormann et al., 2007; Zuur et al., 2009). Violating this assumption can lead to untrustworthy p values, biased coefficients and subsequently poor predictions (Zuur et al., 2009). Fortunately, packages such as **nlme** (Pinheiro et al., 2017) and **mgcv** (Wood, 2017) let the user incorporate various correlation structures into a model. This is most helpful in the presence of temporal and spatial autocorrelation (e.g, Iturritxa et al.,

2015). Sometimes mixed-effect models may also account for autocorrelation since random intercepts allow for correlation within a group (e.g., Peters et al., 2014). Another way of dealing with spatial autocorrelation is e.g., to include auto-regressive correlation structures in a GLM or GAM within a Bayesian modeling approach (Zuur and Ieno, 2017).

To summarize, R offers an incredibly vast suite of advanced statistical and data science methods. On the other hand, QGIS and other GIS software offer a rich suite of geoalgorithms and geocomputational power. Interfacing R with GIS simply combines the best of two worlds for automated statistical geocomputing.

RSAGA and R/GRASS-integration

Compared to the two separate packages **rgrass7** and **RSAGA**, **RQGIS** has the advantage of providing a unified interface to both GRASS and SAGA GIS toolboxes. Moreover, QGIS facilitates the usage of third-party geoalgorithms by automatically converting vector (e.g., shapefiles) and raster formats (e.g., ASCII grid files) into the particular format supported by the third-party module. For instance, SAGA has its own grid format (sgrd-files) and GRASS uses its own database format. Running SAGA or GRASS functions, (R)QGIS automatically converts the input data using `io_gdal()` in the case of SAGA and `v.in.ogr()` or `r.in.gdal()` in the case of GRASS. Though this is extremely user-friendly especially when providing interfaces to various third-party providers, it comes at the prize of increased computing time due to the necessity of multiple format conversions during one geoalgorithm call. Equally user-friendly it the automatic setup of the GRASS environment (projection, region and mapset) through (R)QGIS, if necessary. This certainly facilitates access to GRASS, especially for less experienced GIS users. Finally, **RQGIS** is overall quite user-friendly due to its convenience functions `open_help()` and `get_args_man()` and through its support of R named arguments for geoalgorithm parameters (see sections Basic concepts and Usage).

However, (R)QGIS only integrates a subset of the modules available in SAGA and GRASS GIS. While this fraction is likely to grow in the near future, a full integration of all modules is improbable as it would duplicate functionality (though this of course already has happened) and interface functions that are unnecessary within the QGIS environment, such as the GRASS database functions. If the user's intention is to use GRASS's database management system (DBMS), the direct R–GRASS integration via the **spgrass6** (Bivand et al., 2013) and **rgrass7** packages (Bivand and Neteler, 2000) would be the appropriate path. **RQGIS** does not provide access to this DBMS since the GRASS plugin of the QGIS processing toolbox only allows restricted access to GRASS's DBMS functionality. The use of **rgrass7** also allows the user to operate within a single GRASS session instead of calling a new one for each GRASS command as implicitly done by QGIS.

In the case of SAGA GIS, **RSAGA** has additional benefits. First of all, **RSAGA** provides numerous user-friendly wrapper functions with arguments (and meaningful default values) documented in the R help pages. **RSAGA** also strives to provide unified access to a range of SAGA versions while using, if possible, persistent function and argument names as well as default values. This allows for an easier migration between SAGA versions. At the moment **RSAGA** supports versions 2.0.4–2.2.3, but support for SAGA versions until the current version 6.1 has already been developed, and is currently being tested. By contrast, QGIS 2.14 supports SAGA 2.1.2–2.3.1. With the release of QGIS 2.18.10, this support was limited to the long term SAGA release 2.3.x. Extremely useful are furthermore **RSAGA**'s special geocomputing functions that allow, for example, the application of any user-defined R function to a stack of grids, either locally or within moving windows (functions `multi.focal.function()` and `multi.local.function()`). In conjunction with `grid.predict()`, predict methods of models fitted in R can therefore also be applied to stacks of raster files, as shown in Brenning (2008).

In the end, it will be up to the user to decide whether to use **RQGIS**, **RSAGA**, **rgrass7** or some combination of these, depending on the user's preferences, expertise and tasks at hand. In any case, we would recommend **RQGIS** if a user

- requires mainly the more commonly used SAGA and GRASS functions,
- does not want to be bothered with setting up the GRASS environment,
- does not plan to use GRASS geodatabase capabilities,
- does not want to worry about spatial data format conversions,
- would like to use spatial objects residing in R as input-arguments, and load GIS output automatically into R,
- cherishes the flexibility of seamlessly integrating QGIS, GRASS, SAGA and other third-party software (GDAL/OGR, Orfeo Toolbox, LAStools, TauDEM) within a single geoprocessing workflow in R.

Accessing ArcGIS through RPyGeo

The integration of GIS functionality into R is not limited to the mentioned open-source GIS software, but also includes the global leader in commercial GIS software (Longley et al., 2011), ArcGIS, through the **RPyGeo** package (Brenning, 2012b). This package piggybacks on ArcGIS's own Python interface to ArcGIS functions, or 'tools'. **RPyGeo** generates Python code that calls ArcGIS. Some of the challenges currently include

- latency times related to launching Python and ArcGIS before each GIS call,
- passing data and files between R and ArcGIS,
- error tracking based on Python and/or ArcGIS error messages,
- interpretation of ArcGIS help pages from the perspective of a geoprocessing interface function in R.

While some of these limitations may be overcome in future releases of **RPyGeo** and ArcGIS, the growing interest in integrating R and ArcGIS is also evident from recent efforts to provide access to R from within ArcGIS (`https://r-arcgis.github.io/`). This so-called R–Bridge allows users to

- retrieve data from ArcGIS geodatabases into R as `"sp"` objects, and export R data back into ArcGIS geodatabases,
- run R code within ArcGIS as a user-defined 'tool'. This is similar to QGIS's ability to integrate R scripts as user-defined GIS modules in the Processing toolbox.

While this connectivity, in principle, goes both ways, the integration of ArcGIS into R through the R–Bridge is currently limited to data import/export, which is complementary to **RPyGeo**'s capability of executing ArcGIS modules from within R.

Current and future developments

There are several interesting developing directions in the R-spatial world and for **RQGIS**. For example, we have been asked multiple times to include the QGIS mapping widget within R for fast interactive visualization and styling of multiple layers. In section Extending **RQGIS** through Python and PyQGIS we have shown that this is theoretically possible. However, mixing R and Qt events seems to cause frequently trouble (also pers. comm. Barry Rowlingson). For instance, when running `QgsMapCanvas()` twice or enlarging the standalone window manually (Figure 6), one runs a high risk of crashing the current R session. Therefore, Kevin Stadler, Barry Rowlingson and Julia Wagemann have opted for a different approach realized within a Google Summer of Code Project. In this project they wrote the QGIS Plugin Network API which enables the user to access the QGIS API via a HTTP interface from another language capable of making HTTP calls (such as R). Along with the sibling R package **qgisremote** (`https://qgisapi.gitlab.io/qgisremote/index.html`) this allows R users to seamlessly exchange spatial data between R and QGIS. For example, one can add vector and raster layers from within R to the QGIS map canvas, interactively edit them (such as adding new points or changing polygon vertices), and load the results back into R. Similarly, the packages **mapview** (build on top of **leaflet**, Cheng et al., 2017) and **mapedit** lets the user interactively visualize and edit spatial objects on leaflet maps. The advantage of **qgisremote** over these two packages is that the QGIS map canvas probably is better able to handle larger quantities of spatial data compared to leaflet, and that it provides the user with the full power of a Desktop GIS for manually editing spatial data (trace, snap, etc.). Nevertheless, **mapview** is a great tool for interactive visualization, and **mapedit** is a good alternative for small and quick modifications of spatial vector data within R. Coming back to the user request to include the QGIS mapping widget into **RQGIS**, we can state that **qgisremote** already filled this gap perfectly. In summary, **RQGIS** and **qgisremote** complement one another. The first gives an R user direct access to the QGIS processing engine, and the latter hands an R user the power of a Desktop GIS graphical user interface.

Adding new functionalities to **RQGIS** will generally include Python programming. Since QGIS migrates from Python 2 to Python 3 by the end of 2017, it is probably best to postpone major **RQGIS** extensions until after the migration. To guarantee a smooth transition, and to offer the possibility to work either with QGIS 2 or QGIS 3, we have designed **RQGIS** in such a way that we ideally would merely have to add a Python 3 script to 'inst/python' to make **RQGIS** also work with QGIS 3. In the case of a bumpier transition than anticipated, **RQGIS** users may rest assured that **RQGIS** will work in any case with the QGIS long term release 2.18 which will be further developed and regularly updated (see `https://www.qgis.org/en/site/getinvolved/development/roadmap.html#release-schedule`).

Another envisaged **RQGIS** update includes the support for `"stars"` classes (see `https://github.com/r-spatial/stars`). **stars** aims to mainly extend the **raster** package.

Conclusions

Combining R and GIS software creates a powerful environment for advanced statistical geocomputing. **RQGIS** makes this also possible with QGIS—one of the most-widely used open-source GIS, which is therefore probably also very appealing to R users. Conveniently, **RQGIS** offers a unified interface to various desktop GIS (SAGA, GRASS, etc.) that are integrated into QGIS. The use of GIS tools is facilitated through auxiliary functions for the automatic retrieval of function arguments and their default values, the support of R named arguments in `run_qgis()`, the seamless exchange of spatial data types, and the quick access of the online help for any QGIS geoalgorithm.

Acknowledgments

We greatly appreciate the work of the QGIS development and the R core team. We are also greatly indebted to Dr. Rainer Krug (University of Zurich) who not only reviewed our manuscript two times but also improved it with his valuable suggestions. Of course, we thank Dr. Roger Bivand, our editor, for handling our manuscript. We are also grateful to Dr. Tomislav Hengl (SRIC – World Soil Information, Wageningen), and an anonymous reviewer for valuable feedback on a previous manuscript version. Finally, we would like to thank Barry Rowlingson (Lancaster University) for fruitful discussions on Python, QGIS and R.

Bibliography

J. Allaire, Y. Tang, and M. Geelnard. *reticulate: R Interface to Python*, 2017. URL https://CRAN.R-project.org/package=reticulate. R package version 1.3.1. [p410]

T. Appelhans and K. Russell. *mapedit: Interactive Editing of Spatial Data in R*, 2017. URL https://CRAN.R-project.org/package=mapedit. R package version 0.3.2. [p409]

T. Appelhans, F. Detsch, C. Reudenbach, and S. Woellauer. *mapview: Interactive Viewing of Spatial Data in R*, 2017. URL https://CRAN.R-project.org/package=mapview. R package version 2.2.0. [p409]

M. Basille and D. Bucklin. *rpostgis: R Interface to a 'PostGIS' Database*, 2017. URL https://CRAN.R-project.org/package=rpostgis. R package version 1.3.0. [p]

C. J. G. Bellosta. *rPython: Package Allowing R to Call Python*, 2015. URL https://CRAN.R-project.org/package=rPython. R package version 0.0-6. [p410]

R. Bivand. Using the R–GRASS interface. *OSGeo Journal*, 1:36–38, 2007. URL http://fdo.osgeo.org/files/journal/final_pdfs/OSGeo_vol1_GRASS-R.pdf. [p410]

R. Bivand. *rgrass7: Interface Between GRASS 7 Geographical Information System and R*, 2017. URL https://CRAN.R-project.org/package=rgrass7. R package version 0.1-10. [p409, 410]

R. Bivand and N. Lewin-Koh. *maptools: Tools for Reading and Handling Spatial Objects*, 2017. URL https://CRAN.R-project.org/package=maptools. R package version 0.9-2. [p409]

R. Bivand and M. Neteler. Open source geocomputation: Using the R data analysis language integrated with GRASS GIS and PostgreSQL data base systems. In *GeoComputation*, 2000. URL http://www.geocomputation.org/2000/GC009/Gc009.htm. [p422]

R. Bivand and C. Rundel. *rgeos: Interface to Geometry Engine - Open Source ('GEOS')*, 2017. URL https://CRAN.R-project.org/package=rgeos. R package version 0.3-26. [p410]

R. Bivand, E. Pebesma, and V. Gómez-Rubio. *Applied Spatial Data Analysis with R*, volume 10 of *Use R!* Springer, New York, 2nd edition, 2013. ISBN 9781461476177. URL https://doi.org/10.1007/978-1-4614-7618-4. [p409, 422]

R. Bivand, T. Keitt, and B. Rowlingson. *rgdal: Bindings for the 'Geospatial' Data Abstraction Library*, 2017. URL https://CRAN.R-project.org/package=rgdal. R package version 1.2-16. [p410]

A. Brenning. Spatial prediction models for landslide hazards: Review, comparison and evaluation. *Natural Hazards and Earth System Sciences*, 5(6):853–862, 2005. URL https://doi.org/10.5194/nhess-5-853-2005. [p421]

A. Brenning. Statistical geocomputing combining R and SAGA: The example of landslide susceptibility analysis with generalized additive models. *Hamburger Beiträge zur Physischen Geographie und Landschaftsökologie*, 19:23–32, 2008. [p410, 415, 419, 422]

A. Brenning. Spatial cross-validation and bootstrap for the assessment of prediction rules in remote sensing: The R package sperrorest. In *IEEE International Symposium on Geoscience and Remote Sensing IGARSS*, 2012a. URL https://doi.org/10.1109/IGARSS.2012.6352393. [p421]

A. Brenning. *RPyGeo: ArcGIS Geoprocessing in R via Python*, 2012b. URL https://CRAN.R-project.org/package=RPyGeo. R package version 0.9-3. [p410, 423]

A. Brenning, S. Koszinski, and M. Sommer. Geostatistical homogenization of soil conductivity across field boundaries. *Geoderma*, 143(3-4):254–260, 2008. URL https://doi.org/10.1016/j.geoderma.2007.11.007. [p410]

A. Brenning, S. Long, and P. Fieguth. Detecting rock glacier flow structures using gabor filters and IKONOS imagery. *Remote Sensing of Environment*, 125:227–237, 2012. URL https://doi.org/10.1016/j.rse.2012.07.005. [p421]

A. Brenning, M. Schwinn, A. P. Ruiz-Páez, and J. Muenchow. Landslide susceptibility near highways is increased by 1 order of magnitude in the Andes of Southern Ecuador, Loja Province. *Natural Hazards and Earth System Sciences*, 15(1):45–57, 2015. URL https://doi.org/10.5194/nhess-15-45-2015. [p410]

P. E. Brown. Maps, coordinate reference systems and visualising geographic data with mapmisc. *The R Journal*, 8(1):64–91, 2016. URL https://journal.r-project.org/archive/2016/RJ-2016-005/RJ-2016-005.pdf. [p409]

J. M. Chambers. *Extending R*. The R Series. Chapman & Hall CRC, Boca Raton, Florida, 2016. ISBN 9781498775717. [p410]

J. Cheng, B. Karambelkar, and Y. Xie. *leaflet: Create Interactive Web Maps with the JavaScript 'Leaflet' Library*, 2017. URL https://CRAN.R-project.org/package=leaflet. R package version 1.1.0. [p423]

L.-C. Chien, Y. Guo, and K. Zhang. Spatiotemporal analysis of heat and heat wave effects on elderly mortality in Texas, 2006-2011. *Science of the Total Environment*, 562:845–851, 2016. URL https://doi.org/10.1016/j.scitotenv.2016.04.042. [p418]

A. Comber, H. Balzter, B. Cole, P. Fisher, S. C. M. Johnson, and B. Ogutu. Methods to quantify regional differences in land cover change. *Remote Sensing*, 8(3), 2016. URL https://doi.org/10.3390/rs8030176. [p418]

O. Conrad, B. Bechtel, M. Bock, H. Dietrich, E. Fischer, L. Gerlitz, J. Wehberg, V. Wichmann, and J. Böhner. System for Automated Geoscientific Analyses (SAGA) v. 2.1.4. *Geoscientific Model Development*, 8(7):1991–2007, 2015. URL https://doi.org/10.5194/gmd-8-1991-2015. [p410, 421]

M. O. Dillon, M. Nakazawa, and S. G. Leiva. The lomas formations of coastal Peru: Composition and biogeographic history. In J. Haas and M. O. Dillon, editors, *El Niño in Peru: Biology and Culture over 10,000 Years*, volume 10 of *Botany*, pages 1–9. Field Museum of Natural History, Chigaco, 2003. [p415]

C. F. Dormann, J. M. McPherson, M. B. Araújo, R. Bivand, J. Bolliger, G. Carl, R. G. Davies, A. Hirzel, W. Jetz, W. D. Kissling, I. Kühn, R. Ohlemüller, P. R. Peres-Neto, B. Reineking, B. Schröder, F. M. Schurr, and R. Wilson. Methods to account for spatial autocorrelation in the analysis of species distributional data: A review. *Ecography*, 30(5):609–628, 2007. URL https://doi.org/10.1111/j.2007.0906-7590.05171.x. [p421]

ESRI. *ArcGIS, Version 10.5*, 2017. URL http://www.arcgis.com. [p410]

M. J. A. Eugster and T. Schlesinger. osmar: OpenStreetMap and R. *R Journal*, 5(1):53–63, 2013. URL https://journal.r-project.org/archive/2013-1/eugster-schlesinger.pdf. [p409]

M. A. L. Fernandez, M. Schomaker, P. R. Mason, J. F. Fesselet, Y. Baudot, A. Boulle, and P. Maes. Elevation and cholera: An epidemiological spatial analysis of the cholera epidemic in Harare, Zimbabwe, 2008-2009. *BMC Public Health*, 12, 2012. URL https://doi.org/10.1186/1471-2458-12-442. [p418]

J. Franklin, P. McCullough, and C. Gray. Terrain variables used for predictive mapping of vegetation communities in Southern California. In J. P. Wilson and J. C. Gallant, editors, *Terrain Analysis*, pages 331–353. John Wiley & Sons, New York and Chichester, 2000. ISBN 978-0-471-32188-0. [p415]

GDAL Development Team. *GDAL - Geospatial Data Abstraction Library, version 2.2.2*. Open Source Geospatial Foundation, 2017. URL http://www.gdal.org. [p410]

GEOS Development Team. *GEOS - Geometry Engine Open Source, version 3.6.2*. Open Source Geospatial Foundation, 2017. URL https://trac.osgeo.org/geos. [p410]

J. N. Goetz, R. H. Guthrie, and A. Brenning. Integrating physical and empirical landslide susceptibility models using generalized additive models. *Geomorphology*, 129(3-4):376–386, 2011. URL https://doi.org/10.1016/j.geomorph.2011.03.001. [p421]

A. Graser and V. Olaya. Processing: A Python framework for the seamless integration of geoprocessing tools in QGIS. *ISPRS International Journal of Geo-Information*, 4(4):2219–2245, 2015. URL https://doi.org/10.3390/ijgi4042219. [p410, 420]

C. H. Grohmann. Morphometric analysis in Geographic Information Systems: Applications of free software GRASS and R. *Computers & Geosciences*, 30(9-10):1055–1067, 2004. URL https://doi.org/10.1016/j.cageo.2004.08.002. [p421]

V. Gómez-Rubio and A. López-Quílez. RArcInfo: Using GIS data with R. *Computers & Geosciences*, 31 (8):1000–1006, 2005. URL https://doi.org/10.1016/j.cageo.2005.02.009. [p409]

T. Hastie. *gam: Generalized Additive Models*, 2017. URL https://CRAN.R-project.org/package=gam. R package version 1.14-4. [p421]

T. Hengl, G. B. M. Heuvelink, and E. E. van Loon. On the uncertainty of stream networks derived from elevation data: The error propagation approach. *Hydrology and Earth System Sciences*, 14(7): 1153–1165, 2010. URL https://doi.org/10.5194/hess-14-1153-2010. [p410, 421]

T. Hengl, P. Roudier, D. Beaudette, and E. Pebesma. plotKML: Scientific visualization of spatio-temporal data. *Journal of Statistical Software*, 63(5), 2015. URL https://doi.org/10.18637/jss.v063.i05. [p409]

R. J. Hijmans. *raster: Geographic Data Analysis and Modeling*, 2017. URL https://CRAN.R-project.org/package=raster. R package version 2.6-7. [p409]

J. Inglada and E. Christophe. The Orfeo Toolbox remote sensing image processing software. In *Geoscience and Remote Sensing Symposium, 2009 IEEE International, IGARSS 2009*, volume 4, pages IV–733, 2009. [p410]

E. Iturritxa, N. Mesanza, and A. Brenning. Spatial analysis of the risk of major forest diseases in Monterey pine plantations. *Plant Pathology*, 64(4):880–889, 2015. URL https://doi.org/10.1111/ppa.12328. [p421]

R. Jones-Webb and M. Wall. Neighborhood racial/ethnic concentration, social disadvantage, and homicide risk: An ecological analysis of 10 us cities. *Journal of Urban Health*, 85(5):662–676, 2008. URL https://doi.org/10.1007/s11524-008-9302-y. [p418]

J. Kühn, A. Brenning, M. Wehrhan, S. Koszinski, and M. Sommer. Interpretation of electrical conductivity patterns by soil properties and geological maps for precision agriculture. *Precision Agriculture*, 10(6):490–507, 2009. URL https://doi.org/10.1007/s11119-008-9103-z. [p415]

P. Longley, M. Goodchild, D. J. Maguire, and D. W. Rhind. *Geographic Information Systems and Science*. John Wiley & Sons, Hoboken, N.J, 3rd ed. edition, 2011. ISBN 0470721448. [p409, 410, 423]

A. B. McBratney, I. O. Odeh, T. F. Bishop, M. S. Dunbar, and T. M. Shatar. An overview of pedometric techniques for use in soil survey. *Geoderma*, 97(3-4):293–327, 2000. URL https://doi.org/10.1016/S0016-7061(00)00043-4. [p415]

M. Mergili and H. Kerschner. Gridded precipitation mapping in mountainous terrain combining GRASS and R. *Norsk Geografisk Tidsskrift-Norwegian Journal of Geography*, 69(1):2–17, 2015. URL https://doi.org/10.1080/00291951.2014.992807. [p410]

M. Mergili, J. Krenn, and H.-J. Chu. r.randomwalk v1, a multi-functional conceptual tool for mass movement routing. *Geoscientific Model Development*, 8(12):4027–4043, 2015. URL https://doi.org/10.5194/gmd-8-4027-2015. [p410]

D. R. Montgomery and W. E. Dietrich. A physically based model for the topographic control on shallow landsliding. *Water Resources Research*, 30(4):1153–1171, 1994. URL https://doi.org/10.1029/93WR02979. [p415]

D. Moreno-Fernández, I. Cañellas, I. Barbeito, M. Sánchez-González, and A. Ledo. Alternative approaches to assessing the natural regeneration of Scots pine in a mediterranean forest. *Annals of Forest Science*, 72(5):569–583, 2015. URL https://doi.org/10.1007/s13595-015-0479-4. [p418]

J. Muenchow and P. Schratz. *RQGIS: Integrating R with QGIS*, 2017. URL https://CRAN.R-project.org/package=RQGIS. R package version 1.0.3. [p410]

J. Muenchow, A. Brenning, and M. Richter. Geomorphic process rates of landslides along a humidity gradient in the tropical Andes. *Geomorphology*, 139:271–284, 2012. URL https://doi.org/10.1016/j.geomorph.2011.10.029. [p410, 415]

J. Muenchow, A. Bräuning, E. Frank Rodríguez, and H. von Wehrden. Predictive mapping of species richness and plant species' distributions of a Peruvian fog oasis along an altitudinal gradient. *Biotropica*, 45(5):557–566, 2013a. URL https://doi.org/10.1111/btp.12049. [p415, 421]

J. Muenchow, H. Feilhauer, A. Bräuning, E. F. Rodríguez, F. Bayer, R. A. Rodríguez, and H. von Wehrden. Coupling ordination techniques and GAM to spatially predict vegetation assemblages along a climatic gradient in an ENSO-affected region of extremely high climate variability. *Journal of Vegetation Science*, 24(6):1154–1166, 2013b. URL https://doi.org/10.1111/jvs.12038. [p415, 419]

J. Muenchow, S. Hauenstein, A. Bräuning, R. Bäumler, E. F. Rodríguez, and H. von Wehrden. Soil texture and altitude, respectively, largely determine the floristic gradient of the most diverse fog oasis in the Peruvian desert. *Journal of Tropical Ecology*, 29:427–438, 2013c. URL https://doi.org/10.1017/S0266467413000436. [p415]

J. Muenchow, P. Dieker, J. Kluge, M. Kessler, and H. von Wehrden. A review of ecological gradient research in the Tropics: identifying research gaps, future directions, and conservation priorities. *Biodiversity and Conservation*, 2017. URL https://doi.org/10.1007/s10531-017-1465-y. [p415]

M. Neteler and H. Mitasova. *Open Source GIS: A GRASS GIS Approach*. Springer-Verlag, New York, 2008. [p410]

M. Padgham and A. Peutschnig. *dodgr: Distances on Directed Graphs*, 2017. URL https://CRAN.R-project.org/package=dodgr. R package version 0.0.3. [p409]

E. Pebesma. *sf: Simple Features for R*, 2017. URL https://CRAN.R-project.org/package=sf. R package version 0.5-5. [p409]

T. Peters, A. Bräuning, J. Muenchow, and M. Richter. An ecological paradox: High species diversity and low position of the upper forest line in the Andean depression. *Ecology and Evolution*, 4(11):2134–2145, 2014. URL https://doi.org/10.1002/ece3.1078. [p422]

R. J. Pike, I. S. Evans, and T. Hengl. Geomorphometry: A brief guide. In T. Hengl and H. I. Reuter, editors, *Geomorphometry*, Developments in soil science, pages 1–28. Elsevier, Amsterdam and Oxford, 2008. ISBN 0123743451. [p415]

J. Pinheiro, D. Bates, and R-core. *nlme: Linear and Nonlinear Mixed Effects Models*, 2017. URL https://CRAN.R-project.org/package=nlme. R package version 3.1-131. [p421]

L. Poggio and A. Gimona. Downscaling and correction of regional climate models outputs with a hybrid geostatistical approach. *Spatial Statistics*, 14:4–21, 2015. URL https://doi.org/10.1016/j.spasta.2015.04.006. [p410]

L. Y. Pomara, K. Ruokolainen, H. Tuomisto, and K. R. Young. Avian composition co–varies with floristic composition and soil nutrient concentration in Amazonian upland forests. *Biotropica*, 44(4):545–553, 2012. URL https://doi.org/10.1111/j.1744-7429.2011.00851.x. [p415]

QGIS Development Team. *QGIS Geographic Information System, Version 2.18.14-Las Palmas*. Open Source Geospatial Foundation, 2017. URL http://www.qgis.org/. [p410]

Rapidlasso. *LAStools, Version 171124*, 2017. URL https://rapidlasso.com/lastools/. [p410]

B. Ripley, D. Bates, and S. DebRoy. R data import/export, 2016. URL https://cran.r-project.org/doc/manuals/r-release/R-data.html. [p409]

D. Rocchini, L. Delucchi, G. Bacaro, P. Cavallini, H. Feilhauer, G. M. Foody, K. S. He, H. Nagendra, C. Porta, C. Ricotta, S. Schmidtlein, L. D. Spano, M. Wegmann, and M. Neteler. Calculating landscape diversity with information-theory based indices: A GRASS GIS solution. *Ecological Informatics*, 17:82–93, 2013. URL https://doi.org/10.1016/j.ecoinf.2012.04.002. [p421]

G. Sherman. *The PyQGIS programmer's guide: Extending QGIS 2.x with Python*. Locate Press, Chugiak Alas., 2014. ISBN 978-0-9894217-2-0. [p410]

D. Tarboton and I. Mohammed. *Terrain analysis using digital elevation models. TauDEM, Version 5*, 2017. URL http://hydrology.usu.edu/taudem/taudem5/index.html. [p410]

USGS. *U.S. Geological Survey (USGS) Earth Resources Observation and Science (EROS) Center*, 2017. URL http://earthexplorer.usgs.gov/. Last accessed 28 June 2017. [p]

K. A. Vanselow and C. Samimi. Predictive mapping of dwarf shrub vegetation in an arid high mountain ecosystem using remote sensing and random forests. *Remote Sensing*, 6(7):6709–6726, 2014. URL https://doi.org/10.3390/rs6076709. [p410]

S. Wood. *mgcv: Mixed GAM Computation Vehicle with Automatic Smoothness Estimation*, 2017. URL https://doi.org/10.1111/j.1467-9868.2010.00749.x. R package version 1.8-20. [p421]

H. Zandler, A. Brenning, and C. Samimi. Quantifying dwarf shrub biomass in an arid environment: Comparing empirical methods in a high dimensional setting. *Remote Sensing of Environment*, 158: 140–155, 2015. URL https://doi.org/10.1016/j.rse.2014.11.007. [p410]

A. F. Zuur and E. N. Ieno. *A Beginner's Guide to Regression Models with Spatial and Temporal Correlation*. Highland Statistics Ltd., Newburgh, UK, 2017. [p422]

A. F. Zuur, E. N. Ieno, N. Walker, A. A. Saveliev, and G. M. Smith. *Mixed Effects Models and Extensions in Ecology with R*. Statistics for Biology and Health. Springer-Verlag, New York, 2009. ISBN 978-0-387-87457-9. [p421]

Jannes Muenchow
http://orcid.org/0000-0001-7834-4717
Friedrich Schiller University Jena
Department of Geography
Löbdergraben 32
07743 Jena, Germany
jannes.muenchow@uni-jena.de

Patrick Schratz
https://orcid.org/0000-0003-0748-6624
Friedrich Schiller University Jena
Department of Geography
Löbdergraben 32
07743 Jena, Germany
patrick.schratz@uni-jena.de

Alexander Brenning
https://orcid.org/0000-0001-6640-679X
Friedrich Schiller University Jena
Department of Geography
Löbdergraben 32
07743 Jena, Germany
alexander.brenning@uni-jena.de

Partial Rank Data with the hyper2 Package: Likelihood Functions for Generalized Bradley-Terry Models

by Robin K. S. Hankin

Abstract Here I present the **hyper2** package for generalized Bradley-Terry models and give examples from two competitive situations: single scull rowing, and the competitive cooking game show MasterChef Australia. A number of natural statistical hypotheses may be tested straightforwardly using the software.

Introduction: the Bradley-Terry model

The Bradley-Terry model for datasets involving paired comparisons has wide uptake in the R community. However, existing functionality[1] is restricted to paired comparisons. The canonical problem is to consider n players who compete against one another; the basic inference problem is to estimate numbers $\mathbf{p} = (p_1, \ldots, p_n)$, $p_i \geqslant 0$, $\sum p_i = 1$ which correspond to player "strengths". Information about the p_i may be obtained from the results of paired comparisons between the players.

Applications are legion. The technique is widely used in a competitive sport context (Turner and Firth, 2012), in which matches are held between two opposing individuals or teams. It can also be applied to consumer choice experiments in which subjects are asked to choose a favourite from among two choices (Hatzinger and Dittrich, 2012), in which case the p_i are known as "worth parameters".

If player i competes against player j, and wins with probability P_{ij} then the likelihood function for $p_1, \ldots p_n$ corresponding to a win for i is $\frac{p_i}{p_i+p_j}$. As Turner and Firth (2012) point out, this may be expressed as

$$\text{logit}\left(P_{ij}\right) = \log p_i - \log p_j$$

and this fact may be used to estimate $\mathbf{p}$ using generalized linear models. However, consider the case where three competitors, i, j, and k compete. The probability that i wins is then $\frac{p_i}{p_i+p_j+p_k}$ (Luce, 1959); but there is no simple way to translate this likelihood function into a GLM. However, working directly with the likelihood function for $\mathbf{p}$ has several advantages which are illustrated below. The resulting likelihood functions may readily be generalized to accommodate more general order statistics, as in a race. In addition, likelihood functions may be specified for partial order statistics; also, observations in which a *loser* is identified may be given a likelihood function using natural R idiom in the package.

Further generalizations

Observing the winner w from a preselected set of competitors $\mathcal{C}$ has a likelihood function of $p_w / \sum_{i\in\mathcal{C}} p_i$. But consider a more general situation in which two disjoint teams $\mathcal{A}$ and $\mathcal{B}$ compete; this would have likelihood $\sum_{i\in\mathcal{A}} p_i / \sum_{i\in\mathcal{A}\cup\mathcal{B}} p_i$. Such datasets motivate consideration of likelihood functions $\mathcal{L}(\cdot)$ with

$$\mathcal{L}(\mathbf{p}) = \prod_{s\in\mathcal{O}} \left(\sum_{i\in s} p_i\right)^{n_s} \tag{1}$$

where $\mathcal{O}$ is a set of observations and s a subset of $[n] = \{1, 2, \ldots, n\}$; numbers n_s are integers which may be positive or negative. The approach adopted by the **hyperdirichlet** package is to store each of the 2^n possible subsets of $[n]$ together with an exponent:

$$\prod_{s\in 2^{[n]}} \left(\sum_{i\in s} p_i\right)^{n_s}. \tag{2}$$

but this was noted as being needlessly memory intensive and slow; it is limited, in practice, to $n \leqslant 9$.

Consider, for example, the following inference problem. Suppose we wish to make inferences about $p_1, \ldots, p_{20}$, the unknown parameters of a multinomial distribution with classes $c_1, \ldots, c_{20}$; we

[1] In theory, the deprecated **hyperdirichlet** package (Hankin, 2010) provides similar functionality but it is slow and inefficient. It is limited to a small number of players and cannot cope with the examples considered here, and is superceded by **hyper2**, which was originally called **hyperdirichlet2**.

demand that $p_i \geqslant 0$ and $\sum p_i = 1$. If our observation is a *single* trial with result $c_1 \cup c_2$ [that is, the observation was known to be either c_1 or c_2], then a likelihood function might be $\mathcal{L}_1(p_1, \ldots, p_{20}) = p_1 + p_2$. However, observe that this very simple example is not accessible to the **hyperdirichlet** package, which would have to store $2^{20} > 10^6$ powers, almost all of which are zero.

The **hyper2** package uses the obvious solution to this problem: work with equation 1, rather than equation 2 and store only nonzero powers. However, this requires one to keep track of which subsets of $[n]$ have nonzero powers. Suppose we wish to incorporate subsequent observations into our likelihood function p_1. We might observe two further independent trials, with results $c_1 \cup c_2$ and $c_1 \cup c_3$ respectively, having a likelihood $(p_1 + p_2)(p_1 + p_3)$. Then a likelihood function for all three trials might be $\mathcal{L}_2(p_1, \ldots, p_{20}) = (p_1 + p_2)^2 (p_1 + p_3)$.

One natural representation for the likelihood function $(p_1 + p_2)^2 (p_1 + p_3)$ would be as a function mapping subsets of $\{1, 2, \ldots, 20\}$ to the real numbers; in this case we would map the set $\{1, 2\}$ to (the power) 2, and map $\{1, 3\}$ to 1. However, note that updating our likelihood function from $\mathcal{L}_1$ to $\mathcal{L}_2$ increments the power of $p_1 + p_2$: some mechanism for identifying that the same sum appears in both marginal likelihood functions is needed.

The hyper2 package

One such mechanism is furnished by the `C++` Standard Template Library's "map" class (Musser et al., 2009) to store and retrieve elements. In STL terminology, a *map* is an associative container that stores values indexed by a key, which is used to sort and uniquely identify the values. In the package, the key is a (STL) `set` of strictly positive integers $\leqslant n$. The relevant `typedef` statements are:

```
typedef set<unsigned int> bracket;
typedef map<bracket, double> hyper2;
```

Thus a `bracket` object is a set of (unsigned) integers—here a sum of some p_i; and a `hyper2` object is a function that maps `bracket` objects to real numbers—here their power. The following `C++` pseudocode shows how the aforementioned likelihood function would be created:

```
const bracket b1.insert({1,2});  // b1 = (p1+p2)
const bracket b2.insert({1,3});  // b2 = (p1+p3)

hyper2 L;       // L2 is the likelihood function

// first observation:
L[b1]  = 1;        // L = (p1+p2)

//second observation:
L[b1] += 1;        // L = (p1+p2)^2   # updating of existing map element
L[b2] += 1;        // L = (p1+p2)^2*(p1+p3)^1
```

In the `STL`, a map object stores keys and associated values in whatever order the software considers to be most propitious. This allows faster access and modification times but the order in which the maps, and indeed the elements of a set, are stored is not defined. In the case of likelihood functions such as Equation 1, this is not an issue because both multiplication and addition are associative and commutative operations. One side-effect of using this system is that the order of the bracket-power key-value pairs is not preserved.

The package in use

Consider the `Chess` dataset of the **hyperdirchlet** package, in which matches between three chess players are tabulated (Table 1). The Bradley-Terry model (Bradley and Terry, 1952) is appropriate here (Caron and Doucet, 2012), and the **hyper2** package provides a likelihood function for the strengths of the players, p_1, p_2, p_3 with $p_1 + p_2 + p_3 = 1$. A likelihood function might be

$$\frac{p_1^{30} p_2^{36} p_3^{22}}{(p_1 + p_2)^{35} (p_2 + p_3)^{35} (p_1 + p_3)^{18}}.$$

Using the **hyper2** package, the R idiom to create this likelihood function would be a two-stage process. The first step would be to implement the numerator, that is the number of games won by each player:

Topalov	Anand	Karpov	total
22	13	-	35
-	23	12	35
8	-	10	18
30	36	22	88

Table 1: Results of 88 chess matches (dataset `chess` in the **aylmer** package (Hankin, 2008)) between three Grandmasters; entries show number of games won up to 2001 (draws are discarded). Topalov beats Anand 22-13; Anand beats Karpov 23-12; and Karpov beats Topalov 10-8.

```
R> library("hyper2")
R> chess <- hyper2(list(1, 2, 3), c(30, 36, 22))
R> chess

p1^30 * p2^36 * p3^22
```

Thus the `chess` object has the correct number of players (three), and has the numerator recorded correctly. To specify the denominator, which indicates the number of matches played by each pair of players, the package allows the following natural idiom:

```
R> chess[c(1, 2)] <- -35
R> chess[c(2, 3)] <- -35
R> chess[c(1, 3)] <- -18
R> chess

p1^30 * (p1 + p2)^-35 * (p1 + p3)^-18 * p2^36 * (p2 + p3)^-35 * p3^22
```

Note how the terms appear in an essentially random order, a side-effect of the efficient map class. It is sometimes desirable to name the elements explicitly:

```
R> pnames(chess) <- c("Topalov", "Anand", "Karpov")
R> chess

Topalov^30 * (Topalov + Anand)^-35 * (Topalov + Karpov)^-18 * Anand^36
* (Anand + Karpov)^-35 * Karpov^22
```

The package can calculate log-likelihoods:

```
R> loglik(chess, c(1/3, 1/3))

[1] -60.99695
```

[the second argument of function `loglik()` is a vector of length 2, third element of $\mathbf{p}$ being the "fillup" value (Aitchison, 1986)]; the gradient of the log-likelihood is given by function `gradient()`:

```
R> gradient(chess, c(1/3, 1/3))

[1] 24.0 16.5
```

Such functionality allows the rapid location of the maximum likelihood estimate for $\mathbf{p}$:

```
R> maxp(chess)

  Topalov     Anand    Karpov
0.4036108 0.3405168 0.2558723
```

Men's single sculling in the 2016 Summer Olympic Games

In this section, I will take results from the 2016 Summer Olympic Games and create a likelihood function for the finishing order in Men's single sculling. In Olympic sculling, up to six individual competitors race a small boat called a scull over a course of length 2 km; the object is to cross the finishing line first. Note that actual timings are irrelevant, given the model, as the sufficient statistic is the order in which competitors cross the finishing line. The 2016 Summer Olympics is a case in point: the gold and silver medallists finished less than 5 milliseconds apart, corresponding to a lead of ~ 2.5 cm. Following Luce (1959), the probability of competitor i winning in a field of $j = 1, \ldots, n$ is

$$\frac{p_i}{p_1 + \cdots + p_n}.$$

However, there is information in the whole of the finishing order, not just the first across the line. Once the winner has been identified, then the runner-up may plausibly be considered to be the winner among the remaining competitors; and so on down the finishing order. Without loss of generality, if the order of finishing were $1, 2, 3, 4, 5, 6$, then a suitable likelihood function would be, following Plackett (1975):

$$\frac{p_1}{p_1+p_2+p_3+p_4+p_5+p_6} \cdot \frac{p_2}{p_2+p_3+p_4+p_5+p_6} \cdot \frac{p_3}{p_3+p_4+p_5+p_6} \cdot \frac{p_4}{p_4+p_5+p_6} \cdot \frac{p_5}{p_5+p_6} \cdot \frac{p_6}{p_6} \tag{3}$$

The result of heat 1 may be represented as

$$\text{fournier} \succ \text{cabrera} \succ \text{bhokanal} \succ \text{saensuk} \succ \text{kelmelis} \succ \text{teilemb}$$

(a full list of the finishing order for all 25 events is given in the package as `rowing.txt`). The first step to incorporating the whole finishing order into a likelihood function is to define a `hyper2` object which stores the names of the participants:

```
R> data("rowing")
R> H <- hyper2(pnames = allrowers)
R> H
```

```
(banna + bhokanal + boudina + cabrera + campbell + dongyong + drysdale
+ esquivel + fournier + gambour + garcia + grant + hoff + kelmelis +
khafaji + kholmirzayev + martin + memo + molnar + monasterio + obreno +
peebles + rivarola + rosso + saensuk + shcharbachenia + synek +
szymczyk + taieb + teilemb + yakovlev + zambrano)^0
```

Observe that the resulting likelihood function is uniform, as no information has as yet been included. Incorporating the information from Heat 1 into a likelihood function corresponding to Equation 3 is straightforward using the `order_likelihood()` function:

```
R> heat1 <- c("fournier", "cabrera", "bhokanal", "saensuk",
+   "kelmelis", "teilemb")
R> H <- H + order_likelihood(char2num(heat1, allrowers))
R> H
```

```
bhokanal * (bhokanal + cabrera + fournier + kelmelis + saensuk +
teilemb)^-1 * (bhokanal + cabrera + kelmelis + saensuk + teilemb)^-1 *
(bhokanal + kelmelis + saensuk + teilemb)^-1 * cabrera * fournier *
kelmelis * (kelmelis + saensuk + teilemb)^-1 * (kelmelis + teilemb)^-1
* saensuk
```

(variable `heat1` shows the finishing order for Heat 1). Again observe that object `H` includes its terms in no apparent order. Although it would be possible to incorporate information from subsequent heats in a similar manner, the package includes a ready-made dataset, `sculls2016`:

```
R> head(sculls2016)
```

```
banna^4 * (banna + boudina + cabrera + molnar + obreno + rivarola)^-1 *
(banna + boudina + cabrera + molnar + rivarola)^-1 * (banna + boudina +
molnar + rivarola)^-1 * (banna + cabrera + campbell + grant + hoff)^-1
* (banna + cabrera + campbell + grant + hoff + szymczyk)^-1
```

Finding the maximum likelihood estimate for the parameter $p_{\text{banna}}, \ldots, p_{\text{zambrano}}$ is straightforward using the `maxp()` function, provided with the package (Figure 1). The optimization routine has access to derivatives which means that the estimate is found very quickly.

Figure 1 shows very clearly that the competitor with the highest strength is Drysdale, the gold medallist for this event. The bronze and silver medallists were Synek and Martin respectively, whose estimated strengths were second and third highest in the field.

MasterChef Australia

MasterChef Australia is a game show in which amateur cooks compete for a title (Wikipedia, 2017a). From a statistical perspective the contest is interesting because the typical show format is to identify the

```
R> dotchart(maxp(sculls2016))
```

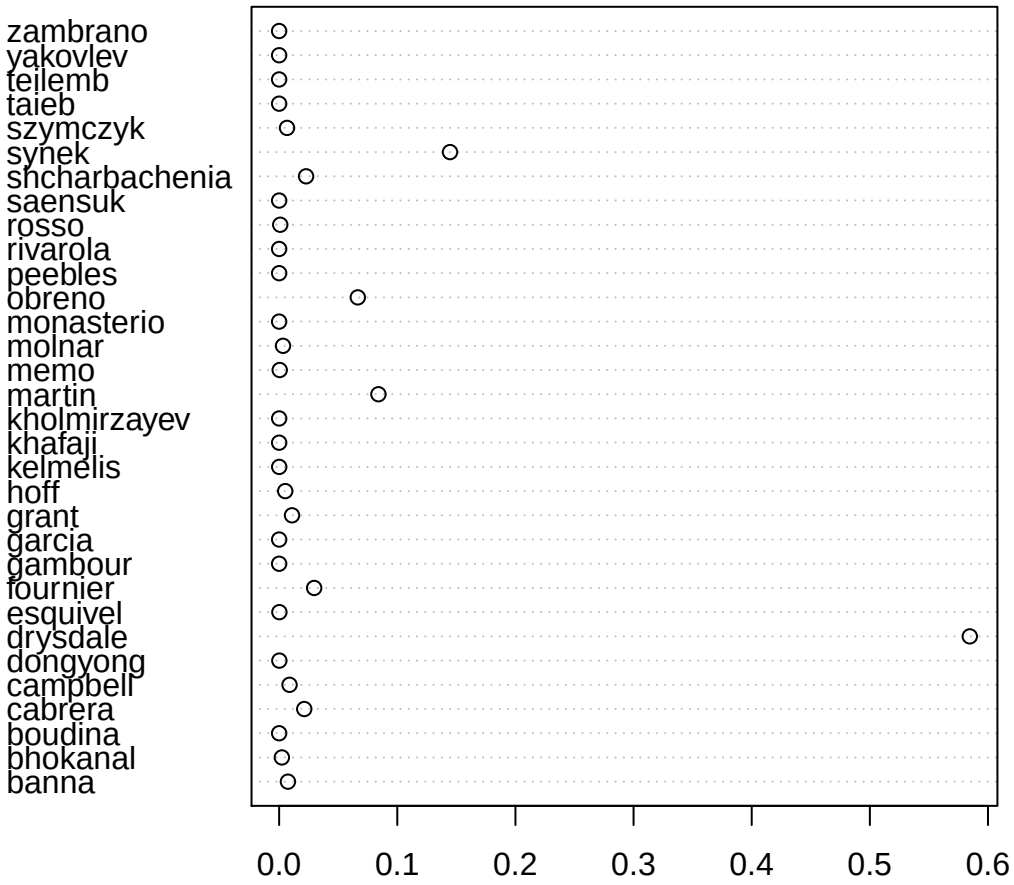


Figure 1: Maximum likelihood estimate for the strengths of the 32 competitors in the Men's singles sculls in the 2016 Summer Olympics.

weakest player, who is then eliminated from the competition. Here, results from MasterChef Australia Series 6 (Wikipedia, 2017b) will be analysed; an extended discussion of the data used is given in the package at `masterchef.Rd`.

We wish to make inferences about the contestents' generalized Bradley-Terry strengths $p_1, \ldots, p_n$, $\sum p_i = 1$. One informative event was a team challenge in which the contestants were split randomly into two teams, red and blue:

```
R> team_red <- c("Jamie", "Tracy", "Ben", "Amy", "Renae", "Georgia")
R> team_blue <- c("Brent", "Laura", "Emelia", "Colin", "Kira", "Tash")
```

We may represent the fact that the red team won as

$$\{\text{Jamie} + \text{Tracy} + \text{Ben} + \text{Amy} + \text{Renae} + \text{Georgia}\} \succ \{\text{Brent} + \text{Laura} + \text{Emelia} + \text{Colin} + \text{Kira} + \text{Tash}\}\,. \tag{4}$$

A plausible likelihood function can be generated using the standard assumption (Hankin, 2010) that the competitive strength of a team is the sum of the strengths of its members. The likelihood function for the observation given in Equation 4 would then be

$$\frac{p_{\text{Jamie}} + p_{\text{Tracy}} + p_{\text{Ben}} + p_{\text{Amy}} + p_{\text{Renae}} + p_{\text{Georgia}}}{p_{\text{Jamie}} + p_{\text{Tracy}} + p_{\text{Ben}} + p_{\text{Amy}} + p_{\text{Renae}} + p_{\text{Georgia}} + p_{\text{Brent}} + p_{\text{Laura}} + p_{\text{Emelia}} + p_{\text{Colin}} + p_{\text{Kira}} + p_{\text{Tash}}}\,. \tag{5}$$

To generate a likelihood function in R, we need to set up a `hyper2` object with appropriate contestants:

```
R> H <- hyper2(pnames = c(
+    "Amy", "Ben", "Brent", "Colin", "Emelia",
+    "Georgia", "Jamie", "Kira", "Laura", "Renae",
+    "Sarah", "Tash", "Tracy"))
R> H

(Amy + Ben + Brent + Colin + Emelia + Georgia + Jamie + Kira + Laura +
Renae + Sarah + Tash + Tracy)^0
```

Object `H` is a uniform likelihood function. The package R idiom for incorporating likelihood from Equation 5 is straightforward and natural:

```
R> H[team_red] <- +1
R> H[c(team_red, team_blue)] <- -1
R> H

(Amy + Ben + Brent + Colin + Emelia + Georgia + Jamie + Kira + Laura +
Renae + Tash + Tracy)^-1 * (Amy + Ben + Georgia + Jamie + Renae +
Tracy)
```

(Sarah did not take part). The above idiom makes it possible to define likelihoods for observations that have a peculiar probability structure, and I give two examples below.

One event involved eight competitors who were split randomly into four teams of two. The show format was specified in advance as follows: The teams were to be judged, and placed in order. The two top teams were to be declared safe, and the other two teams sent to an elimination trial from which an individual winner and loser were identified, the loser being obliged to leave the competition. The format for this event is also typical in MasterChef.

The observation was that Laura and Jamie's team won, followed by Emelia and Amy, then Brent and Tracy. Ben and Renae's team came last:

$$\{\text{Laura} + \text{Jamie}\} \succ \{\text{Emelia} + \text{Amy}\} \succ \{\text{Brent} + \text{Tracy}\} \succ \{\text{Ben} + \text{Renae}\}\,. \tag{6}$$

Again assuming that the team strength is the sum of its members' strengths, a likelihood function for this observation may be obtained by using the order statistic technique of Plackett (1975):

$$\frac{p_{\text{Laura}} + p_{\text{Jamie}}}{p_{\text{Laura}} + p_{\text{Jamie}} + p_{\text{Emelia}} + p_{\text{Amy}} + p_{\text{Brent}} + p_{\text{Tracy}} + p_{\text{Ben}} + p_{\text{Renae}}} \cdot \frac{p_{\text{Emelia}} + p_{\text{Amy}}}{p_{\text{Emelia}} + p_{\text{Amy}} + p_{\text{Brent}} + p_{\text{Tracy}} + p_{\text{Ben}} + p_{\text{Renae}}} \cdot \frac{p_{\text{Brent}} + p_{\text{Tracy}}}{p_{\text{Brent}} + p_{\text{Tracy}} + p_{\text{Ben}} + p_{\text{Renae}}} \tag{7}$$

and we would like to incorporate information from this observation into object `H`, which is a likelihood function for the two-team challenge discussed above. The corresponding package idiom is natural:

```
R> blue   <- c("Laura", "Jamie")
R> yellow <- c("Emelia", "Amy")
R> green  <- c("Brent", "Tracy")
R> red    <- c("Ben", "Renae")
```

(the teams were randomly assigned a colour). We may now generate a likelihood function for the observation that the order of teams was blue, yellow, green, red, as per Equation 7:

```
R> H[blue] <- 1
R> H[c(blue, yellow, green, red)] <- -1
R> H[yellow] <- 1
R> H[c(yellow, green, red)] <- -1
R> H[green] <- 1
R> H[c(green, red)] <- -1
R> H
```

```
(Amy + Ben + Brent + Colin + Emelia + Georgia + Jamie + Kira + Laura +
Renae + Tash + Tracy)^-1 * (Amy + Ben + Brent + Emelia + Jamie + Laura
+ Renae + Tracy)^-1 * (Amy + Ben + Brent + Emelia + Renae + Tracy)^-1 *
(Amy + Ben + Georgia + Jamie + Renae + Tracy) * (Amy + Emelia) * (Ben +
Brent + Renae + Tracy)^-1 * (Brent + Tracy) * (Jamie + Laura)
```

We may incorporate subsequent observations relating to the elimination trial among the four competitors comprising the losing two teams. The observation was that Laura won, and Renae came last, being eliminated. We might write

$$\{\text{Laura}\} \succ \{\text{Brent}, \text{Tracey}, \text{Ben}\} \succ \{\text{Renae}\}, \tag{8}$$

which indicates that Laura came first, then Brent/Tracey/Ben in some order, then Renae came last. For this observation a likelihood function, following Critchlow (1985), might be

$$\begin{aligned}
\mathcal{L}(p_1, p_2, p_3, p_4, p_5) &= \text{Prob}\left(p_1 \succ p_2 \succ p_3 \succ p_4 \succ p_5 \cup p_1 \succ p_2 \succ p_4 \succ p_3 \succ p_5 \cup \ldots\right) && (9)\\
&= \text{Prob}\left(\bigcup_{[abc]} p_1 \succ p_a \succ p_b \succ p_c \succ p_5\right) && (10)\\
&= \frac{p_1}{p_1+p_2+p_3+p_4+p_5} \cdot \frac{p_2}{p_2+p_3+p_4+p_5} \cdot \frac{p_3}{p_3+p_4+p_5} \cdot \frac{p_4}{p_4+p_5}\\
&\quad + \frac{p_1}{p_1+p_2+p_4+p_3+p_5} \cdot \frac{p_2}{p_2+p_4+p_3+p_5} \cdot \frac{p_4}{p_4+p_3+p_5} \cdot \frac{p_3}{p_3+p_5}\\
&\quad + \frac{p_1}{p_1+p_3+p_2+p_4+p_5} \cdot \frac{p_3}{p_3+p_2+p_4+p_5} \cdot \frac{p_2}{p_2+p_4+p_5} \cdot \frac{p_4}{p_4+p_5}\\
&\quad + \cdots
\end{aligned}$$

where Laura's strength is shown as p_1 etc for brevity. The R idiom is as follows:

```
R> L <- ggol(H,
+    winner     = "Laura",
+    btm4       = c("Brent", "Tracy", "Ben"),
+    eliminated = "Renae")
```

Arguments to `ggol()` are disjoint subsets of the players, the subsets themselves being passed in competition order from best to worst. Object `L` includes information from the team challenge (via first argument `H`) and the elimination results. It is a list of length $3! = 6$ objects of class `hyper2`, each of which gives a Luce likelihood function for a consistent total ordering of the competitors.

A final example (taken from MasterChef series 8, week 10) is given as a generalization of the Luce likelihood. The format was as follows. Eight contestents were split randomly into four teams of two, the top two teams being declared safe. Note that the likelihood interpretation differs from the previous team challenge, in which the observation was an unambiguous team ranking: here, there is only a partial ranking of the teams and one might expect this observation to be less informative. Without loss of generality, the result may be represented as

$$\{p_1 + p_2, p_3 + p_4\} \succ \{p_5 + p_6, p_7 + p_8\} \tag{11}$$

and a likelihood function on $p_1, \dots p_8$ for this observation might be

$$\begin{aligned}\mathcal{L}(p_1, \dots, p_8) = \text{Prob}\Big(&\{p_1+p_2\} \succ \{p_3+p_4\} \succ \{p_5+p_6\} \succ \{p_7+p_8\} \cup \\ &\{p_1+p_2\} \succ \{p_3+p_4\} \succ \{p_7+p_8\} \succ \{p_5+p_6\} \cup \\ &\{p_3+p_4\} \succ \{p_1+p_2\} \succ \{p_5+p_6\} \succ \{p_7+p_8\} \cup \\ &\{p_3+p_4\} \succ \{p_1+p_2\} \succ \{p_5+p_6\} \succ \{p_7+p_8\}\Big) \qquad (12) \\ = &\frac{p_1+p_2}{p_1+p_2+p_3+p_4+p_5+p_6+p_7+p_8} \cdot \frac{p_3+p_4}{p_3+p_4+p_5+p_6+p_7+p_8} \cdot \frac{p_5+p_6}{p_5+p_6+p_7+p_8} \\ &+ \cdots + \\ &\frac{p_3+p_4}{p_3+p_4+p_1+p_2+p_7+p_8+p_5+p_6} \cdot \frac{p_1+p_2}{p_3+p_4+p_7+p_8+p_5+p_6} \cdot \frac{p_7+p_8}{p_7+p_8+p_5+p_6}.\end{aligned}$$

Maximum likelihood estimation

The package provides an overall likelihood function for all informative judgements in the series on the final 13 players in object `masterchef_series6`. We may assess a number of related hypotheses using the package. The first step is to calculate the likelihood for the hypothesis that all players are of equal strength:

```
R> data("masterchef")
R> n <- 13
R> equal_strengths <- rep(1/n,n-1)
R> like_series(equal_strengths, masterchef_series6)

[1] -78.68654
```

The strengths of the 13 players may be estimated using standard maximum likelihood techniques. This requires constrained optimization in order to prevent the search from passing through inadmissible points in p-space:

```
R> UI <- rbind(diag(n-1), -1)
R> CI <- c(rep(0, n-1), -1)
R> constrOptim(
+    theta = equal_strengths,
+    f = function(p){-like_series(p, L)},
+    ui = UI, ci = CI,
+    grad = NULL)
```

In the above code, `UI` enforces $p_i \geqslant 0$ and `CI` enforces $p_1 + \cdots + p_{n-1} \leqslant 1$. The resulting maximum likelihood estimate, `pmax_masterchef6` in the package, is shown pictorially in Figure 2. The support at the precalculated evaluate is

```
R> like_series(indep(pmax_masterchef6), masterchef_series6)

[1] -66.19652
```

and this allows us to test the hypothesis of equal player strengths: by Wilks's theorem (Wilks, 1938) the quantity $-2\log\Lambda$ (where Λ is the likelihood ratio) has an asymptotic null distribution of χ^2_{12}. This corresponds to a p-value of

```
R> pchisq(2*(78.7-66.2), df = 12, lower.tail = FALSE)

[1] 0.01482287
```

showing that the observations do constitute evidence for differential player strengths. Figure 2 suggests that Laura, the runner-up, is actually a stronger competitor than the winner, Brent. We may assess this statistically by finding the maximum likelihood for $\mathbf{p}$, subject to the constraint that $p_{\text{Laura}} \leqslant p_{\text{Brent}}$:

```
R> UI <- rbind(UI, c(0, 0, 1, 0, 0, 0, 0, 0, -1, 0, 0, 0, 0))
R> CI <- c(CI, 0)
R> ans2 <-
+    constrOptim(
```

```
R> pmax_masterchef6

         Amy          Ben        Brent        Colin       Emelia      Georgia
1.086182e-01 7.457970e-02 1.343553e-01 2.819606e-02 1.169766e-01 6.850455e-09
       Jamie         Kira        Laura        Renae        Sarah         Tash
1.065412e-01 2.055794e-02 2.750621e-01 4.070643e-02 2.803893e-02 1.142094e-09
       Tracy
6.636755e-02

R> dotchart(pmax_masterchef6)
```

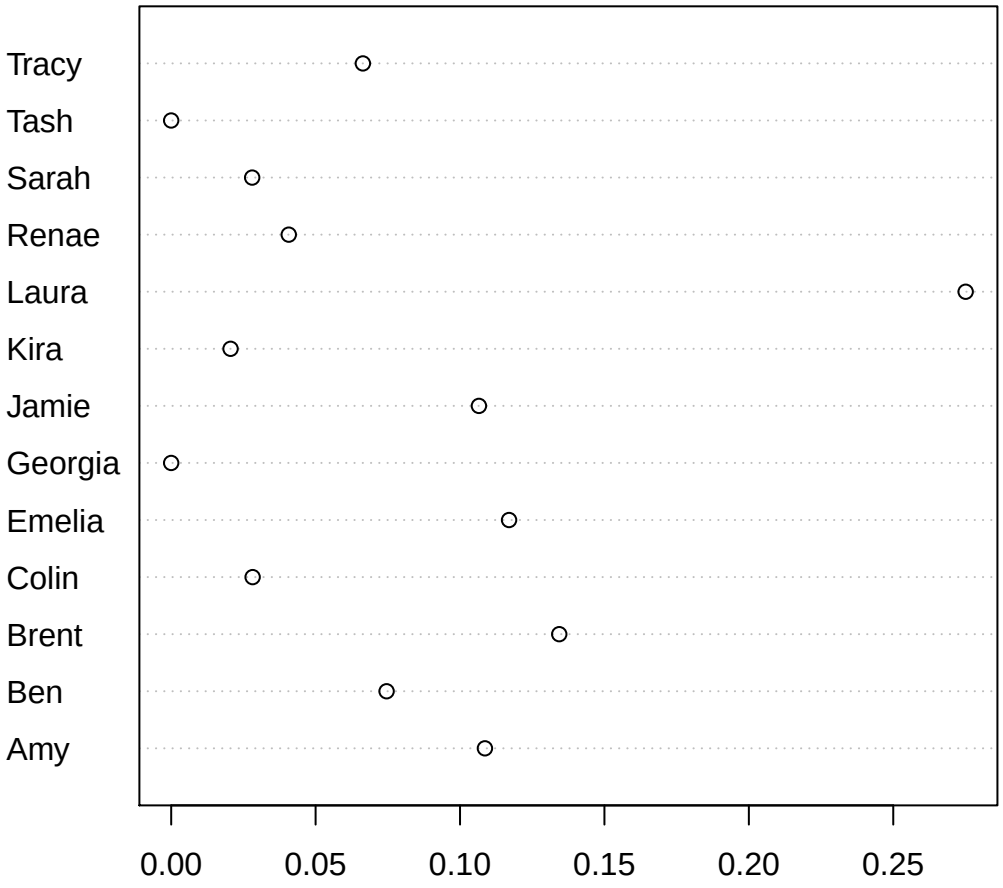


Figure 2: Maximum likelihood estimate for the strengths of the top 13 competitors in Series 6 of *MasterChef Australia*.

```
+      theta = equal_strengths,
+      f = function(p){-like_series(p, masterchef_series6)},
+      grad = NULL,
+      ui = UI, ci = CI)
```

(updated object `UI` represents the constraint that Brent's strength exceeds that of Laura). In the package, object `pmax_masterchef6_constrained` is included as the result of the above optimization, at which point the likelihood is

```
R> like_series(indep(pmax_masterchef6_constrained), masterchef_series6)

[1] -67.37642
```

The two estimates differ by about 1.18, less than the two-units-of-support criterion of Edwards (1992); alternatively, one may observe that the likelihood ratio is not in the tail region of its asymptotic distribution (χ^2_1) as the p-value is about 0.12. This shows that there is no strong evidence for Laura's competitive strength being higher than that of Brent. Similar techniques can be used to give a profile likelihood function; the resulting support interval for Laura's strength is $[0.145, 0.465]$, which does not include $\frac{1}{13} \simeq 0.077$, the mean player strength.

However, further work would be needed to make statistically robust inferences from these findings. Suppose, for example, that all competitors have equal competitive ability: then all the p_i are identical, and players are exchangeable. Under these circumstances, one could run a tournament and identify a winner. One might expect that the winning player would have the highest p_i as estimated by `pmax()`. It is not clear at this stage how to interpret likelihood functions for players conditional on their competition performance. Another issue would be the applicability of Wilks's theorem (Wilks, 1938) which states only that the *asymptotic* distribution of $-2\log\Lambda$ is chi-squared. Although the likelihood ratio statistic is inherently meaningful, its sampling distribution is not clear at this stage.

Conclusions

Several generalizations of Bradley-Terry strengths are appropriate to describe competitive situations in which order statistics are sufficient.

The `hyper2` package is introduced, providing a suite of functionality for generalizations of the partial rank analysis of Critchlow (1985). The software admits natural R idiom for translating commonly occurring observations into a likelihood function.

The package is used to calculate maximum likelihood estimates for generalized Bradley-Terry strengths in two competitive situations: Olympic rowing, and *MasterChef Australia*. The estimates for the competitors' strengths are plausible; and several meaningful statistical hypotheses are assessed quantitatively.

Bibliography

J. Aitchison. *The Statistical Analysis of Compositional Data*. The Blackburn Press, 1986. [p431]

R. A. Bradley and M. E. Terry. The rank analysis of incomplete block designs I. The method of paired comparisons. *Biometrika*, 39:324–345, 1952. [p430]

F. Caron and A. Doucet. Efficient Bayesian inference for generalized Bradley-Terry models. *Journal of Computational and Graphical Statistics*, 21(1):174–196, 2012. URL https://dx.doi.org/10.1080/10618600.2012.638220. [p430]

D. E. Critchlow. *Metric Methods for Analyzing Partially Ranked Data*. Springer-Verlag, New York, 1985. [p435, 438]

A. W. F. Edwards. *Likelihood (Expanded Edition)*. John Hopkins, 1992. [p438]

R. K. S. Hankin. Exact tests for two-way contingency tables with structural zeros. *Journal of Statistical Software*, 28(11):1–19, 2008. URL https://doi.org/10.18637/jss.v028.i11. [p431]

R. K. S. Hankin. A generalization of the Dirichlet distribution. *Journal of Statistical Software*, 33(11): 1–18, 2010. URL https//doi.org/10.18637/jss.v033.i11. [p429, 434]

R. Hatzinger and R. Dittrich. Prefmod: An R package for modeling preferences based on paired comparisons, rankings, or ratings. *Journal of Statistical Software*, 48:1–31, 2012. URL https://10.18637/jss.v048.i10. [p429]

R. Luce. *Individual Choice Behaviour: A Theoretical Analysis*. John Wiley & Sons, New York, 1959. [p429, 431, 435]

D. R. Musser, G. J. Derge, and A. Saini. *STL Tutorial and Reference Guide: C++ Programming with the Standard Template Library*. Addison-Wesley Professional, 3rd edition, 2009. ISBN 0321702123, 9780321702128. [p430]

R. L. Plackett. The analysis of permutations. *Applied Statistics*, 24:193–202, 1975. [p432, 434]

H. Turner and D. Firth. Bradley-terry models in R: The BradleyTerry2 package. *Journal of Statistical Software*, 48(9):1–21, 2012. URL https://doi.org/10.18637/jss.v048.i09. [p429]

Wikipedia. MasterChef Australia — Wikipedia, the free encyclopedia, 2017a. URL https://en.wikipedia.org/w/index.php?title=MasterChef_Australia&oldid=761024807. [Online; accessed 1-February-2017]. [p432]

Wikipedia. MasterChef Australia (series 6) — Wikipedia, the free encyclopedia, 2017b. URL https://en.wikipedia.org/w/index.php?title=MasterChef_Australia_(series_6)&oldid=762395535. [Online; accessed 1-February-2017]. [p434]

S. S. Wilks. The large-sample distribution of the likelihood ratio for testing composite hypotheses. *The Annals of Mathematical Statistics*, 9(1):60–62, 1938. URL http://www.jstor.org/stable/2957648. [p436, 438]

Robin K. S. Hankin
AUT University
Auckland
New Zealand
ORCiD: 0000-0002-6727-9347
hankin.robin@gmail.com

riskRegression: Predicting the Risk of an Event using Cox Regression Models

by Brice Ozenne, Anne Lyngholm Sørensen, Thomas Scheike, Christian Torp-Pedersen, Thomas Alexander Gerds

Abstract In the presence of competing risks a prediction of the time-dynamic absolute risk of an event can be based on cause-specific Cox regression models for the event and the competing risks (Benichou and Gail, 1990). We present computationally fast and memory optimized C++ functions with an R interface for predicting the covariate specific absolute risks, their confidence intervals, and their confidence bands based on right censored time to event data. We provide explicit formulas for our implementation of the estimator of the (stratified) baseline hazard function in the presence of tied event times. As a by-product we obtain fast access to the baseline hazards (compared to `survival::basehaz()`) and predictions of survival probabilities, their confidence intervals and confidence bands. Confidence intervals and confidence bands are based on point-wise asymptotic expansions of the corresponding statistical functionals. The software presented here is implemented in the **riskRegression** package.

Introduction

Predictions of hazards and risks based on a Cox regression analysis need to be fast and memory efficient, especially in large data, in simulation studies, and for cross-validation or bootstrap loops. The CRAN task view *Survival* lists many R packages implementing the Cox regression model and extensions thereof. Among the most popular routines are the function `coxph()` from the **survival** package (Therneau, 2017) and the function `cph()` from the **rms** package (Harrell Jr, 2017). We present a fast and memory efficient algorithm to extract baseline hazards and predicted risks with confidence intervals from an object obtained with either of these functions.

In the presence of competing risks one needs to combine at least two Cox regression models to predict the absolute risk of an event (cumulative incidence) conditional on covariates (Benichou and Gail, 1990). We present the `CSC()`-function of the R package **riskRegression** which fits the Cox regression models using either `coxph()` or `cph()`. We also present a concomitant `predict()` S3 method which computes the absolute risks of the event of interest for given combinations of covariate values and time points. Optionally, the `predict()` method computes asymptotic confidence intervals and confidence bands for the predicted absolute risks. We review the formula behind the estimators implemented and illustrate the R interface.

It is possible to obtain the predictions of absolute risks based on cause-specific Cox regression also with the **survival** package or with the **mstate** package (Putter et al., 2016). However, both require more work from the user. Finally, it should be noted that there are alternative regression methods for absolute risks in the presence of competing risks such as Fine-Gray regression (Fine and Gray, 1999) or direct binomial regression (Gerds et al., 2012; Scheike et al., 2008).

Data used for examples

For the sole purpose of illustration we use the 'Melanoma' data set which is included in the **riskRegression** package. It contains data from 205 malignant melanoma patients. Among the risk factors for cancer specific death were patient age and sex and the histological variables tumor thickness, invasion (levels 0,1,2), and epithelioid cells (no present vs. present). Within the limitation of the follow-up periods, it was observed that 57 patients had died from cancer ("status" equals 1) and 14 had died from other causes ("status" equals 2). The remaining patients were right censored ("status" equals 0).

```
library(riskRegression, verbose = FALSE, quietly = TRUE)
library(survival)
data(Melanoma)
str(Melanoma)
```

```
## 'data.frame': 205 obs. of  7 variables:
##  $ time     : int  10 30 35 99 185 204 210 232 232 279 ...
##  $ status   : int  3 3 2 3 1 1 1 3 1 1 ...
##  $ sex      : int  1 1 1 0 1 1 1 0 1 0 ...
##  $ age      : int  76 56 41 71 52 28 77 60 49 68 ...
##  $ year     : int  1972 1968 1977 1968 1965 1971 1972 1974 1968 1971 ...
```

```
##  $ thickness: num  6.76 0.65 1.34 2.9 12.08 ...
##  $ ulcer    : int  1 0 0 0 1 1 1 1 1 1 ...
```

Predicting absolute risks based on cause-specific Cox regression

We denote by T the time between a baseline date and the date of an event and by $D \in \{1, ..., K\}$ the cause of the event. We assume that $\{D = 1\}$ is the event of interest. Let $X = (X^1, \ldots, X^p)$ be a p-dimensional vector of baseline covariates with arbitrary distribution, and $Z = (Z^1, \ldots, Z^q)$ be the strata variables, i.e. a set of categorical baseline covariates with finitely many possible values. Without loss of generality and to ease the notation we set $q = 1$. We use $\{1, \ldots, L\}$ for the categories of Z.

We consider a setting in which the event time T is right censored at a random time C. We assume that C is conditionally independent of T given (X, Z) and fix a time τ such that almost surely $P(C > \tau | X, Z) > 0$. We denote $\tilde{T} = \min(T, C)$, $\tilde{D} = \Delta D$, and $\Delta = 1\{T \leq C\}$.

Cause-specific Cox regression

Given covariates (X, Z), let $S_0(t|x, z) = P(T > t|X = x, Z = z)$ denote the event-free survival function and $F_j(t|x, z) = P(T \leq t, D = j|X = x, Z = z)$ the cumulative incidence function for event j. The cause-specific hazard rates are defined as $\lambda_{j,z}(t|x) = \mathrm{d}F_j(t|x, z)/S_0(t|x, z)$ (Andersen et al., 1993). We also denote the cumulative hazard rates by $\Lambda_{j,z}(t|x) = \int_0^t \lambda_{j,z}(s|x)\mathrm{d}s$. The stratified Cox regression model (Cox, 1972) for cause j is given by

$$\lambda_{j,z}(t|x) = \lambda_{0j,z}(t) \exp(x\beta_j), \tag{1}$$

where $\beta_j = (\beta_j^1, \ldots, \beta_j^p)^\intercal$ is a p-dimensional vector of regression coefficients (the log-hazard ratios), and $\{\lambda_{0j,z}(t) : z = 1, \ldots, L\}$ a set of unspecified baseline hazard functions.

Predicting the absolute risk of an event

The cause-specific Cox regression models can be combined into a prediction of the absolute risk of an event of type 1 until time t conditional on the covariates x, z. For the case where $K = 2$ the absolute risk formula of Benichou and Gail (1990) is given by:

$$F_1(t|x, z) = \int_0^t S(s - |x, z)\lambda_{1,z}(s|x)\mathrm{d}s. \tag{2}$$

where $s-$ denotes the right sided limit, e.g. $\Lambda_{1,z}(s - |x) = lim_{v \to s, v<s}\Lambda_{1,z}(v|x)$. The absolute risk accumulates over time the product between the event-free survival and the hazard of experiencing the event of interest, both conditional to the baseline covariates and to the strata variable. The event free survival can be estimated from the cause-specific hazards using the product integral estimator:

$$S(t|x, z) = \prod_{s \leq t} (1 - d\Lambda_{1,z}(t|x) - d\Lambda_{2,z}(t|x))$$

or the exponential approximation:

$$\widehat{S}(t|x, z) = \exp\left[- \widehat{\Lambda}_{1,z}(t|x) - \widehat{\Lambda}_{2,z}(t|x)\right]. \tag{3}$$

which is asymptotically equivalent to the product-limit estimator if the distribution of the event times is continuous. Using the product integral estimator ensures that $S(t|x, z) + F_1(t|x, z) + F_2(t|x, z)$ equals exactly 1. This is a desirable property since the sum of the transition probabilities over all possible transitions should sum to one.

Formula (2) generalizes to situations with more than 2 competing risks, i.e., $K > 2$. However, in applications with many competing risks there will sometimes be few events of specific causes, and it may be hard to fit a Cox regression model for each cause separately. One possibility when $K > 2$ is to combine all causes where $\tilde{D} > 1$ into a single competing risk for the cause of interest $\tilde{D} = 1$. While the **riskRegression** package allows the use of more than 2 competing risks, we will illustrate its use considering only 2 competing risks . The package implements formula (2) in two steps.

Step 1: estimation of the cause-specific hazards

The first step is to fit the Cox regression models with the `CSC()` function in order to estimate $\lambda_{1,z}$ and $\lambda_{2,z}$:

```
cfit0 <- CSC(formula = Hist(time,status) ~ age + logthick + epicel + strata(sex),
             data = Melanoma)
coef(cfit0)
```

```
## $`Cause 1`
##           age      logthick epicelpresent
##    0.01548722    0.68178505   -0.73848649
##
## $`Cause 2`
##           age      logthick epicelpresent
##    0.07680909    0.04750975    0.31497177
```

In the call of `CSC()` the argument `formula` is used to define the outcome variables with the help of the function `prodlim::Hist()`. The variable "time" in the data set contains the values of the observed event time $\tilde{T}$ and the variable "status" the cause of the event $\tilde{D}$. Objects generated with the function `prodlim::Hist()` have a print method:

```
h <- with(Melanoma, prodlim::Hist(time,status))
h
```

```
## Right-censored response of a competing.risks model
##
## No.Observations: 205
##
## Pattern:
##
## Cause     event right.censored
##   1          57              0
##   2          14              0
##   unknown     0            134
```

and a plot method:

```
plot(h, arrowLabelStyle = "count",
     stateLabels = c("Radical\noperation", "Cancer\nrelated death", "Death\nother causes"))
```

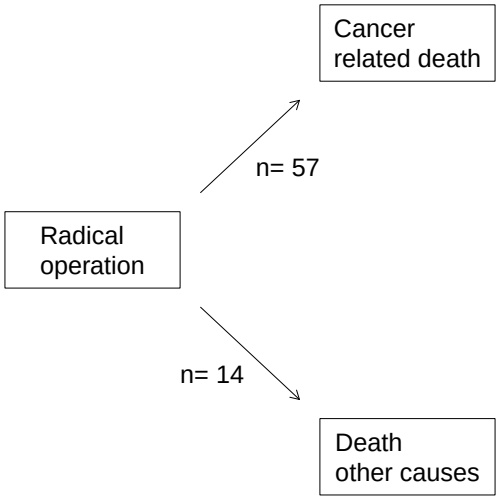


Figure 1: Box-arrow diagram showing the three states of the competing risk model and the number of observed transitions in the *Melanoma* data set.

A nice complement to the regression models is the marginal Aalen-Johansen estimate of the absolute risk of cancer related death (Figure 2):

```
library(prodlim)
plot(prodlim(Hist(time,status)  ~1, data = Melanoma),
     atrisk.at = seq(0,3652.5,365.25), xlim = c(0,3652.5),
     axis1.at = seq(0,3652.5,365.25), axis1.lab = 0:10,
     xlab = "Years", ylab = "Absolute risk", cause = 1)
```

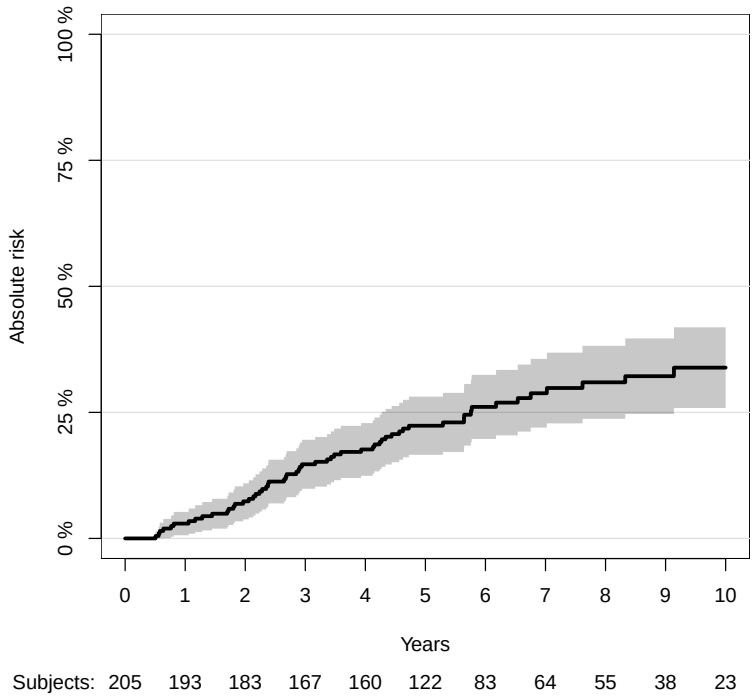


Figure 2: Non-parametric estimation of the absolute risk of cancer related death over time obtained using the Aalen-Johansen estimator.

The right hand side of the formula in the call of the `CSC()` function:

```
Hist(time,status) ~ age + logthick + epicel + strata(sex)
```

defines the covariate(s) X which enter into the linear predictor $x\beta$ in formula (1), and the strata variable(s) Z which define the baseline hazard functions $\lambda_{0j,z}$. Strata variables are specified by wrapping the variable names into the special function `strata()`, as one would do when using the `coxph()` function. If only one formula is provided, the `CSC()` function will use the same baseline covariates and strata variables for all cause-specific Cox regression models. Instead one may feed a list of formulas into the argument `formula`, one for each cause:

```
cfit1 <- CSC(formula = list(Hist(time,status) ~ age + logthick + epicel + strata(sex),
                            Hist(time,status) ~ age + strata(sex)),
             data = Melanoma)
coef(cfit1)
```

```
## $`Cause 1`
##           age      logthick epicelpresent
##    0.01548722    0.68178505   -0.73848649
##
## $`Cause 2`
##        age
## 0.07919648
```

Note that the choice of the baseline covariates relative to each cause made here is not based on clinical or statistical criteria; it was done to illustrate the software possibilities. The causes are internally ordered with respect to the levels of the variable "status", if this variable is a factor, and otherwise with respect to `sort(as.character(unique(status)))`. The order of the causes is saved as `cfit1[["causes"]]`. Accordingly, the first formula is used to fit a Cox regression model to the first cause and the second formula is used to fit a Cox regression model to the second cause and so on.

Internally, CSC() constructs dummy variables, one for each cause, and then calls the function defined by the argument fitter on a suitably constructed Surv() formula. By default the cause-specific Cox models are fitted with the function survival::coxph(). Alternatively, one can set the argument fitter to the name of a different routine, e.g., cph.

Step 2: computation of the absolute risk

The object obtained with CSC() has class "CauseSpecificCox". The second step is to call the corresponding predict() method. In addition to the object obtained with CSC() this requires three additional arguments: newdata, times, cause. The argument newdata should be a "data.frame" which contains the covariates X and Z in the same format as the data used to fit CSC(). The argument cause defines the cause of interest D and the argument times defines a vector of prediction horizon(s) whose values are used as the upper integration limit t in formula (2). The predict() method computes the absolute risks (formula (2)) for each row in newdata and each value of times:

```
newdata <- data.frame(age = c(45,67), logthick = c(0.1,0.2),
                      epicel = c("present","not present"),
                      sex = c("Female","Male"))
pfit1 <- predict(cfit1, newdata = newdata, cause = 1, times = c(867,3500))
```

By default, the product integral estimator is used to estimate the event-free survival function. Setting the argument productLimit to FALSE when calling the predict function enables to use the exponential approximation. The predict function returns a structured list of class "predictCSC". The corresponding print() method calls as.data.table.predictCSC() to display the predictions as follows:

```
print(pfit1)
```

```
##    observation age logthick      epicel    sex times     strata absRisk
## 1:           1  45      0.1     present Female   867 sex=Female   0.021
## 2:           2  67      0.2 not present   Male   867   sex=Male   0.149
## 3:           1  45      0.1     present Female  3500 sex=Female   0.117
## 4:           2  67      0.2 not present   Male  3500   sex=Male   0.428
```

For each row in newdata (values are repeated for each prediction horizon) and each prediction horizon (column times) the column "absRisk" contains the absolute risk of cancer specific mortality (cause 1). Standard errors and confidence intervals for the absolute risk can be obtained setting the argument se to TRUE:

```
pfit1se <- predict(cfit1, newdata = newdata, cause = 1, times = c(867,3500),
                   se = TRUE, keep.newdata = FALSE)
print(pfit1se)
```

```
##    observation times     strata absRisk absRisk.se absRisk.lower absRisk.upper
## 1:           1   867 sex=Female   0.021      0.122       0.00738        0.0478
## 2:           2   867   sex=Male   0.149      0.161       0.07356        0.2502
## 3:           1  3500 sex=Female   0.117      0.151       0.05552        0.2025
## 4:           2  3500   sex=Male   0.428      0.320       0.20416        0.6355
```

Here we have set the argument keep.newdata to FALSE to not export the value of the covariates. The structure of the "predictCSC" object is as follows.

```
str(pfit1se)
```

```
## List of 11
##  $ absRisk               : num [1:2, 1:2] 0.021 0.149 0.117 0.428
##  $ absRisk.se            : num [1:2, 1:2] 0.122 0.161 0.151 0.32
##  $ absRisk.lower         : num [1:2, 1:2] 0.00738 0.07356 0.05552 0.20416
##  $ absRisk.upper         : num [1:2, 1:2] 0.0478 0.2502 0.2025 0.6355
##  $ times                 : num [1:2] 867 3500
##  $ strata                : Factor w/ 2 levels "sex=Female","sex=Male": 1 2
##  $ conf.level            : num 0.95
##  $ se                    : logi TRUE
##  $ band                  : logi FALSE
##  $ nsim.band             : num 10000
##  $ transformation.absRisk:function (x)
##  - attr(*, "class")= chr "predictCSC"
```

The elements `$absRisk`, `$absRisk.se`, `$absRisk.lower` and `$absRisk.upper` are matrices where each row corresponds to a row in `newdata` and each column to a value of the `times` vector. All these matrices are sorted according to the original orders of the arguments `newdata` and `times`. To conveniently extract a subset of the results, one should first call `as.data.table.predictCSC()` to combine these results into a `"data.table"` object. Here is an example:

```
ptable1 <- as.data.table(pfit1se)
ptable1[times == 3500 & observation == 1,
        .(times,absRisk,absRisk.lower,absRisk.upper)]

##    times   absRisk absRisk.lower absRisk.upper
## 1:  3500 0.1166383    0.05551508     0.2025222
```

In the same way confidence bands can be obtained by setting the argument `band` to `TRUE`:

```
vec.times <- cfit1$eventTimes
pfit1band <- predict(cfit1, newdata = newdata[1], cause = 1,
                     times = vec.times, se = TRUE, band = TRUE)
newdata[1]

##    age logthick  epicel    sex
## 1:  45      0.1 present Female
```

By default 10,000 simulations will be used to estimate the appropriate quantile for the confidence bands (see explanations in section Construction of the confidence bands). This default behavior can be changed by setting the argument `nsim.band` to another value. The `autoplot()` function can then be used to compare confidence bands and the confidence intervals:

```
figure3 <- autoplot(pfit1band, band = TRUE, ci = TRUE)$plot
figure3 <- figure3 + xlab("Time (days)") + ylab("Absolute risk")
print(figure3)
```

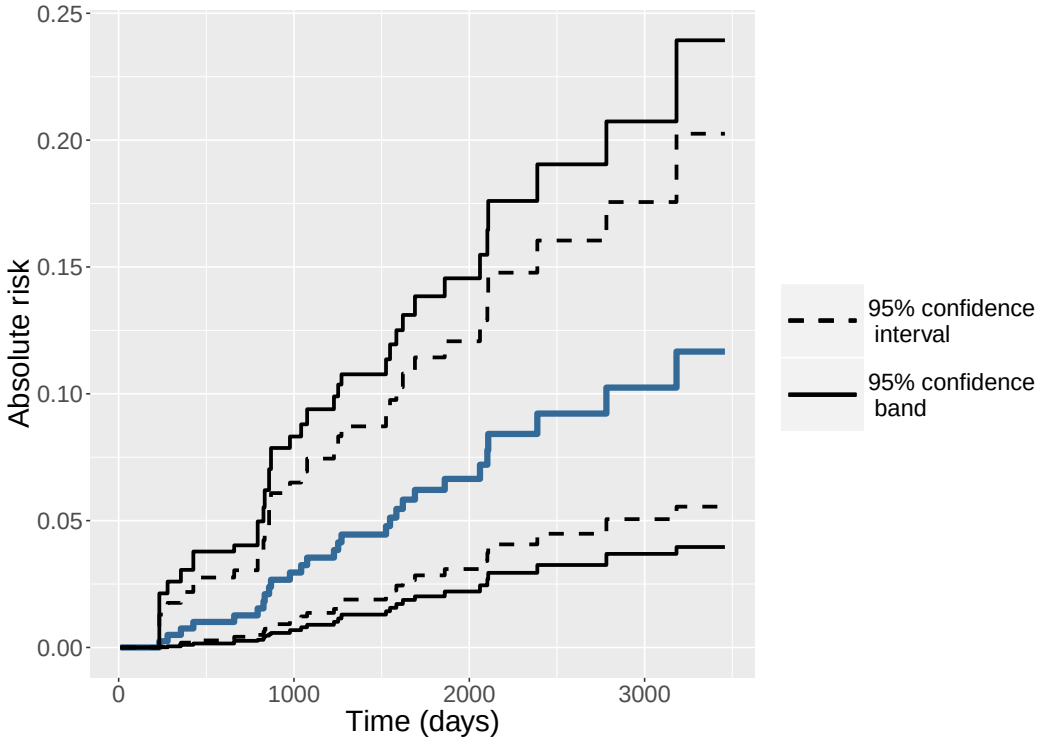


Figure 3: Absolute risk over time for a 45 years old female patient with a tumor thickness of 0.1 mm and epithelioid cells (blue line). The continuous black lines represent the confidence bands while the dashed black lines represent the range of the confidence intervals.

Note that the resulting object is a `"ggplot"` graphic. This can be useful to personalize the graph, e.g. change the font size of the text.

Construction of the confidence intervals

In this section we describe the asymptotic formula behind the confidence intervals for the predicted absolute risks and the empirical counterpart which is implemented in **riskRegression**. We assume a sample $(\mathcal{X}_i)_{i\in\{1,\dots,n\}}$ of n independent and identically distributed replications of $\mathcal{X} = (\tilde{T}, \tilde{D}, X, Z)$. The estimator $\hat{F}_1$ of F_1 is obtained by substituting the Cox partial likelihood estimate $\hat{\beta}_j$ for β_j and the baseline hazard estimate $\hat{\lambda}_{0j,z}$ for $\lambda_{0j,z}$ in equations (1) and (2).

The asymptotic confidence intervals for the covariate specific absolute risks of event 1 before time t are based on the following von Mises expansion (van der Vaart, 1998):

$$\sqrt{n}(\hat{F}_1(t|x,z) - F_1(t|x,z)) = \frac{1}{\sqrt{n}} \sum_{i=1}^{n} \phi_{F_1}(\mathcal{X}_i; t, x, z) + o_P(1). \tag{4}$$

where the exponential approximation is used for defining the event free survival in $F_1(t|x,z)$. Given (4), for fixed values t, x, z the central limit theorem implies that $\hat{F}_1(t|x,z)$ has an asymptotic normal distribution with asymptotic variance $V_{F_1}(t,x,z) = \mathrm{E}(\phi_{F_1}(\mathcal{X}_i; t, x, z)^2)$. Based on the estimate $\widehat{\phi}_{F_1}$ of the influence function ϕ_{F_1} (both defined in subsequent subsections) our variance estimate is given by

$$\widehat{V}(t,x,z) = \frac{1}{n} \sum_{i=1}^{n} \widehat{\phi}_{F_1}(\mathcal{X}_i; t, x, z)^2. \tag{5}$$

We then construct Wald confidence intervals for $F_1(t|x,z)$ in the usual way:

$$\left[F_1(t|x,z) + q_{\alpha/2}\sqrt{\widehat{V}(t,x,z)}); F_1(t|x,z) + q_{1-\alpha/2}\sqrt{\widehat{V}(t,x,z)}\right]$$

where q_α is the α-quantile of the normal distribution. Since the absolute risk is bounded below by 0 and above by 1, the confidence interval is automatically restricted to this interval. Alternatively the confidence interval can be computed using a log-log transformation: first the confidence interval is computed on the log-log scale:

$$\left[\log(-\log(F_1(t|x,z))) + q_{\alpha/2}\sqrt{\widehat{V}_{log-log}(t,x,z)}); \log(-\log(F_1(t|x,z))) - q_{1-\alpha/2}\sqrt{\widehat{V}_{log-log}(t,x,z)}\right]$$

where $V_{log-log}$ is the variance of the influence function on the log-log scale. Then the confidence interval is back transformed using the link function: $x \mapsto \exp(-\exp(x))$. This ensures that the confidence interval is bounded below by 0 and above by 1. By default, the confidence intervals are computed using the log-log transformation. To compute them without using the log-log transformation, the argument `log.transform` needs to be set to `FALSE` when calling `predict.CauseSpecificCox()`:

```
pfit2se <- predict(cfit1, newdata = newdata, cause = 1, times = c(867,3500),
                   se = TRUE, log.transform = FALSE, keep.newdata = FALSE)
print(pfit2se)
```

```
##    observation times     strata absRisk absRisk.se absRisk.lower absRisk.upper
## 1:           1   867 sex=Female   0.021    0.00992       0.00157        0.0404
## 2:           2   867   sex=Male   0.149    0.04586       0.05945        0.2392
## 3:           1  3500 sex=Female   0.117    0.03795       0.04226        0.1910
## 4:           2  3500   sex=Male   0.428    0.11620       0.20021        0.6557
```

Here the column "absRisk.se" contains $V_{log-log}$ (log-log scale) while the columns "absRisk", "absRisk.lower", and "absRisk.upper" are on the original scale. The benefit of using a log-log transformation is studied in the section Coverage of the confidence interval and the confidence bands.

Asymptotic formula

The influence function ϕ_{F_1} can be expressed as a function of the influence functions ϕ_{λ_1} and ϕ_{λ_2} of the cause-specific hazard rates:

$$\begin{aligned}\phi_{F_1}(\mathcal{X}, t, x, z) = &\int_0^t \exp(-\Lambda_{1,z}(s|x) - \Lambda_{2,z}(s|x)) d\phi_{\Lambda_{1,z}}(\mathcal{X}; s, x) \\ &- \int_0^t \lambda_{1,z}(s|x) \exp(-\Lambda_{1,z}(s|x) - \Lambda_{2,z}(s|x)) \left(\phi_{\Lambda_{1,z}}(\mathcal{X}; s, x) + \phi_{\Lambda_{2,z}}(\mathcal{X}; s, x)\right) ds.\end{aligned} \tag{6}$$

We use the shorthand notations $v^{\otimes 0} = 0, v^{\otimes 1} = v, v^{\otimes 2} = vv^\intercal$ and

$$\int f(s,d,v,w)\mathrm{dP}(s,d,v,w) = \int_0^\infty \sum_{d=0}^{K} \int_{R^p} \sum_{z=1}^{L} f(s,d,v,w)\mathrm{dP}(s,d,v,w).$$

We also suppress the dependence on τ when we in the following adapt the usual notation for Cox model asymptotic theory to the cause-specific and stratified case:

$$\begin{aligned}
\mathcal{S}^{(r)}(t,\beta_j,z) &= \int 1\{t \le s \le \tau, w = z\} \exp(v\beta_j) v^{\otimes r} \mathrm{dP}(s,d,v,w) \\
E(t,\beta_j,z) &= \frac{\mathcal{S}^{(1)}(t,\beta_j,z)}{\mathcal{S}^{(0)}(t,\beta_j,z)} \\
\mathcal{I}(\beta_j) &= \int 1\{d=j\} \left(\frac{\mathcal{S}^{(2)}(s,\beta_l,w)}{\mathcal{S}^{(0)}(s,\beta_j,w)} - E(s,\beta_j,w)^{\otimes 2} \right) \mathrm{dP}(s,d,v,w).
\end{aligned}$$

For fixed cause j, time t and strata z the expansions $\sqrt{n}(\hat\beta_j - \beta_j) = \frac{1}{\sqrt{n}} \sum_{i=1}^n \phi_{\beta_j}(\mathcal{X}_i) + o_\mathrm{P}(1)$ and $\sqrt{n}(\hat\Lambda_{0j,z}(t) - \Lambda_{0j,z}(t)) = \frac{1}{\sqrt{n}} \sum_{i=1}^n \phi_{\Lambda_{0j,z}}(\mathcal{X}_i, t) + o_\mathrm{P}(1)$ are then characterized by the influence functions

$$\begin{aligned}
\phi_{\beta_j}(\mathcal{X}) = \mathcal{I}(\beta_j)^{-1} \Bigg(& 1\{\tilde D = j, \tilde T < \tau\} \left(X - E(\tilde T, \beta_j, Z) \right) \\
& - \exp(X\beta_j) \int 1\{d = j, s \le \tilde T\} \frac{X - E(s,\beta_j,Z)}{\mathcal{S}^{(0)}(s,\beta_j,Z)} \mathrm{d}P(s,d,v,w) \Bigg) \qquad (7)
\end{aligned}$$

$$\begin{aligned}
\phi_{\Lambda_{0j,z}}(\mathcal{X};t) = & -\phi_{\beta_j}(\mathcal{X}) \int_0^t E(s,\beta_j,z)\lambda_{0j,z}(s)ds \\
& - 1\{Z = z\} \Bigg(\exp(X\beta_j) \int_0^{min(t,\tilde T)} \frac{\lambda_{0j,z}(s)}{\mathcal{S}^{(0)}(s,\beta_j,z)} ds - \frac{1\{\tilde T \le t, \tilde D = j\}}{\mathcal{S}^{(0)}(\tilde T, \beta_j, Z)} \Bigg). \qquad (8)
\end{aligned}$$

In absence of strata, formula (7) is equal to formula 2 of (Reid, 1981) and formula (8) equals the one given in Gerds and Schumacher (2001, top of page 576; note however that there is a sign mistake in their first term). To connect these formulas with formula (6) it remains to note that under the Cox regression model the influence function of the cause-specific hazard rate can be written as:

$$\phi_{\Lambda_{j,z}}(\mathcal{X};t,x) = \exp(x\beta_j) \left(\phi_{\Lambda_{0j,z}}(\mathcal{X};t) + \Lambda_{0j,z}(t)\ x\ \phi_{\beta_j}(\mathcal{X})^\intercal \right). \qquad (9)$$

Empirical estimates

The following formulas are obtained with the plug-in principle substituting the Cox partial likelihood estimates $\hat\beta_j$ for β_j and the baseline hazard estimates $\hat\lambda_{0j,z}$ for $\lambda_{0j,z}$ into formulas (6) - (9). We denote by $N_j^z(t) = \sum_{i=1}^n 1\{\tilde T_i \le t, \tilde D_i = j, Z_i = z\}$ the strata and cause specific counting process, and by $Y^z(t) = \sum_{i=1}^n 1\{T_i \ge t, Z_i = z\}$ the strata specific "at-risk" process (Andersen et al., 1993). The empirical estimate of the influence function of the partial likelihood estimate is given by

$$\widehat{\phi}_{\beta_j}(\mathcal{X}_i) = \widehat{\mathcal{I}}(\beta_j)^{-1} \left(\Delta_i \left(X_i - \hat E(\tilde T_i, \hat\beta_j, Z_i) \right) - \exp(X_i\hat\beta_j) \int_0^{T_i} \frac{X_i - \hat E(s,\hat\beta_j,Z_i)}{\widehat{\mathcal{S}^{(0)}}(s,\hat\beta_j,Z_i)} \mathrm{d}N_j(s) \right)$$

where

$$\widehat{\mathcal{S}^{(r)}}(t,\hat\beta_j,z) = \int_0^\tau \exp(X_i\hat\beta_j) X_i^{\otimes r} \mathrm{d}N_j^z(t).$$

The influence functions for the cumulative baseline hazard and its first differential are estimated by:

$$\begin{aligned}\widehat{\phi}_{\Lambda_{0j,z}}(\mathcal{X}_i;t) &= -\widehat{\phi}_{\beta_j}(\mathcal{X}_i)\int_0^t E(s,\hat{\beta}_j,z)\hat{\lambda}_{0j,z}(s)\mathrm{d}s \\ &\quad - 1\{Z_i=z\}\left(\exp(X_i\hat{\beta}_j)\int_0^{min(t,\tilde{T}_i)}\frac{\hat{\lambda}_{0j,z}(s)}{\widehat{\mathcal{S}^{(0)}}(s,\hat{\beta}_j,z)}ds - \frac{1\{\tilde{T}_i\le t,\tilde{D}_i=j\}}{\widehat{\mathcal{S}^{(0)}}(\tilde{T}_i,\hat{\beta}_j,Z_i)}\right) \qquad (10)\\ d\widehat{\phi}_{\Lambda_{0j,z}}(\mathcal{X}_i;t) &= -\widehat{\phi}_{\beta_j}(\mathcal{X}_i)E(t,\hat{\beta}_j,z)\hat{\lambda}_{0j,z}(t) \\ &\quad - 1\{Z_i=z\}\left(\exp(X_i\hat{\beta}_j)1\{t\le\tilde{T}_i\}\frac{\hat{\lambda}_{0j,z}(t)}{\widehat{\mathcal{S}^{(0)}}(t,\hat{\beta}_j,z)} - \frac{1\{\tilde{T}_i=t,\tilde{D}_i=j\}}{\widehat{\mathcal{S}^{(0)}}(\tilde{T}_i,\hat{\beta}_j,Z_i)}\right). \qquad (11)\end{aligned}$$

These estimates lead to the following estimates of the influence functions of the covariate specific cumulative hazard and its derivative relative to the time:

$$\hat{\phi}_{\Lambda_{j,z}}(\mathcal{X}_i;t,x) = \exp(x\hat{\beta}_j)\left(\widehat{\phi}_{\Lambda_{0j,z}}(\mathcal{X}_i;t) + \hat{\Lambda}_{0j,z}(t)\ x\ \widehat{\phi}_{\beta_j}(\mathcal{X}_i)^\intercal\right) \qquad (12)$$

$$d\hat{\phi}_{\Lambda_{j,z}}(\mathcal{X}_i;t,x) = \exp(x\hat{\beta}_j)\left(d\widehat{\phi}_{\Lambda_{0j,z}}(\mathcal{X}_i;t) + \hat{\lambda}_{0j,z}(t)\ x\ \widehat{\phi}_{\beta_j}(\mathcal{X}_i)^\intercal\right). \qquad (13)$$

Finally, we obtain our estimate of the influence function of the absolute risk:

$$\widehat{\phi}_{F_1}(\mathcal{X}_i,t,x,z) = \int_0^t \exp(-\hat{\Lambda}_{1,z}(s|x) - \hat{\Lambda}_{2,z}(s|x))d\widehat{\phi}_{\Lambda_{1,z}}(\mathcal{X}_i;s,x) \qquad (14)$$

$$-\int_0^t \hat{\lambda}_{1,z}(s|x)\exp(-\hat{\Lambda}_{1,z}(s|x) - \hat{\Lambda}_{2,z}(s|x))\left(\widehat{\phi}_{\Lambda_{1,z}}(\mathcal{X}_i;s,x) + \widehat{\phi}_{\Lambda_{2,z}}(\mathcal{X}_i;s,x)\right)\mathrm{d}s. \qquad (15)$$

Construction of the confidence bands

A confidence band with confidence level $1-\alpha$ for the absolute risk $F_1(.|x,z)$ restricted to the time interval $\mathcal{T}=[\tau_1;\tau_2]$ is a region $R_{\mathcal{T}}(x,z)=[l_{x,z}(t);u_{x,z}(t)]_{t\in\mathcal{T}}$ satisfying:

$$\mathrm{P}(F_1(t|x,z)\in[l_{x,z}(t);u_{x,z}(t)]\,|\forall t\in\mathcal{T}) = 1-\alpha.$$

In figure Figure 3, $\tau_1=0$ and $\tau_2=3458$, the time at which the last event occurred. Using the martingale central limit theorem, (Cheng et al., 1998) have shown that $\hat{F}_1(t|x,z)-F_1(t|x,z)$ converges weakly to a zero-mean Gaussian process on $\mathcal{T}$. The asymptotic variance of this process is $V(t,x,z)$. However the $1-\alpha$ quantile achieving simultaneous coverage is larger than the $1-\alpha$ quantile of a standard normal distribution. Since the dependence between the increments of the process $\hat{F}_1(t|x,z)-F_1(t|x,z)$ makes the derivation of an explicit expression for the quantile difficult, we used instead a resampling technique (Scheike and Zhang, 2008). Consider over $t\in\mathcal{T}$ the normalized process:

$$\psi_{F_1}(\mathcal{X}_i;t,x,z) = \phi_{F_1}(\mathcal{X}_i;t,x,z)/\sqrt{V(t,x,z)}.$$

Denote by $c_{1-\alpha/2}$ the $1-\alpha/2$ quantile of the sample:

$$\sup_{t\in\mathcal{T}}|\psi_{F_1}(\mathcal{X}_i;t,x,z)|$$

and using the symmetry of the Gaussian distribution, i.e. $c_{\alpha/2}=-c_{1-\alpha/2}$, a $1-\alpha$ confidence band over $\mathcal{T}$ is constructed as follows:

$$\left[F_1(t|x,z) - c_{1-\alpha/2}\sqrt{\widehat{V}(t,x,z)});F_1(t|x,z) + c_{1-\alpha/2}\sqrt{\widehat{V}(t,x,z)}\right].$$

Like for the confidence intervals, the confidence bands will be restricted to the interval [0;1] when they are not computed using a log-log transformation.

Implementation details

The function `predict.CauseSpecificCox()` calls two important functions, `predictCox()` that computes the hazard and cumulative hazard for a fitted Cox model, and `iidCox()` that computes the influence function for the baseline hazard and regression coefficients. In this section we first explain

how predictCox() deals with ties in the event times. We then show that the function predictCox() can also be used to obtain confidence intervals and bands for covariate specific survival probabilities in the situation without competing risks. Implementation details about the function iidCox() are postponed to appendix B. These details may be useful for programmers who need to care about memory usage.

Handling of tied event times

We speak of ties when two or more observations have the same value of the time variable $\tilde{T}$. Ties occur for example when time is recorded on a discrete scale such as months. The **survival** package implements three different methods to deal with ties ("efron", "breslow", and "exact", see help(coxph)) for the partial likelihood estimator of the log hazard ratios β_j (Therneau and Grambsch, 2000). We have implemented the "efron" and the "breslow" method but not the "exact" method. For a comparison of these methods and yet another method see Scheike and Sun (2007). We now state the formula for the baseline hazard function under Breslow's (Breslow, 1974) and Efron's method (Efron, 1977) for the handling of ties. The baseline hazard estimate in strata z given by the Breslow method is:

$$d\hat{\Lambda}^B_{0j,z}(t) = \frac{\mathrm{d}N^z_j(t)}{\sum_{i\in Y^z(t)} \exp(\beta^t X_i)}.$$

With the Efron method for handling ties the formula is given by:

$$d\hat{\Lambda}^E_{0j,z}(t) = \sum_{k=1}^{\mathrm{d}N^z_j(t)} \frac{1}{\sum_{i\in Y^z(t)} \exp(\beta^t X_i) - \frac{k-1}{\mathrm{d}N^z_j(t)} \sum_{i=1}^{\mathrm{d}N^z_j(t)} \exp(\beta^t X_i)}.$$

Both estimators are implemented in the function predictCox() which provides estimates of the baseline hazard, the cumulative baseline hazard and baseline survival. The predictCox() function does not have an argument to specify whether Breslow method or Efron method should be used; instead it uses the same method that has been used to estimate the regression coefficients. In the case of the coxph() function, the default method is Efron:

```
f1 <- coxph(Surv(time,status != 0) ~ age + logthick + epicel + strata(sex),
            data = Melanoma, x = TRUE, y = TRUE)
f1$method
```

```
## [1] "efron"
```

Therefore the baseline hazard will be estimated using the Efron method when calling predictCox():

```
baseH1 <- predictCox(f1)
as.data.table(baseH1[c("time","cumhazard","strata","survival")])
```

```
##      time  cumhazard strata  survival
##   1:   99 0.00623279 Female 0.9937866
##   2:  232 0.01256406 Female 0.9875145
##   3:  279 0.01897728 Female 0.9812017
##   4:  295 0.02555481 Female 0.9747690
##   5:  355 0.03221850 Female 0.9682950
##  ---
## 199: 3909 0.69673810   Male 0.4982078
## 200: 4119 0.69673810   Male 0.4982078
## 201: 4207 0.69673810   Male 0.4982078
## 202: 4310 0.69673810   Male 0.4982078
## 203: 4492 0.69673810   Male 0.4982078
```

The cumulative baseline hazard and baseline survival are displayed in the columns "cumHazard", and "survival" of the output. This corresponds, respectively, to $\Lambda_{0j,z}(t)$ and $exp(-\Lambda_{0j,z}(t))$ where j is 1, z can be found in the column "strata" and t in "time". The covariate specific cumulative hazard $\Lambda_{j,z}(t|x)$ and survival $exp(-\Lambda_{j,z}(t|x))$ can be estimated using the same function:

```
predictCox(f1, newdata = Melanoma[c(17,101,123),],
           times = c(7,3,5)*365.25)
```

```
##    observation times strata cumhazard survival
## 1:           1  2557   Male     0.884    0.413
```

```
## 2:           2  2557 Female     0.555    0.574
## 3:           3  2557 Female     0.949    0.387
## 4:           1  1096   Male     0.453    0.635
## 5:           2  1096 Female     0.202    0.817
## 6:           3  1096 Female     0.346    0.708
## 7:           1  1826   Male     0.670    0.512
## 8:           2  1826 Female     0.366    0.693
## 9:           3  1826 Female     0.626    0.535
```

Confidence intervals and confidence bands for survival probabilities

In absence of competing risks, the influence function of a Cox model can be used to estimate confidence intervals for the survival. This can be done by setting the argument `se` to `TRUE` when calling the `predictCox()` function:

```
p1 <- predictCox(f1, newdata = Melanoma[3:5,],
                 times = c(Melanoma$time[5:7],1000),
                 se = TRUE, type="survival")
```

The `predictCox()` function output an object of class `"predictCox"`. The `print()` method can be used to display the confidence intervals for the survival computed at different times:

```
print(p1)
```

```
##     observation times strata survival survival.se survival.lower survival.upper
##  1:           1   185   Male    0.980       0.616          0.935          0.994
##  2:           2   185 Female    0.985       1.014          0.893          0.998
##  3:           3   185   Male    0.947       0.652          0.823          0.985
##  4:           1   204   Male    0.973       0.567          0.921          0.991
##  5:           2   204 Female    0.985       1.014          0.893          0.998
##  6:           3   204   Male    0.929       0.594          0.791          0.977
##  7:           1   210   Male    0.967       0.503          0.913          0.987
##  8:           2   210 Female    0.985       1.014          0.893          0.998
##  9:           3   210   Male    0.912       0.552          0.762          0.969
## 10:           1  1000   Male    0.864       0.321          0.760          0.925
## 11:           2  1000 Female    0.769       0.286          0.631          0.860
## 12:           3  1000   Male    0.673       0.391          0.426          0.832
```

Here the confidence intervals were computed using a log-log transformation. The argument `log.transform` can be set to `FALSE` to compute them without using the log-log transformation. Confidence bands can also be obtained using `predictCox()` by setting the argument band to `TRUE`. As for the `predict.CauseSpecificCox()` function, 10,000 simulations will be used to compute the confidence bands; this can be changed specifying the argument `nsim.band`. The function `as.data.table.predictCox()` makes it easy to extract subsets from `"predictCox"` object:

```
p1 <- as.data.table(p1)
p1[times == 185,]
```

```
##    observation times strata  survival survival.se survival.lower survival.upper
## 1:           1   185   Male 0.9801395   0.6163925      0.9350597      0.9940246
## 2:           2   185 Female 0.9845660   1.0137483      0.8927600      0.9978695
## 3:           3   185   Male 0.9470887   0.6524116      0.8226132      0.9849795
```

Coverage of the confidence interval and the confidence bands

To assess the validity of the estimation of the standard error, we performed a simulation study. The sample size was varied between 50 and 10000. For each sample size, 5000 datasets were simulated using the `SimCompRisk()` function from the **riskRegression** package. The time of first event typically ranged between 0.001 and 20 with a median around 4. For each dataset, a Cox model specific to cause 1 and a 2 cause-specific Cox model were fitted considering 2 covariates ($X1$ and $X2$). The survival, absolute risk, their confidence intervals were estimated (with or without log-log transformation) at time 1, 1.5, 2, 3, 4, 5, 6, and 7 conditional on $X1 = 0$ and $X2 = 1$. The confidence bands over those 8 times were also computed.

The true absolute risk was defined as the median of the absolute risks over the 5000 datasets. The coverage of the confidence intervals was computed as the percentage of times that the true absolute

risk was inside the confidence interval, at a given time. The coverage of the confidence bands was computed as the percentage of times that the true absolute risk was inside the confidence bands, simultaneously for all of the 8 times.

As expected, the coverage of the confidence intervals is improving with increasing sample size and reaches its nominal level of 95% at around sample size 1000 (figure Figure 4, left panel). Using a log-log transformation leads to better small sample properties and similar large sample properties (figure Figure 4, right panel). A larger sample size was necessary for the confidence bands to converge toward the nominal coverage level (n = 5000, figure Figure 5 left panel). Again log-log transformation leads to better small sample properties. We only displayed here the coverage for the absolute risk but similar coverage was obtained for the survival function.

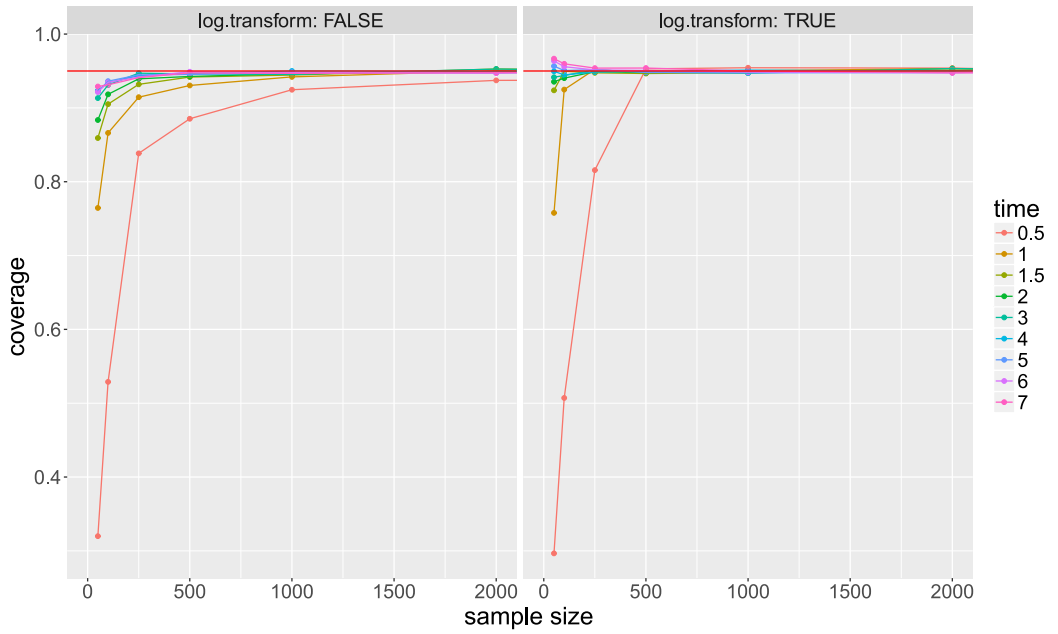


Figure 4: Coverage of the asymptotic confidence interval of the absolute risk plotted against the sample size. Each color corresponds to a prediction time. The figure is only shown for samples size below 2000 since for larger sample sizes the coverage is always approximately equal to 0.95. Left panel: confidence intervals are computed on the original scale. Right panel: confidence intervals are computed on the log-log scale and back-transformed.

Runtime and memory usage

Baseline hazard

We compare the performance of `predictCox()` regarding the estimate of the baseline hazard function with that of the function `survival::basehaz()`. For this purpose we simulate data with 10 covariates including both continuous and discrete type using the `sampleData()`.

In our performance study we vary the sample size ranging from 500 to 1,000,000 observations and consider both stratified:

```
Surv(time,event) ~ strata(X1) + X2 + X3 + X4 + X5 + X6 + X7 + X8 + X9 + X10
```

and non-stratified Cox regression models:

```
Surv(time,event) ~ X1 + X2 + X3 + X4 + X5 + X6 + X7 + X8 + X9 + X10
```

The computation times were estimated using the **rbenchmark** package (Kusnierczyk, 2012) and averaged across 100 simulated data sets. The memory usage was estimated using the **profvis** package (Chang and Luraschi, 2017) and averaged across 10 simulations. When the execution time of the function was extremely fast (i.e. <0.005s), the memory usage could not be reliably assessed and was set to `NA`.

Memory usage and computation time are displayed on figure Figure 6 and Figure 7. Both functions lead to a reasonable computation time (<1 min) even when considering very large datasets (e.g. >10,000 observations). Nevertheless the `predictCox()` function outperforms the `basehaz()` function by a factor

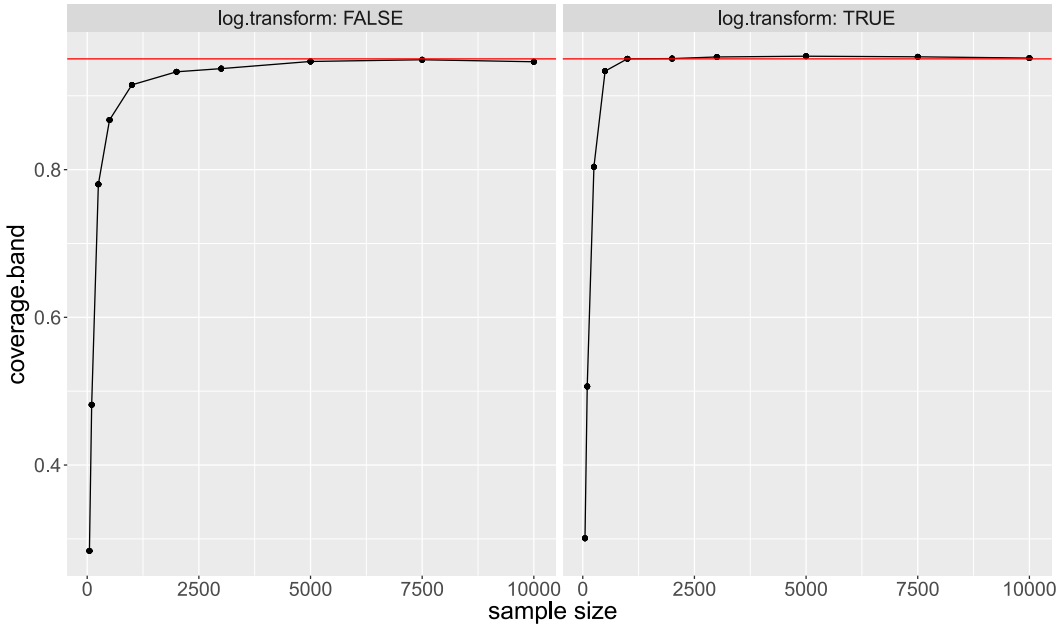


Figure 5: Coverage of the asymptotic confidence band of the absolute risk plotted against the sample size. Left panel: confidence intervals are computed on the original scale. Right panel: confidence intervals are computed on the log-log scale and back-transformed.

varying between 3 and 11 in terms of computation time. The gain in speed is especially expressed in large datasets. Memory usage is also lower for `predictCox()` and decreases by a factor between 1 and 1.6 as compared to `basehaz()`.

Absolute risk

We now compare two implementations for computing the standard errors of the absolute risk. The default implementation corresponds to setting the argument `store.iid` to `"full"` when calling `predict.CauseSpecificCox()` or `predictCox()` while the second is obtained by setting `store.iid` to `"minimal"`. The two implementations are described in more detail appendix B.

First we compare their computation time and memory consumption when making only one prediction. As before, we simulated datasets for $K = 2$ using the `SimCompRisk()` function with increasing sample size. Then, the two cause-specific Cox models were fitted to each of the simulated datasets using `riskRegression::CSC`. Then we measured the computation time and memory usage necessary to estimate the absolute risk with its standard error for the first observation at time 4, when using the argument `store.iid="minimal"` or `store.iid="full"` in `predict.CauseSpecificCox()`.

The results are show on figure Figure 8. While both implementations lead to a similar computation time, the memory usage for `store.iid="minimal"` grows linearly while for `store.iid="full"` it grows approximately in $n^{1.4}$. However, if instead of estimating the absolute risk with its standard error for one prediction, we estimate it for all the observations the implementation `store.iid="minimal"` becomes slower compared to `store.iid="full"` (e.g. 28.5 minutes vs. 6 minutes for n=2000).

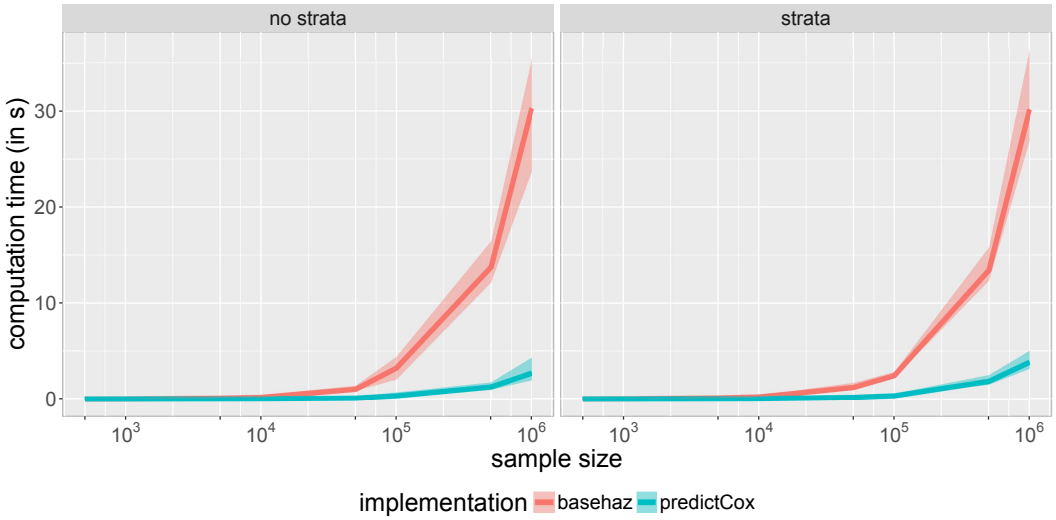


Figure 6: The computation time (in seconds) of `predictCox()` and `basehaz()` plotted against the sample size for a Cox model (left panel) and a stratified Cox model (right panel). The x axis is displayed using a logarithmic scale but its labels refer to the (untransformed) sample size. The curves represent the median values over 100 simulations while the shaded areas represent the 95% confidence intervals.

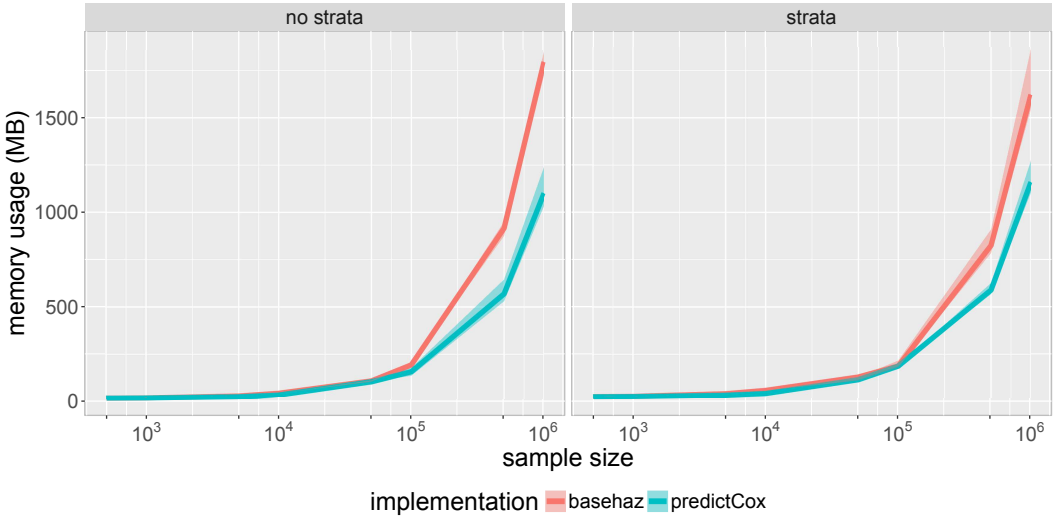


Figure 7: The memory usage (in megabytes) of `predictCox()` and `basehaz()` plotted against the sample size for a Cox model (left panel) and a stratified Cox model (right panel). The x axis is displayed using a logarithmic scale but its labels refer to the (untransformed) sample size. The curves represent the median values over 100 simulations while the shaded areas represent the 95% confidence intervals.

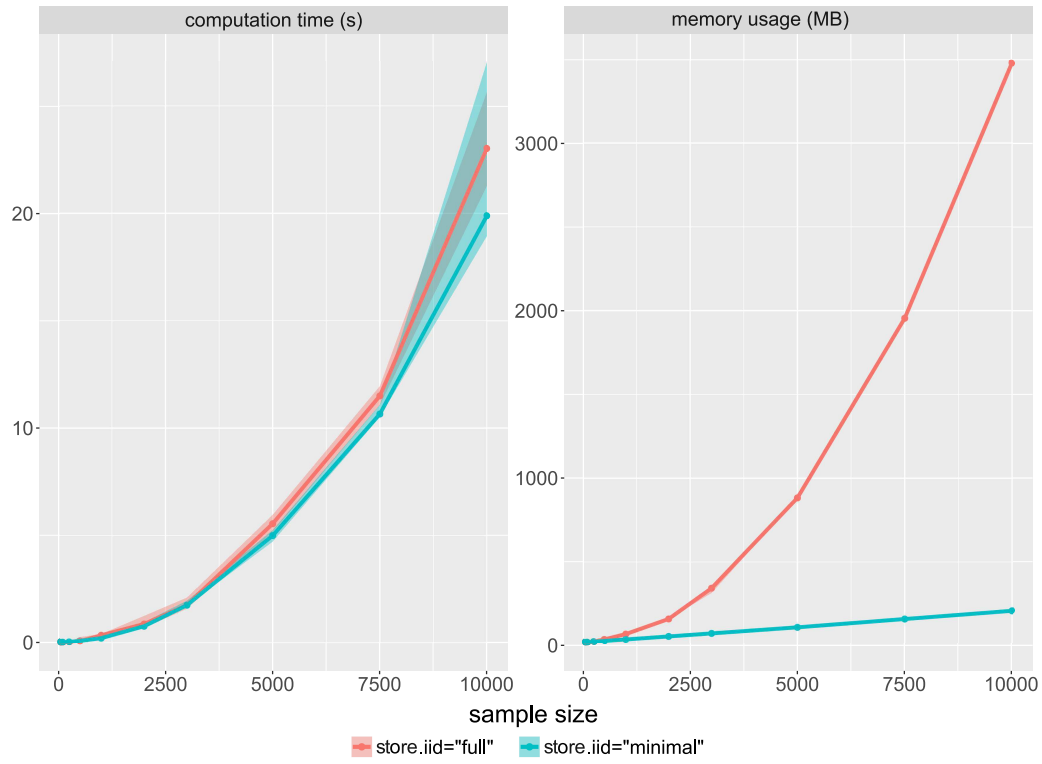


Figure 8: The computation time (left panel) and memory usage (right panel) for computing the absolute risk with its standard error for one observation plotted against the sample size, setting the argument `store.iid` to `"full"` or `"minimal"` when calling `predict.CauseSpecificCox()`. The curves represent the median values over 100 simulations while the shaded areas represent the 95% confidence intervals.

Summary

This paper introduces new features of the **riskRegression** package for prediction of absolute risks from cause-specific Cox regression models using computationally efficient functions. Table 1 summarizes the main functions described in this paper. Confidence intervals and confidence bands for the absolute risks can be computed using the `predict()` function and displayed using the `print()` method. The `predictCox()` function can be applied on `"coxph"` and `"cph"` objects to predict the survival with its confidence interval or confidence bands. In both cases, the `autoplot()` function can display the predicted risk (or survival) over time. When dealing with small to moderate sample sizes, we advise to compute confidence intervals or confidence bands using a log-log transformation (argument `log.transform`).

Table 1: Functions implemented in the **riskRegression** package for making prediction from Cox regression models

`CSC()`	Fit cause-specific Cox models
`predict()`	Predict covariate specific absolute risks for given time horizons
`iidCox()`	Compute the influence function of the baseline hazard estimator and of the partial likelihood estimator of the regression coefficients
`predictCox()`	Compute the (cumulative) baseline hazard function and predictions of hazards and survival probabilities in new data
`autoplot()`	Graphical display of the predicted risk across time

Brice Ozenne
Section of Biostatistics, Department of Public Health, University of Copenhagen
Østerfarimagsgade 5, 1014 Copenhagen
Denmark
broz@sund.ku.dk

Anne Lyngholm Sørensen
Section of Biostatistics, Department of Public Health, University of Copenhagen
Østerfarimagsgade 5, 1014 Copenhagen
Denmark
als@sund.ku.dk

Thomas Scheike
Section of Biostatistics, Department of Public Health, University of Copenhagen
Østerfarimagsgade 5, 1014 Copenhagen
Denmark
thsc@sund.ku.dk

Christian Torp-Pedersen
Public Health and Epidemiology Group, Department of Health Science and Technology, Aalborg University
Niels Jernes Vej 12, 9220 Aalborg
Denmark
ctp@hst.aau.dk

Thomas Alexander Gerds
Section of Biostatistics, Department of Public Health, University of Copenhagen
Østerfarimagsgade 5, 1014 Copenhagen
Denmark
tag@biostat.ku.dk

Bibliography

P. K. Andersen, Ø. Borgan, R. D. Gill, and N. Keiding. *Statistical Models Based on Counting Processes*. Springer Series in Statistics. Springer-Verlag, New York, 1993. URL http://dx.doi.org/10.1007/

978-1-4612-4348-9. [p441, 447]

J. Benichou and M. H. Gail. Estimates of absolute cause-specific risk in cohort studies. *Biometrics*, 46 (3):813–826, 1990. URL https://dx.doi.org/10.2307/2532098. [p440, 441]

N. Breslow. Covariance analysis of censored survival data. *Biometrics*, 30(1):89–99, 1974. URL https://dx.doi.org/10.2307/2529620. [p449]

W. Chang and J. Luraschi. *Profvis: Interactive Visualizations for Profiling R Code*, 2017. URL https://CRAN.R-project.org/package=profvis. R package version 0.3.3. [p451]

S. Cheng, J. P. Fine, and L. Wei. Prediction of cumulative incidence function under the proportional hazards model. *Biometrics*, pages 219–228, 1998. URL https://dx.doi.org/10.2307/2534009. [p448]

D. R. Cox. Regression models and life-tables. *Journal of the Royal Statistical Society B*, 34(2):187–220, 1972. URL http://dx.doi.org/10.1007/978-1-4612-4380-9_37. [p441]

B. Efron. The efficiency of Cox's likelihood function for censored data. *Journal of American Statistical Association*, 72(359):557–565, 1977. URL https://dx.doi.org/10.2307/2286217. [p449]

J. P. Fine and R. J. Gray. A proportional hazards model for the subdistribution of a competing risk. *Journal of the American Statistical Association*, 94(446):496–509, 1999. URL https://dx.doi.org/10.1080/01621459.1999.10474144. [p440]

T. Gerds and M. Schumacher. On functional misspecification of covariates in the Cox regression model, 2001. ISSN 0006-3444. URL https://dx.doi.org/10.1093/biomet/88.2.572. [p447]

T. A. Gerds, T. H. Scheike, and P. K. Andersen. Absolute risk regression for competing risks: Interpretation, link functions, and prediction. *Statistics in Medicine*, 31(29):3921–3930, 2012. URL https://dx.doi.org/10.1002/sim.5459. [p440]

F. E. Harrell Jr. *Rms: Regression Modeling Strategies*, 2017. URL https://CRAN.R-project.org/package=rms. R package version 5.1-1. [p440]

K. K. Holst and T. Scheike. *Mets: Analysis of Multivariate Event Times*, 2017. URL https://CRAN.R-project.org/package=mets. R package version 1.2.2. [p457]

W. Kusnierczyk. *Rbenchmark: Benchmarking Routine for R*, 2012. URL https://CRAN.R-project.org/package=rbenchmark. R package version 1.0.0. [p451]

H. Putter, L. de Wreede, M. Fiocco, and with contributions by Ronald Geskus. *Mstate: Data Preparation, Estimation and Prediction in Multi-State Models*, 2016. URL https://CRAN.R-project.org/package=mstate. R package version 0.2.10. [p440]

N. Reid. Influence functions for censored data. *The Annals of Statistics*, 9(1):78–92, 1981. URL http://dx.doi.org/10.1214/aos/1176345334. [p447]

T. H. Scheike and Y. Sun. Maximum likelihood estimation for tied survival data under Cox regression model via EM-algorithm. *Lifetime Data Analysis*, 13:399–420, 2007. URL https://dx.doi.org/10.1007/s10985-007-9043-3. [p449]

T. H. Scheike and M.-J. Zhang. Flexible competing risks regression modeling and goodness-of-fit. *Lifetime data analysis*, 14(4):464–483, 2008. URL http://dx.doi.org/10.1007/s10985-008-9094-0. [p448]

T. H. Scheike, M. J. Zhang, and T. A. Gerds. Predicting cumulative incidence probability by direct binomial regression. *Biometrika*, 95(1):205–220, 2008. URL https://dx.doi.org/10.1093/biomet/asm096. [p440]

T. M. Therneau. *Survival: Survival Analysis*, 2017. URL https://CRAN.R-project.org/package=survival. R package version 2.41-3. [p440]

T. M. Therneau and P. M. Grambsch. *Modeling Survival Data: Extending the Cox Model*. Springer-Verlag, 2000. URL https://dx.doi.org/10.1007/978-1-4757-3294-8. [p449]

A. van der Vaart. *Asymptotic Statistics*. Cambridge University Press, 1998. URL http://dx.doi.org/10.1017/CBO9780511802256. [p446]

Appendix A Modularity

The CSC() function requires a routine to estimate the regression coefficients of the Cox model. By default, CSC() calls the coxph() function from the **survival** package to do so. This has several reasons: coxph() has been thoroughly tested, is reasonably fast, widely used and provides flexible modeling options. However in specific contexts, other routines may be more appropriate, e.g. faster. Also, developers that have implemented their own routine may be interested in computing the baseline hazard or the influence function.

To be able to apply the predictCox() and iidCox() functions on new classes, one needs to define the methods extracting the necessary information from the new class. For instance, the **mets** package (Holst and Scheike, 2017) contains an efficient routine for estimating Cox models. However "phreg" object that are quite different from "coxph" objects:

```
library(mets, verbose = FALSE)
Melanoma$entry <- 0
f1.phreg <- phreg(Surv(entry, time, status != 0) ~ age + logthick + epicel +
                    strata(sex), data = Melanoma)
list(coxph=names(f1),
     phreg=names(f1.phreg))
```

```
## $coxph
##  [1] "coefficients"      "var"               "loglik"            "score"
##  [5] "iter"              "linear.predictors" "residuals"         "means"
##  [9] "concordance"       "method"            "n"                 "nevent"
## [13] "terms"             "assign"            "wald.test"         "x"
## [17] "strata"            "y"                 "formula"           "xlevels"
## [21] "contrasts"         "call"
##
## $phreg
##  [1] "coef"        "ploglik"     "gradient"    "hessian"     "U"           "S0"
##  [7] "nevent"      "ord"         "time"        "jumps"       "jumptimes"   "strata"
## [13] "entry"       "exit"        "status"      "p"           "X"           "id"
## [19] "opt"         "call"        "model.frame"
```

Therefore to be able to use the predictCox() and iidCox() functions on "phreg" one needs to define methods to extract:

The values used to center the covariates (CoxCenter() method). Instead of working on X, many routines estimating the Cox model parameters works on a centered version $\tilde{X} = X - \bar{X}$. The CoxCenter() method returns $\bar{X}$:

```
riskRegression:::coxCenter.coxph(f1)
```

```
##        age      logthick epicelpresent
## 52.4634146     0.6181706     0.4341463
```

The design matrix used to fit the Cox model (CoxDesign() method). The first two columns describe the beginning and the end of the interval of time when the individual was followed. The third contains the event type (0 corresponding to censoring and 1 to an observed event, e.g. death). The remaining columns contain the design matrix corresponding to the coefficients β of the Cox model and the strata variable (if any):

```
head(riskRegression:::coxDesign.coxph(f1))
```

```
##   start stop status age   logthick epicelpresent strata
## 1     0   10      1  76  1.9110229             1      2
## 2     0   30      1  56 -0.4307829             0      2
## 3     0   35      0  41  0.2926696             0      2
## 4     0   99      1  71  1.0647107             0      1
## 5     0  185      1  52  2.4915512             1      2
## 6     0  204      1  28  1.5769147             0      2
```

The formula of the Cox model (CoxFormula() method):

```
riskRegression:::coxFormula.coxph(f1)
```

```
Surv(time, status != 0) ~ age + logthick + epicel + strata(sex)
```

The value of the linear predictor $X\beta$ (`CoxLP()` method). This function has three arguments: `object`, `data`, and `center`.

```
head(riskRegression:::coxLP.coxph(f1, data = NULL, center = FALSE))
```

```
## [1] 2.433793 1.136828 1.168604 2.332216 2.165533 1.559629
```

When setting `data` to `NULL`, the `CoxLP()` method will return the linear predictor computed on the dataset used to fit the Cox model. The `center` argument indicates whether the covariates should be centered before computing the linear predictor.

The number of observations used to fit the Cox model (`CoxN()` method):

```
riskRegression:::coxN.coxph(f1)
```

```
## [1] 205
```

The character string indicating the strata variable(s) in the formula (`CoxSpecialStrata()` method):

```
riskRegression:::coxSpecialStrata.coxph(f1)
```

```
## [1] "strata"
```

The variable encoding to which strata belongs each observation (`CoxStrata()` method). This variable must be univariate, aggregating all the strata variables.

```
head(riskRegression:::coxStrata.coxph(f1, data = NULL, strata.vars = "strata(sex)"))
```

```
## [1] Male   Male   Male   Female Male   Male
## Levels: Female Male
```

Similarly to `CoxLP()`, when setting `data` to `NULL`, the `CoxStrata()` method will return the strata variable computed on the dataset used to fit the Cox model.

The variance-covariance matrix of the regression coefficients (`CoxVarCov()` method):

```
riskRegression:::coxVarCov.coxph(f1)
```

```
##                          age      logthick epicelpresent
## age            6.553939e-05 -0.0002102408 -0.0004363469
## logthick      -2.102408e-04  0.0206977373  0.0076882093
## epicelpresent -4.363469e-04  0.0076882093  0.0692047486
```

Most of the above methods correspond to a very small piece of code that reformats the information contained in the object, e.g.:

```
riskRegression:::coxVarCov.coxph
```

```
## function (object)
## {
##     Sigma <- object$var
##     if (!is.null(Sigma)) {
##         coefName <- names(coef(object))
##         colnames(Sigma) <- coefName
##         rownames(Sigma) <- coefName
##     }
##     return(Sigma)
## }
## <environment: namespace:riskRegression>
```

We refer to the help page of each method for a more precise description of each method arguments and expected output, as well as examples. Once all of the methods have been defined for a new object (e.g. `"phreg"`), `predictCox()` and `iidCox()` can be applied on the new object:

```
all.equal(predictCox(f1), predictCox(f1.phreg))
```

```
## [1] TRUE
```

```
all.equal(iidCox(f1), iidCox(f1.phreg))
```

```
## [1] TRUE
```

Appendix B Saving the influence functions

The function `iidCox()` computes the estimates of the influence functions ϕ_{β_j}, $\phi_{\lambda_{0j}}$, and $\phi_{\Lambda_{0j}}$ for a given set of covariates and time points:

```
f1.iid <- iidCox(f1)
```

The default implementation (`store.iid="full"`) stores the influence function for the baseline cumulative hazard as a list of matrices, one for each strata. The size of each matrix is the number of observations n times the number of unique event times $n_T(z)$ in stratum z. Storing the influence function can be very memory demanding when considering large datasets. This is why the influence function is only temporary stored during the execution of the `predict()` method.

When dealing with very large datasets, e.g. following $n = 20000$ patients during $n_T = 365$ days, storing the influence function can be too memory demanding. Instead of computing and storing $\phi_{\lambda_{0j}}$, an alternative solution is to only store the necessary quantities to compute $\phi_{\lambda_{0j}}$:

$$\frac{1\{\tilde{T}_i = t, \tilde{D}_i = j\}}{\widehat{\mathcal{S}^{(0)}}(\tilde{T}_i, \hat{\beta}_j, Z_i)},\ E(t, \hat{\beta}_j, z)\hat{\lambda}_{0j,z}(t),\ \int_0^t E(s, \hat{\beta}_j, z)\hat{\lambda}_{0j,z}(s)\mathrm{d}s,\ \frac{\hat{\lambda}_{0j,z}(t)}{\widehat{\mathcal{S}^{(0)}}(t, \hat{\beta}_j, z)},\ \int_0^t \frac{\hat{\lambda}_{0j,z}(s)}{\widehat{\mathcal{S}^{(0)}}(s, \hat{\beta}_j, z)}ds. \quad (16)$$

These quantities are lists containing L vectors of length n or $n_T(z)$. Since in most applications n and $n_T(z)$ are large compared to L this approach is much more memory efficient. Setting the argument `store.iid` to `"minimal"` when calling `iidCox` will return the influence function using this alternative storage method:

```
f2.iid <- iidCox(f1, store.iid = "minimal")
```

We can compare the memory cost of the default implementation vs. the alternative one:

```
size.f1.iid <- object.size(f1.iid$IFcumhazard)
size.f2.iid <- object.size(f2.iid$calcIFhazard)
as.numeric(size.f2.iid/size.f1.iid)
```

```
## [1] 0.08627399
```

and see that the memory use for storing the influence function of the cumulative hazard has been divided by more than 10.

Computing the influence function of the absolute risk with the default implementation only requires to:

- use $\widehat{\phi}_{\Lambda_{0j,z}}(\mathcal{X}_i; t)$ and $d\widehat{\phi}_{\Lambda_{0j,z}}(\mathcal{X}_i; t)$ with formula (12) and (13) to obtain $\hat{\phi}_{\Lambda_{j,z}}(\mathcal{X}_i; t, x)$ and $d\hat{\phi}_{\Lambda_{j,z}}(\mathcal{X}_i; t, x)$.
- use formula (14) to obain $\widehat{\phi}_{F_1}(\mathcal{X}_i, t, x, z)$.

When using the alternative implementation, $\widehat{\phi}_{\Lambda_{0j,z}}(\mathcal{X}_i; t)$ and $d\widehat{\phi}_{\Lambda_{0j,z}}(\mathcal{X}_i; t)$ have not been computed. Therefore an additional step is needed:

- use (16) with formula (10) and (11) to compute $\widehat{\phi}_{\Lambda_{0j,z}}(\mathcal{X}_i; t)$ and $d\widehat{\phi}_{\Lambda_{0j,z}}(\mathcal{X}_i; t)$.

The two implementations will lead to the same influence function and therefore the same confidence intervals or confidence bands.

The alternative implementation is performed iterating over the set of covariates used to make the predictions, avoiding to store the influence function of the baseline hazard for all event times and strata. It will also not compute the influence function at unnecessary times and strata. Thus it should always be more memory efficient and, when asking for a single prediction, it should also be have a lower computation time. However, compared to the default implementation where $\widehat{\phi}_{\Lambda_{0j,z}}(\mathcal{X}_i; t)$ and $d\widehat{\phi}_{\Lambda_{0j,z}}(\mathcal{X}_i; t)$ are only computed once, the alternative implementation recomputes these quantities for each prediction.

Therefore when the prediction is to be made for many different sets of covariates (i.e. new patients) this may lead to a substantial increase in computation time. See the subsection Runtime and memory usage for more details.

To use the implementation for computing the standard errors, confidence intervals, or confidence bands one must set the argument `store.iid` to `"minimal"`:

```
pfit3se <- predict(cfit1, newdata = newdata, cause = 1, times = c(867,3500),
                   se = TRUE, store.iid = "minimal", keep.newdata = FALSE)
print(pfit3se)
```

```
##    observation times     strata absRisk absRisk.se absRisk.lower absRisk.upper
## 1:           1   867 sex=Female   0.021      0.122       0.00738        0.0478
## 2:           2   867   sex=Male   0.149      0.161       0.07356        0.2502
## 3:           1  3500 sex=Female   0.117      0.151       0.05552        0.2025
## 4:           2  3500   sex=Male   0.428      0.320       0.20416        0.6355
```

```
range(pfit3se$absRisk.se-pfit1se$absRisk.se)
range(pfit3se$absRisk.upper-pfit1se$absRisk.upper)
range(pfit3se$absRisk.lower-pfit1se$absRisk.lower)
```

```
## [1] -2.775558e-17  5.551115e-17
## [1] -5.551115e-17  0.000000e+00
## [1] -5.551115e-17  0.000000e+00
```

This option also applies when computing standard errors, confidence intervals, or confidence bands for the survival probabilities:

```
p2 <- predictCox(f1, newdata = Melanoma[3:5,],
                 times = c(Melanoma$time[5:7],1000),
                 se = TRUE, store.iid = "minimal", type="survival")
p2 <- as.data.table(p2)
p2[times == 185, survival.lower]-p1[times == 185, survival.lower]
p2[times == 185, survival.upper]-p1[times == 185, survival.upper]
```

```
## [1]  0.000000e+00  1.110223e-16 -1.110223e-16
## [1] 0 0 0
```

LeArEst: Length and Area Estimation from Data Measured with Additive Error

by Mirta Benšić, Petar Taler, Safet Hamedović, Emmanuel Karlo Nyarko and Kristian Sabo

Abstract This paper describes an R package **LeArEst** that can be used for estimating object dimensions from a noisy image. The package is based on a simple parametric model for data that are drawn from uniform distribution contaminated by an additive error. Our package is able to estimate the length of the object of interest on a given straight line that intersects it, as well as to estimate the object area when it is elliptically shaped. The input data may be a numerical vector or an image in JPEG format. In this paper, background statistical models and methods for the package are summarized, and the algorithms and key functions implemented are described. Also, examples that demonstrate its usage are provided.

Availability: LeArEst is available on CRAN.

Introduction

Image noise may arise by the physical processes of imaging, or it can be caused by the presence of some unwanted structures (e.g., soft tissue captured in X-ray images of bones). Such problems can occur, for example, when the object is observed with a fluorescent microscope (Ruzin and others, 1999), ground penetrating radar, medical equipment (X-ray, ultrasound), etc. With the presence of additive noise, the detection of the object edge as well as determining length or area of the object becomes a non-trivial problem. The well known edge detection methods (Qiu, 2005; Canny, 1986) generally do not perform well.

Our approach does not use the mentioned edge detection methods, but looks at the problem in a different way. We start with a simple univariate model where the data represent independent realizations of a random variable X, $X = U + \varepsilon$. In the aforementioned equation U is supposed to be uniformly distributed over the object image and describes the object image without an error, while ε represents measurement error. It is shown that such a simple model can also be very useful in applications itself, not only related to the image analysis. For instance, in Tolić et al. (2017) the sum of uniform and normal distributions is confirmed to be the most representative distribution for modelling transmission loss data.

Different aspects of this model are developed in Benšić and Sabo (2007b), Benšić and Sabo (2007a), Benšić and Sabo (2010), Benšić and Sabo (2016), Sabo and Benšić (2009), and Schneeweiss (2004). The basic one-dimensional model is described in Section B.2 together with the results that are used for statistical inference incorporated in the package. Although this model is not universal in all applications, we find it useful in some cases.

With the assumption that the observed object has a circular or elliptical shape, a two-dimensional approach has been developed, dealing with an object area estimation problem (Benšić and Sabo, 2007a; Sabo and Benšić, 2009). This approach utilizes many border estimations and performs parametric curve fitting on its results.

The package **LeArEst** (Bensic et al., 2017) uses these methods for length and area estimation of an object captured with noise. It supports numerical inputs, which is useful if a machine that records an object stores numerical data (coordinates of recorded points). However, if an object is captured in a picture file, the package includes a web interface with which one can load a picture, specify a line that intersects the object, adjust the parameters, and perform an edge detection on the drawn line. Another web interface allows the user to draw a rectangle around the object and perform area estimation of the marked object. Description of functions dealing with numerical and graphical estimations and examples of their use are given in Section B.3.

Basic statistical model

The basic model we deal with in this package is an additive error model

$$X = U + \varepsilon.$$

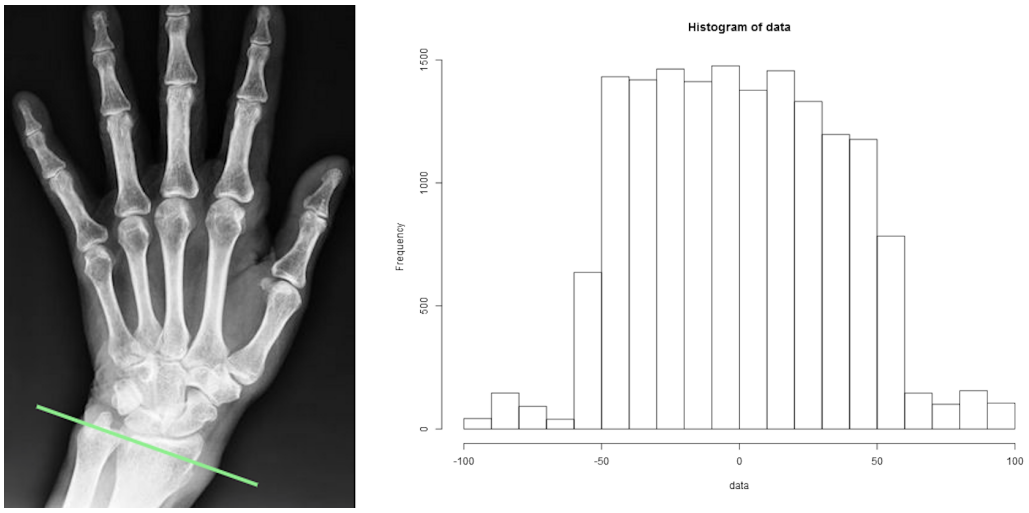


Figure 1: Line intersecting the object and histogram of recorded points for statistical inferences

Here we suppose that the random variable U is uniformly distributed on the interval $[-a, a]$, $a > 0$, i.e., has a density

$$f_U(x; a) = \begin{cases} \frac{1}{2a}, & x \in [-a, a]. \\ 0, & \text{else} \end{cases} \tag{1}$$

and ε is an absolutely continuous random variable describing measurement error. Further, we assume that ε is independent of U with zero mean. Instead of the sample $U_1, U_2, \ldots, U_n$ from U, one can only observe the contaminated i.i.d. sample $X_1, X_2, \ldots, X_n$ from X. We are to estimate the unknown parameter a.

Our model is the special case of a general additive error model $X = Y + \varepsilon$, where Y and ε are assumed to be independent continuous random variables, but only X is observable. Estimating the unknown density f_Y from an i.i.d. sample $X_1, X_2, \ldots, X_n, X_1 \sim X$, is known as the deconvolution problem. Usually, the error part ε is assumed to have a known density f_ε. Several nonparametric methods have been developed to estimate f_Y (Meister, 2009); the most popular and studied is the deconvolution kernel density estimator (Carroll and Hall, 1988; Stefanski and Carroll, 1990). The packages **decon** (Wang and Wang, 2013) and **deamer** (Stirnemann et al., 2012) provide functions for estimating density f_Y in a nonparametric way. Two different approaches in estimating the support of a density from a contaminated sample can be seen in (Meister, 2006) and (Delaigle and Gijbels, 2006). One is based on the deconvolving kernel density estimator (Delaigle and Gijbels, 2006) and the other on the moment estimation (Meister, 2006). To our knowledge, none of them is implemented in some R package submitted to the CRAN repository.

For our purpose (e.g., estimating borders of some object from a noisy image), we find that the model with $Y \sim \mathcal{U}[-a, a]$ is useful in some instances. Namely, in many cases we have a relatively high contrast between an object and its background as it is the case in Figure 5. It seems reasonable to assume that the data extracted from the green line in Figure 5 come from a uniform distribution, but contaminated by an additive error.

Let f_ε and F_ε be the density and distribution function of the error part ε. Then the density of $X = U + \varepsilon$ is

$$f_X(x; a) = \int_{-\infty}^{\infty} f_U(t) f_\varepsilon(x - t) dt = \frac{1}{2a} \left(F_\varepsilon(x + a) - F_\varepsilon(x - a) \right). \tag{2}$$

If we suppose that the distribution of ε belongs to a scale family, with scale parameter σ, then (2) may be rewritten as

$$f_X(x; a, \sigma) = \frac{1}{2a} \left(F\left(\frac{x + a}{\sigma} \right) - F\left(\frac{x - a}{\sigma} \right) \right),$$

with $F(x)$ being the standard ($\sigma = 1$ and zero mean) distribution function.

Let $\mathbf{x} = (x_1, \ldots, x_n)$ denote the realization of the i.i.d. sample $\mathbf{X} = (X_1, \ldots, X_n)$. The likelihood function has the form

$$L(a, \sigma; \mathbf{x}) = \prod_{i=1}^{n} f_X(x_i; a, \sigma) = \frac{1}{(2a)^n} \prod_{i=1}^{n} \left(F\left(\frac{x_i + a}{\sigma} \right) - F\left(\frac{x_i - a}{\sigma} \right) \right),$$

and the log-likelihood function is given by

$$l(a,\sigma) = -n\log(2a) + \sum_{i=1}^{n} \log\left(F\left(\frac{x_i+a}{\sigma}\right) - F\left(\frac{x_i-a}{\sigma}\right)\right).$$

If the distribution of ε is symmetric around zero, then the Fisher information is

$$I(a) = \frac{-1}{a^2} + \frac{1}{a\sigma^2}\int_0^\infty \frac{\left(f\left(\frac{x+a}{\sigma}\right) + f\left(\frac{x-a}{\sigma}\right)\right)^2}{F\left(\frac{x+a}{\sigma}\right) - F\left(\frac{x-a}{\sigma}\right)}\,dx.$$

Supposing that parameter σ is known or consistently estimated then, under regularity, we have

$$\sqrt{n}\left(\hat{a}_{ML} - a_0\right) \xrightarrow{D} \mathcal{N}\left(0, \frac{1}{I(a_0)}\right), \tag{3}$$

where a_0 is the true value of a.

Some flexibility of our model is achieved by changing the error distribution. For now, three types of error distributions are available in the package. The normal distribution $\varepsilon \sim \mathcal{N}\left(0,\sigma^2\right)$ is sometimes a natural choice. Properties of maximum likelihood (ML) and method of moments (MM) estimators with known σ^2 are given in Benšić and Sabo (2007b). The model with $Y \sim \mathcal{U}[0,a]$ is treated in Schneeweiss (2004). Benšić and Sabo (2010) considered the unknown σ^2 case. The possibility of this (one-dimensional) model in two-dimensional problems is given in Benšić and Sabo (2007a) and Sabo and Benšić (2009). For the sake of robustness Laplace and scaled Student (with 5 degrees of freedom) distributions are also incorporated in the package as a choice of the error distribution. Estimating issues with Laplacian error can be seen in Benšić and Sabo (2016), as well as a discussion connected to robustness.

Two procedures for deriving confidence intervals for a are described in (Hamedović et al., 2017). The first one is based on the asymptotic distribution of ML estimator in (3). For a specified $0 < \alpha < 1$, an asymptotic $(1-\alpha)100\%$ confidence interval for a is[1]

$$\left(\hat{a}_{ML} - \frac{z_{\alpha/2}}{\sqrt{nI(\hat{a}_{ML})}}, \hat{a}_{ML} + \frac{z_{\alpha/2}}{\sqrt{nI(\hat{a}_{ML})}}\right).$$

The second method is based on the likelihood ratio statistic

$$\lambda(\mathbf{X}) = \frac{\sup\limits_{H_0} L\left(a;\mathbf{X}\right)}{\sup\limits_{(0,\infty)} L\left(a;\mathbf{X}\right)} = \frac{L\left(a_0;\mathbf{X}\right)}{L\left(\hat{a}_{ML};\mathbf{X}\right)}.$$

From the asymptotic distribution of the log-likelihood ratio statistic

$$-2\log\lambda(\mathbf{X}) \xrightarrow{D} \chi_1^2$$

an approximate $(1-\alpha)100\%$ confidence interval for a is

$$\left\{a|\, l(\hat{a}_{ML}) - l(a) \le 0.5\chi_1^2(1-\alpha)\right\},$$

where $\chi_1^2(1-\alpha)$ is the $1-\alpha$ quantile of χ_1^2 distribution.

These two approaches can be used to test hypotheses regarding the parameter a. For example, in the case of a two-sided hypotheses $H_0 : a = a_0$ versus $H_1 : a \neq a_0$, the critical regions of asymptotic size α are

$$\left\{\mathbf{x}|\, \sqrt{nI(a_0)}|\hat{a}_{ML} - a_0| \ge z_{\alpha/2}\right\}, \text{ and}$$

$$\left\{\mathbf{x}|\, -2\log\lambda(\mathbf{x}) \ge \chi_1^2(1-\alpha)\right\},$$

respectively. Note that both methods are asymptotically equivalent.

Overview of the package

The package **LeArEst** depends on the following packages that should installed in addition to the **LeArEst** package: **conicfit** (Gama and Chernov, 2015), **jpeg** (Urbanek, 2014), and **opencpu** (Ooms,

[1] as usual, z_α is the $1-\alpha$ quantile of the standard normal distribution

Function	Description
`lengthest()`	Performs length estimation from a numerical data set.
`lengthtest()`	Performs one-sided and two-sided tests for uniform distribution half-length.
`areaest()`	Performs area estimation of a numerically described object in plane.
`startweb.esttest()`	Opens default web browser and loads a web page for length estimation and testing (the object of interest is shown in an image).
`startweb.area()`	Opens default web browser and loads a web page for area estimation of the object shown in an image.

Table 1: Overview of **LeArEst** functions

Arguments	Description
`x`	Vector of input data.
`error`	Error distribution.
`var`	Error variance.
`var.est`	Method of error variance estimation.
`conf.level`	Confidence level of the confidence interval. Defaults to 0.95.
Results	**Description**
`radius`	Estimated half-length of the uniform support.
`var.error`	Error variance, estimated or explicitly given by argument `var`.
`conf.int`	Confidence interval for half-length of the uniform support.
`method`	Method used for computing a confidence interval (asymptotic distribution of ML or likelihood ratio statistic).

Table 2: Summary of arguments and results of `lengthest`

2014). The stable version of the package is available on the Comprehensive R Archive Network repository (CRAN; https://CRAN.R-project.org/) and can be downloaded and installed by issuing the following command at the R console:

```
> install.packages("LeArEst")
```

The package is loaded using the following command:

```
> library(LeArEst)
```

An overview of the package's functions is given in Table 1.

Length estimation — a numerical data set

The function `lengthest` computes the length of an interval which is the domain of a uniform distribution from data contaminated by an additive error according to the model described in the previous section. The function's arguments and results are given in Table 2.

In order to perform length estimation, a type of the error distribution must be chosen through the argument `error` with three possibilities: '`laplace`' (Benšić and Sabo, 2016), '`gauss`' (Benšić and Sabo, 2007b, 2010), or '`student`' (scaled Student distribution with 5 degrees of freedom).

The variance of the additive error may or may not be known. If the variance is known, argument `var` should be used and the variance should be assigned to it. In the case of unknown variance, function `lengthest` implements two methods for its estimation: *Method of Moments* and *Maximum Likelihood*. Value '`MM`' of the argument `var.est` instructs the functions to use Method of Moments, while the corresponding value '`ML`' triggers Maximum Likelihood Method. There is the possibility, depending on the input data, that the Method of Moments estimate of error variance does not exist. When that is the case, the function stops and outputs the message instructing the user to use Maximum Likelihood estimator or to give an explicit variance. It is important to mention that arguments `var` and `var.est` may not be used simultaneously.

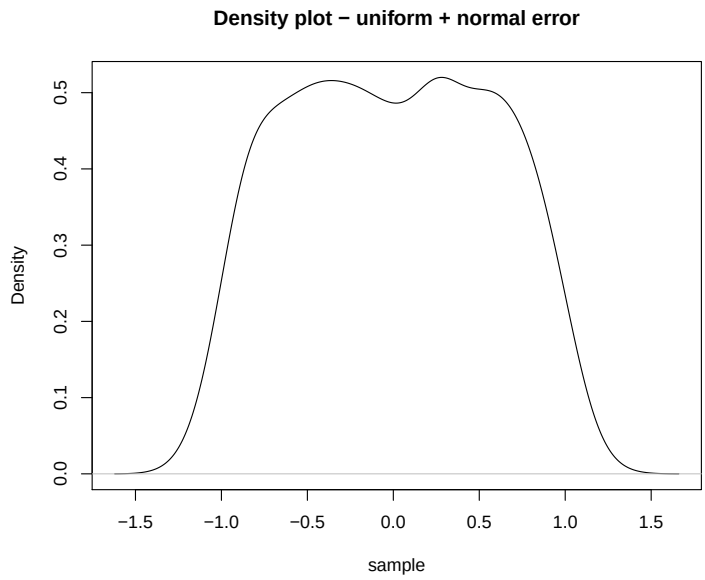


Figure 2: Estimation of the density function

The last argument, `conf.level`, specifies the confidence level of the confidence interval calculated by the function.

The results of this function are the estimated half-length of uniform distribution (i.e., of an object), estimated or explicitly given error variance, confidence interval for half-length (with regard to the given confidence level) and the statistical method for computing a confidence interval.

Usage example. Let us generate a sample of size 1000 from $X = U + \varepsilon$, where $U \sim \mathcal{U}[-1, 1]$ and $\varepsilon \sim \mathcal{N}\left(0, (0.1)^2\right)$:

```
set.seed(12)
sample_1 <- runif(1000, -1, 1)
sample_2 <- rnorm(1000, 0, 0.1)
sample <- sample_1 + sample_2
```

Figure 2 shows density estimation from the generated data obtained with the R function `density`. A half-length estimation of the uniform support for these data can be done with the following command:

```
lengthest(x = sample, error = "gauss", var.est = "MM", conf.level = 0.90)
```

The most important part of its output is:

```
$radius
MLE for radius (a) of uniform distr.: 0.9916513
$var.error
MM estimate for error variance: 0.01279636
$method
[1] "Asymptotic distribution of LR statistic"
$conf.int
[1] 0.9724316 1.0116479
```

Testing hypothesis — a numerical data set

Function `lengthtest` performs one-sided and two-sided tests against hypothesized half-length of the uniform support as it is described in Section B.2. Since the actual calculations inside this function are based on the ML approach most input arguments are similar to those in the function `lengthest` (see Table 3). Argument `null.a` is a positive number representing hypothesized half-length of the uniform support, while argument `alternative` defines the usual forms of alternatives ('`two.sided`', '`greater`', or '`less`').

Function `lengthtest` also performs length estimation, so all values from its output, except `p.value` and the calculated value of the test statistic (`tstat`), are the same as that of the function `lengthest`.

Arguments	Description
x	Vector of input data.
error	Error distribution.
null.a	Specified null value being tested.
alternative	The form of the alternative hypothesis.
var	Error variance.
var.est	Method of error variance estimation.
conf.level	Confidence level of the confidence interval. Defaults to 0.95.
Results	**Description**
p.value	*p-value* of the test.
tstat	The value of the test statistic.
radius	Estimated half-length of the uniform support.
var.error	Error variance, estimated or explicitly given by argument var.
conf.int	Confidence interval for half-length.
method	Method used for computing a confidence interval (asymptotic distribution of ML or likelihood ratio statistic).

Table 3: Summary of arguments and results of lengthtest

Usage example. Generate the data in a similar manner as in the lengthest example:

```
set.seed(12)
sample_1 <- runif(1000, -1, 1)
sample_2 <- rnorm(1000, 0, 0.1)
sample <- sample_1 + sample_2
```

To test that the uniform support half-length equals 1 against that it is less than 1 the function lengthest can be used in the following way:

```
lengthtest(x = sample, error = "gauss", alternative = "less", var.est = "MM",
           null.a = 1, conf.level = 0.95)
```

Part of the output dealing with a testing procedure is:

```
$p.value
[1] 0.2418929
$tstat
[1] -0.7002265
```

Area estimation — a numerical data set

The input for the function areaest is supposed to be a data set of points in the plane representing independent realizations of a two-dimensional random vector

$$X = U + \varepsilon.$$

It is assumed that U has a uniform distribution on an ellipsoid and ε is a two-dimensional error term independent of U. Arguments and results of this function are listed in Table 4.

The algorithm implemented in the function areaest is explained in detail in Benšić and Sabo (2007a). The main task in area estimation is to estimate edge points of the uniform support. In order to achieve this, the original problem is reduced to several corresponding one-dimensional problems, which can in turn be solved by function lengthest.

Let us denote the data set with $D = \{(x_i, y_i), i = 1, \ldots, n\}$. The function areast transforms this data set in two different ways: through the y-axis and through the x-axis. The algorithm for transformation through the y-axis is presented below, while the transformation through the x-axis is done analogously.

ALGORITHM 1 (*Transformation through the y-axis (Benšić and Sabo, 2007a)*)

Step 1. Separating through the y-axis.

Choose an integer $m < n$ and real numbers $\eta_1 < \eta_2 < \cdots < \eta_m$ such that

Arguments	Description
`data`	Two-column data matrix containing the points that describe the observed object. The first column represents the x coordinate of the point, while the second column represents its y coordinate.
`nrSlices`	Number of slices applied for plain data cutting. Defaults to 10.
`error`	Error distribution.
`var`	Error variance.
`var.est`	Method of error variance estimation.
`plot`	Logical parameter that determines whether to plot data set, calculated edge points, and the resulting ellipse. Defaults to `FALSE`.
Results	**Description**
`area`	Estimated area of the object.
`points`	Set of estimated object's edge points.
`semiaxes`	Resulting ellipse's semi-axes.

Table 4: Summary of arguments and results of `areaest`

(i) $\eta_1 \leq \min\{y_i : i = 1, \dots n\}$, $\max\{y_i : i = 1, \dots n\} \leq \eta_m$ and
(ii) $C_k := \{(x_i, y_i) \in D : y_i \in [\eta_k, \eta_{k+1}]\}$, $k = 1, \dots, m-1$ is a nonempty set.

Step 2. Centering through the y-axis.

Let us denote

$$c_k := \frac{1}{|C_k|} \sum_{(x_i,y_i)\in C_k} x_i,\ d_k := \frac{1}{|C_k|} \sum_{(x_i,y_i)\in C_k} y_i,$$
$$k = 1, \dots, m-1,$$

for $k = 1, \dots, m-1$ define $\overline{C}_k := \{x_i - c_k : (x_i, y_i) \in C_k\}$.

Using this algorithm the data are transformed in the way that we have sets $\overline{C}_k$, $k = 1, \dots, m-1$ that represent centered tiny strips. Argument `nrSlices` corresponds to $m-1$ and specifies the number of strips. The lengths of these strips (in x-direction) can be estimated using the function `lengthest` (the parameters `error`, `var`, and `var.est` are used in a `lengthest` call in the way described earlier). After doing so, the algorithm needs to be repeated through the x-axis. Finally, at this point of the procedure, the data that is a noisy version of points from the curve is created – it represents estimated points from the border of the object.

The next task is to choose one of the well-known curve fitting procedures for parameter estimation. Here we are dealing with a nonlinear parameter estimation problem.

Let us suppose that we have an elliptical domain, i.e.,

$$\mathcal{D}(\mathbf{p}) = \left\{ (x, y) \in \mathbb{R}^2 : \frac{(x-p)^2}{\alpha^2} + \frac{(y-q)^2}{\beta^2} \leq 1 \right\},$$
$$\mathbf{p} = (p, q, \alpha, \beta)^T.$$

On the basis of data obtained so far, the vector of unknown parameters p needs to be estimated, and by doing so, the optimal ellipse that fits into our points is to be defined. For this purpose `EllipseDirectFit` function from the **conicfit** package is used. This function implemets the algebraic ellipse fit method by Fitzgibbon-Pilu-Fisher (Fitzgibbon et al., 1999). Having parameters p, it is a trivial task to calculate the area of the ellipse that approximates the observed object.

Usage example. Two internal files are provided with the package: 'ellipse_3_4_0.1_gauss.txt' and 'ellipse_3_4_0.1_laplace.txt'. Both of them represent an ellipsoidal object with center in point $(1, 1)$, half-axes 1.5 and 2, with added measurement error. In the first file, the error distribution is a two-dimensional normal with independent margins and variance 0.01, while in the other it is Laplacian ($\lambda = 0.1$) in both directions, again with independent margins.

In order to use one of these files, the data needs to be read into a data frame:

```
inputfile <- system.file("extdata", "ellipse_3_4_0.1_laplace.txt", package = "LeArEst")
inputdata <- read.table(inputfile)
```

Area estimation of the uniform support can be done with the command:

```
areaest(inputdata, error = "laplace", var.est = "ML", nrSlices = 5, plot = TRUE)
```

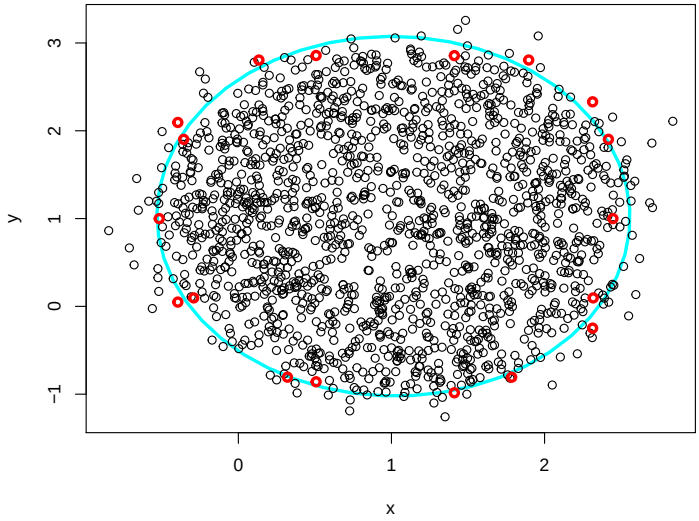


Figure 3: Data points, estimated border points, and the resulting ellipse obtained by the function `areaest`

In the previous example, the parameter `plot` is set to `TRUE`, so the function plots the given input data (black dots), estimated border points (red dots), and the resulting ellipse (cyan ellipse); see Figure 3.

The most important parts of the numerical output are:

```
$area
[1] 9.938305
$semiaxes
[1] 2.048028 1.544638
```

Length estimation and testing for an object shown in a picture

In order to apply the described methods to a picture of an object, two web interfaces have been built and embedded into the package.

As far as we know, **shiny** (Chang et al., 2017) provides the simplest way of building web applications using R. However, limitations of its free version discouraged us from using it, so we decided to use the **opencpu** package. This package provides a reliable and interoperable HTTP API for data analysis based on R. Basically, it provides an interface between functions in R package and a custom-made web page bundled with the package, using JavaScript and AJAX. Building web interfaces using **opencpu** is more complex than using **shiny**, but at the same time, it provides more flexibility in application design. It is assumed that developers are familiar with HTTP protocol, HTML, and the JavaScript language, in order to develop such web applications.

Function `startweb.esttest` will be described in this section. This function takes no arguments and returns no results, its task is to start a web interface for length estimation and hypothesis testing (Figure 4).

To start the analysis using the web interface the picture in JPG format should be loaded (*Load Picture* button). Then, a line should be drawn that intersects the object of interest by clicking on two points in the picture - length estimation will be performed on that line. Finally, parameters for a data set preparation should be set.

The `Levels of gray` parameter determines how many levels of grey the algorithm should take into account. It is important to mention that, although color images can be loaded, they are internally converted to grey-scales prior to any calculations. Since JPG format supports 2^{24} different colors, the number of possible colors should be reduced in order to optimize estimation speed and memory consumption.

`Line thickness` specifies how many picture pixels around the drawn line are taken into account in length estimation. For instance, if `Line thickness` is set to 3, the algorithm takes pixels which are

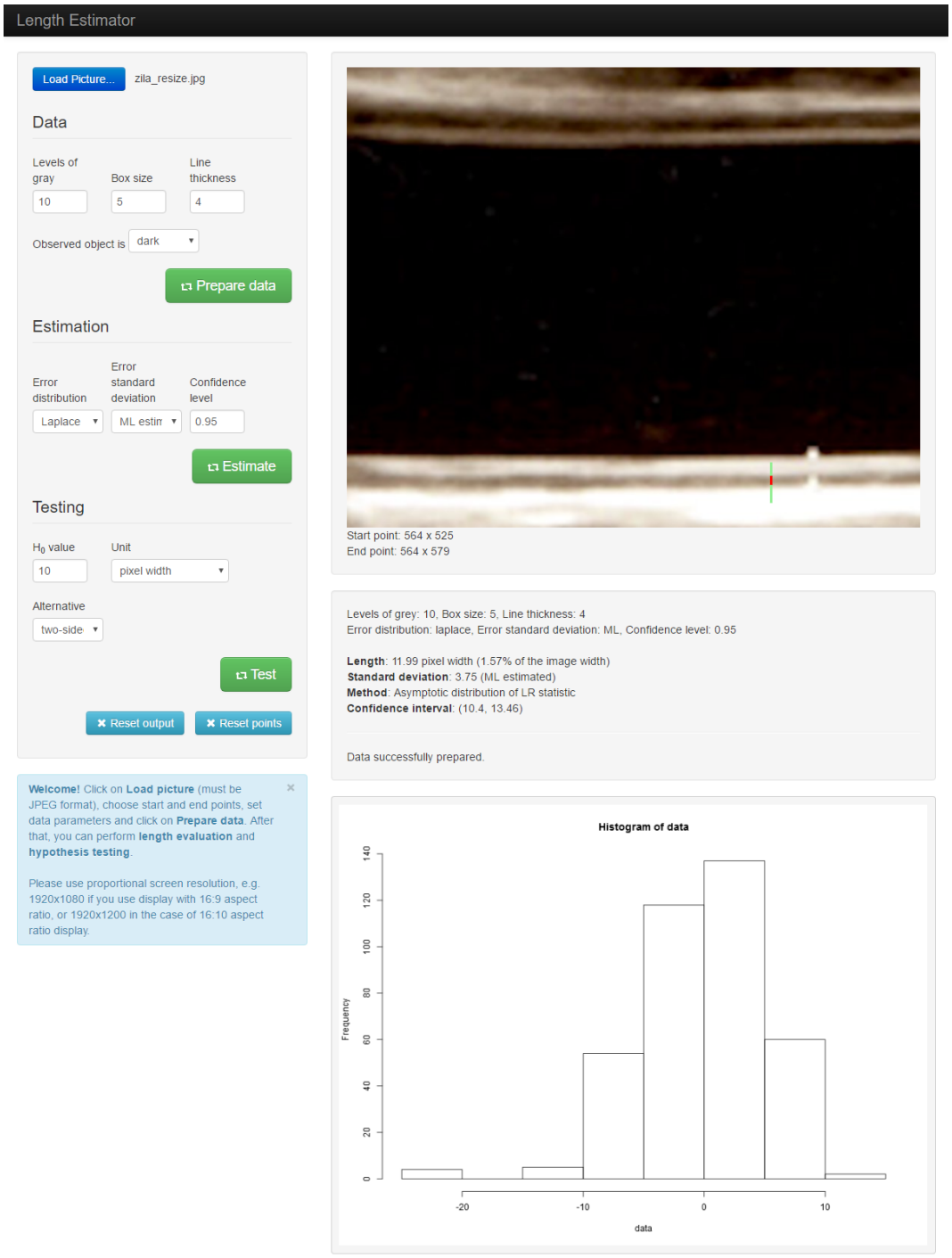


Figure 4: Web interface for length estimation and hypothesis testing. Loaded image shows arterial wall of the carotid artery; we are trying to estimate its intima media thickness (darker layer below artery cavity).

direct left neighbours of the line, pixels on the line itself, and the ones which are direct right neighbours of the line. By doing so, the matrix of (length of the line) $\times$ `Line thickness` pixels – *PixelMatrix* is obtained.

By doing so, we have obtained the matrix of (length of the line) $\times$ `Line thickness` pixels – *PixelMatrix*.

By clicking on the *Prepare data* button the data set will be prepared for the inference.

The following step deals with data preparation and is a crucial step of the algorithm. Each pixel of *PixelMatrix* is mapped to a new matrix of `Box size` $\times$ `Box size` booleans – *DotMatrix* (note that `Box size` is a parameter). Further, every *DotMatrix* is filled with uniformly distributed dots (i.e., `TRUE`

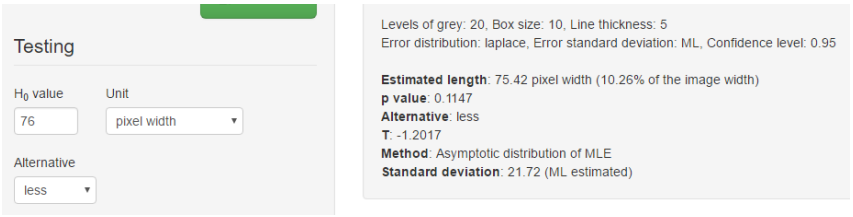


Figure 5: Web interface for length estimation and hypothesis testing - hypothesis testing output

values) in a way that total number of dots in each *DotMatrix* corresponds to the brightness of the pixel it represents. Then, *DotMatrices* are tiled up with respect to the position of corresponding pixel and, by doing so, a new matrix of (length of the line · `Box size`) × (`Line thickness` · `Box size`) booleans is obtained – *FinalDotMatrix*. The last step is to summarize rows of the *FinalDotMatrix* to obtain a vector of (length of the line) · `Box size` integers. The vector's histogram is shown on a web interface (Figure 4, at the bottom) and the vector itself serves as an input to the functions `lengthest` or `lengthtest`.

Parameters in the *Estimation* section of the web interface are transferred to `lengthest`, as well. After the user clicks on *Estimate* button, `lengthest` is executed, and its results are displayed below the picture. Additionally, the estimated uniform support is marked red on the intersecting line.

Estimated length is expressed in width of a pixel and in percentage of whole image's width as well. As stated in the info box, it is important to use a proportional screen resolution on user's display, so the pixels on the screen are square-shaped.

The *Testing* section of this web interface serves for hypothesis testing. Procedures related to image loading, choosing an intersecting line, and data preparation are the same as described above. For the purpose of testing, values for `H0`, `unit`, and `alternative` ('`greater`', '`less`', or '`two-sided`') need to be specified. The part of the web interface dealing with output of hypothesis testing procedure is shown in Figure 5.

Area estimation of an object shown in a picture

The function `startweb.area` starts a web interface for area estimation (Figure 6). Again, the first step is to load an image. To select an object whose surface needs to be evaluated, a rectangle should be drawn around it. It is done by clicking on its upper-left and lower-right corners, after which a green rectangle is drawn on the picture.

Data parameters are similar to ones in the length estimation web interface, with the exception of `number of slices`.

The first step in the area estimation algorithm for this function is to roughly isolate the object in the selected rectangle. In order to do that, pixels from the selected rectangle are divided into two clusters by using the `kmeans` function from base-R **stats** package (the criterion for clustering is pixel brightness). Further, only pixels from the 'object cluster' are observed and divided into horizontal and vertical stripes, as described earlier in Algorithm B.3.3. The number of stripes is dictated by the `number of slices` parameter. A length estimation procedure is conducted on each stripe, obtaining two estimated edge points of the object for each stripe (red dots in Figure 5). Two parameters in the *Estimation* section of the web interface are related to the length estimation procedure of the stripes.

Finally, an optimal ellipse that fits into edge points is found using `EllipseDirectFit` function from the **conicfit** package, as well as in the `areaest` function described earlier. The resulting ellipse is drawn in red in Figure 5. Its area is printed below, this is measured in pixels and the percentage of the whole image area.

Concluding remarks

The R package **LeArEst** provides routines for estimating the support of the random variable U, $U \sim \mathcal{U}[-a, a]$, based on a sample from $X = U + \varepsilon$. The random variable ε represents additive measurement error and is supposed to have either a normal, Laplace, or scaled Student distribution with 5 degrees of freedom. The package also includes functions for estimating either the borders or the area of some object from a noisy image. The package may be useful for this purpose mainly in the case of images with reasonable contrast between the object of interest and background. For greater robustness, we find it convenient to use some error distributions with heavier tails. Sometimes we have different amounts of noise in the tails, so it would be useful to include some asymmetric error distributions as well. These are some features we are going to add in the package in order to improve flexibility of our

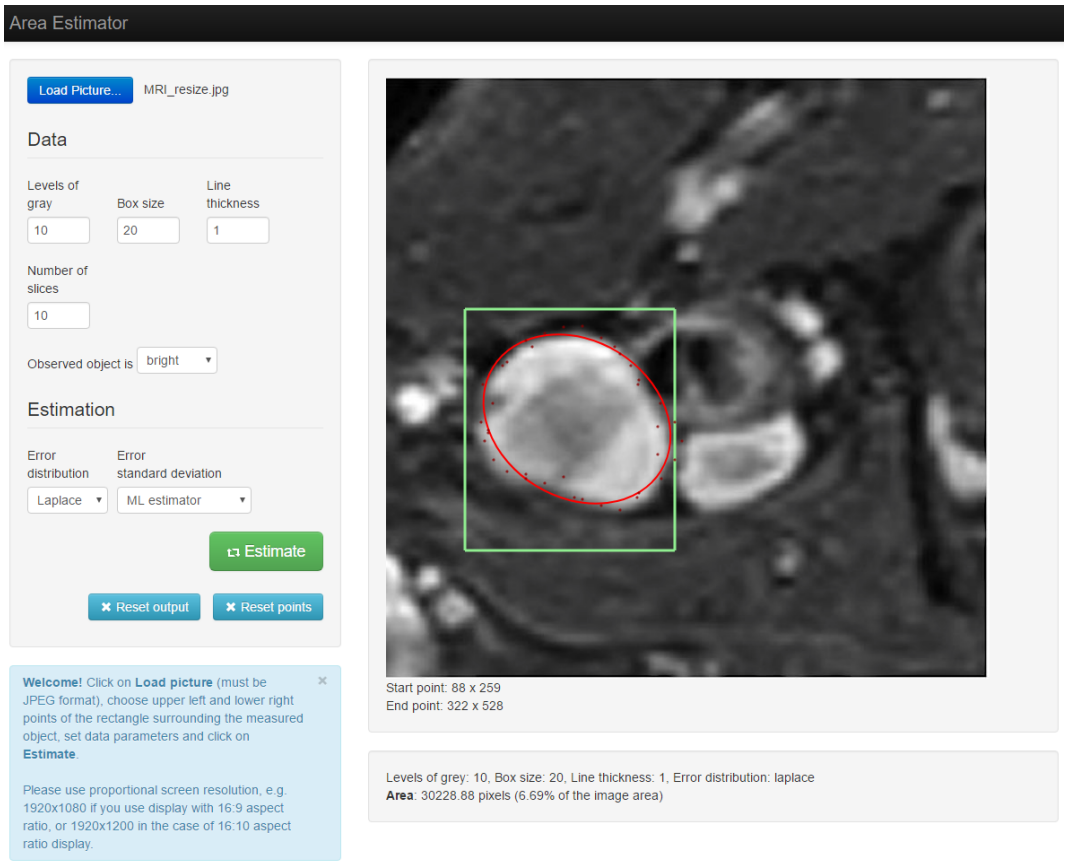


Figure 6: Web interface for area estimation showing an MRI scan detail (taken from Bankman (2008))

models.

Acknowledgement

This work was supported by the Croatian Science Foundation through research grant IP-2016-06-6545. We would like to thank Krunoslav Buljan from Osijek Clinical Hospital Center for providing an image of the carotid artery.

Bibliography

I. Bankman. *Handbook of Medical Image Processing and Analysis*. Academic Press series in biomedical engineering. Elsevier Science, 2008. ISBN 9780080559148. [p471]

M. Bensic, S. Hamedovic, K. Sabo, and P. Taler. *LeArEst: Border and Area Estimation of Data Measured with Additive Error*, 2017. R package version 0.1. [p461]

M. Benšić and K. Sabo. Border estimation of a two-dimensional uniform distribution if data are measured with additive error. *Statistics*, 41(4):311–319, 2007a. [p461, 463, 466]

M. Benšić and K. Sabo. Estimating the width of a uniform distribution when data are measured with additive normal errors with known variance. *Computational statistics & data analysis*, 51(9):4731–4741, 2007b. [p461, 463, 464]

M. Benšić and K. Sabo. Estimating a uniform distribution when data are measured with a normal additive error with unknown variance. *Statistics*, 44(3):235–246, 2010. [p461, 463, 464]

M. Benšić and K. Sabo. Uniform distribution width estimation from data observed with Laplace additive error. *Journal of the Korean Statistical Society*, 45(4):505–517, 2016. [p461, 463, 464]

J. Canny. A computational approach to edge detection. *IEEE Transactions on pattern analysis and machine intelligence*, PAMI-8(6):679–698, 1986. [p461]

R. J. Carroll and P. Hall. Optimal rates of convergence for deconvoluting a density. *J. Amer. Statist. Assoc.*, 83:1184–1186, 1988. [p462]

W. Chang, J. Cheng, J. Allaire, Y. Xie, and J. McPherson. *Shiny: Web Application Framework for R*, 2017. URL https://CRAN.R-project.org/package=shiny. R package version 1.0.3. [p468]

A. Delaigle and I. Gijbels. Estimation of boundary and discontinuity points in deconvolution problems. *Statistica Sinica*, 16:773–788, 2006. [p462]

A. Fitzgibbon, M. Pilu, and R. B. Fisher. Direct least square fitting of ellipses. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(5):476–480, 1999. [p467]

J. Gama and N. Chernov. *Conicfit: Algorithms for Fitting Circles, Ellipses and Conics Based on the Work by Prof. Nikolai Chernov*, 2015. URL https://CRAN.R-project.org/package=conicfit. R package version 1.0.4. [p463]

S. Hamedović, M. Benšić, K. Sabo, and P. Taler. Estimating the size of an object captured with error. Technical report, Department of Mathematics, University of Osijek, 2017. URL http://www.mathos.unios.hr/images/homepages/ksabo/hambensabtal.pdf. Submitted to Central European Journal of Operations Research. [p463]

A. Meister. Support estimation via moment estimation in presence of noise. *Statistics*, 40:259–275, 2006. [p462]

A. Meister. *Deconvolution Problems in Nonparametric Statistics*. Springer-Verlag, 2009. [p462]

J. Ooms. The opencpu system: Towards a universal interface for scientific computing through separation of concerns. *arXiv:1406.4806 [stat.CO]*, 2014. URL http://arxiv.org/abs/1406.4806. [p463]

P. Qiu. *Image Processing and Jump Regression Analysis*, volume 599. John Wiley & Sons, 2005. [p461]

S. E. Ruzin and others. *Plant Microtechnique and Microscopy*, volume 198. Oxford University Press New York, 1999. [p461]

K. Sabo and M. Benšić. Border estimation of a disc observed with random errors solved in two steps. *Journal of computational and applied mathematics*, 229(1):16–26, 2009. [p461, 463]

H. Schneeweiss. Estimating the endpoint of a uniform distribution under measurement errors. *Central European Journal of Operations Research*, 12(2), 2004. [p461, 463]

L. Stefanski and R. J. Carroll. Deconvoluting kernel density estimators. *Statistics*, 21:169–184, 1990. [p462]

J. Stirnemann, A. Samson, F. Comte, and C. Lacour. *Deamer: Deconvolution Density Estimation with Adaptive Methods for a Variable Prone to Measurement Error*, 2012. URL https://CRAN.R-project.org/package=deamer. R package version 1.0. [p462]

I. Tolić, K. Miličević, N. Šuvak, and I. Biondić. Non-linear least squares and maximum likelihood estimation of probability density function of cross-border transmission losses. *IEEE Transactions on Power Systems*, 2017. [p461]

S. Urbanek. *jpeg: Read and Write JPEG Images*, 2014. URL https://CRAN.R-project.org/package=jpeg. R package version 0.1-8. [p463]

X.-F. Wang and B. Wang. *Decon: Deconvolution Estimation in Measurement Error Models*, 2013. URL https://CRAN.R-project.org/package=decon. R package version 1.2-4. [p462]

Mirta Benšić
Department of Mathematics, University of Osijek
Trg Lj. Gaja 6, HR-31 000 Osijek
Croatia
mirta@mathos.hr

Petar Taler
Department of Mathematics, University of Osijek
Trg Lj. Gaja 6, HR-31 000 Osijek

Croatia
petar@mathos.hr

Safet Hamedović
Faculty of Metallurgy and Materials Science
Travnička cesta 1, BA-72 000 Zenica
Bosnia And Herzegovina
safet.hamedovic@famm.unze.ba

Emmanuel Karlo Nyarko
Faculty of Electrical Engineering, Computer Science and Information Technology, University of Osijek
Kneza Trpimira 2B, HR-31 000 Osijek
Croatia
nyarko@etfos.hr

Kristian Sabo
Department of Mathematics, University of Osijek
Trg Lj. Gaja 6, HR-31 000 Osijek
Croatia
ksabo@mathos.hr

Splitting It Up: The spduration Split-Population Duration Regression Package for Time-Varying Covariates

by Andreas Beger, Daniel W. Hill, Jr., Nils. W. Metternich, Shahryar Minhas, and Michael D. Ward

Abstract We present an implementation of split-population duration regression in the **spduration** (Beger et al., 2017) package for R that allows for time-varying covariates. The statistical model accounts for units that are immune to a certain outcome and are not part of the duration process the researcher is primarily interested in. We provide insights for when immune units exist, that can significantly increase the predictive performance compared to standard duration models. The package includes estimation and several post-estimation methods for split-population Weibull and log-logistic models. We provide an empirical application to data on military coups.

Introduction

Duration models are an important class of statistical estimators that take into account the duration dependency of specific outcomes. A prominent example is that the risk of dying depends on the age of an individual. Newborns are at a greater risk of dying, but as they grow older this risk quickly declines and then gradually starts to increase again after the age of 9-10. While individual behavior (smoking, exercising, diet) and structural factors (health care, health regulations, urban vs rural) can increase or decrease the probability of dying over time, the underlying risk is time dependent and impacts on all individuals.

However, there are conditions under which not all individuals have the same underlying risk to experience a specific outcome and might not even be at risk at all. Consider the risk of acquiring a viral infection like the common flu. Let us initially assume that everyone is at risk of infection, and individual behavior (e.g., good hygiene) and structural factors (e.g., workplace) determine whether individuals catch the flu. Under these assumptions a standard duration model should provide us with efficient estimates of how different behavioral and structural factors impact on the general baseline risk. But, there might be individuals that are immune to the flu in the overall population—because they received vaccination, had the specific flu virus in the past, or have some other characteristic that makes it impossible for them to attract the disease. In this instance we have two underlying populations: an at risk population and an immune one. If the immune population is relatively large, estimates using a standard duration model will be biased and predictions inaccurate.

Additionally, risk factors might change over time requiring estimators that can account for such dynamics. We therefore present a split-population estimator that allows for the inclusion of time-varying covariates, which is an important distinction to other R implementations of split-population or cure models.

Immune populations and inference of duration processes

Regular duration models, where baseline risk is modeled by some distribution of time, were originally developed in health and demographic research and grew naturally from life tables and survival records for medical patients. Basic formulations of such models, like the parametric exponential or Weibull regressions or the semi-parametric Cox regression, implicitly assume that all subjects or units under observation, including right-censored observations, will eventually experience the event of interest. This assumption may be violated in many substantive areas and empirical applications, where a sub-population of units or individuals will never experience an event, and thus are effectively "cured."

The insight that populations might be split in regard to their baseline risk has been formulated as early as 1949 by Boag (1949) and Berkson and Gage (1952), who were researching survival rates in cancer patients. In their application some fraction of patients survived because their cancer was cured, while others relapsed after apparent remission due to levels of disease below detectable thresholds. The development of duration methods in health and medicine has also shaped the terminology conventionally used for such models. For example, split-population duration models are also referred to as cure rate models. Similarly, the basic concepts like survival and failure rates reference the survival of humans.

Yet the intuition underlying split-population duration models has led to applications in a broad

range of subject areas outside demographics and medicine. In an early and foundational application that reached beyond medicine, Schmidt and Witte (1989) examined criminal recidivism using data on close to 10,000 prisoners from the North Carolina prison system in the late 1970s and early 1980s to identify factors that influence whether a criminal relapses at all, and if so which factors are related to the amount of time between prison stints. This work already includes a full formulation of a model with independent covariates for both the duration equation and the risk or cure equation, although only with subject-specific covariates rather than time-varying covariates for multiple data points per individual.

In a complementary set of research from public health, Douglas and Hariharan (1994) use data from the United States to model the age at which individuals began to smoke, and Forster and Jones (2001) examine the impact of tobacco taxes on smoking and quitting decisions. DeYoung (2003), an economist, models the failure of new commercial banks in the US during the 1980's. Svolik (2008) uses a split-population duration framework to examine whether democratic regimes persist or revert to authoritarianism, and when. Building on this effort, split-population duration models have also been used, along with other models, to produce regular predictions for five different forms of political conflict for the Integrated Crisis and Early Warning System (ICEWS) project (Ward et al., 2013) and to model irregular leadership changes for the Political Instability Task Force (PITF; Beger et al., 2014).

Model development

Conventional duration models assume that all subjects will eventually fail, die, or experience a specific outcome. The likelihood for a data point with survival time t is thus the failure rate $f(t)$ at that time or the probability of survival beyond t, $S(t)$, depending on whether the subject has already experienced the event (δ_i) or is right-censored ($1 - \delta_i$):

$$\mathcal{L} = \prod_{i=1}^{N} (f(t_i))^{\delta_i} \times (S(t_i))^{1-\delta_i} \tag{1}$$

The major modeling question in this setting, which we will return to below, is the choice of a function form (e.g., exponential, Weibull, or log-logistic) describing the underlying hazard rate $h(t) = \frac{f(t)}{S(t)}$ over time.

The cumulative failure rate ($F(t) = 1 - S(t)$) over time converges to 1, meaning all subjects fail eventually. In the examples of applied research discussed above this assumption is untenable. Some cancer patients are cured after treatment, most young people never become regular smokers, and many states will not experience the kind of violence that persists in other parts of the world. The presence of a large sub-population which is not at risk for an event, will in practice inflate estimates of the survival fraction, and reduce hazard estimates for all subjects. This is the case because the underlying risk is estimated based on subjects that genuinely will fail and those that are cured. Hence, such a model will over predict the hazard for subjects that are not at risk (cured), and under predict for those who are at risk of eventually experiencing the event of interest.

We can incorporate the presence of a sub-population, where we label the subpopulation at risk with π, by rewriting the likelihood as:[1]

$$\mathcal{L}\{\theta|(t_1, \ldots, t_n)\} = \prod_{i=1}^{N} (\pi_i f(t_i))^{\delta_i} \times ((1 - \pi_i) + \pi_i S(t_i))^{1-\delta_i} \tag{2}$$

Crucially, this split-population framework is primarily useful in contexts where sub-populations are not clearly or easily identifiable. For example, there is a clear sub-population in a model of the age at first pregnancy for humans—men—which researchers can easily identify and exclude from their data. Less clear are answers to questions such as whether a cancer patient is cured or not cured given that they have no visible signs of cancer following treatment or have hit the 5-year disease free survival mark. In such situations, split-population models provide a way to infer sub-populations in a probabilistic fashion.

Early efforts focused only on the cure rate ($1 - \pi$) and treated it as a constant, but we can model membership in the subpopulation with its own covariates through a logistic link function:

[1] Usual presentation of the split-population duration framework in medical contexts focus on the "cured" sub-population. In our applications events are typically rare and it thus is easier to emphasize the "risk" subpopulation. As risk = 1− cured, this difference is trivial.

$$\pi_i = \frac{1}{1 + e^{-z_i \gamma}} \tag{3}$$

Where z_i is a vector of covariates for a subject at a given time. For interpretation, it is important to note that with time-varying covariates, the risk (or cured) estimate for a subject is particular to a given time point rather than being constant over all time periods in the spell.[2] Depending on the covariates, the risk estimate for a subject can thus fluctuate over time. To ease interpretation, it might be convenient to restrict covariates in the logit risk model to slow-moving, stable covariates in order to produce stable risk estimates for subjects.

The last component to complete the likelihood is the choice of a distribution for the shape of the hazard rate. The **spduration** package implements two hazard rate shapes, Weibull and log-logistic:

$$\begin{aligned}
&\text{Weibull} \\
f(t) &= \alpha\lambda\,(\lambda t)^{\alpha-1}\, e^{-(\lambda t)^{\alpha}} \\
S(t) &= e^{-(\lambda t)^{\alpha}} \\
h(t) &= \alpha\lambda\,(\lambda t)^{\alpha-1} \\
&\text{log-logistic} \\
f(t) &= \frac{\alpha\lambda\,(\lambda t)^{\alpha-1}}{\left(1 + (\lambda t)^{\alpha}\right)^2} \\
S(t) &= \frac{1}{1 + (\lambda t)^{\alpha}} \\
h(t) &= \frac{\alpha\lambda\,(\lambda t)^{\alpha-1}}{1 + (\lambda t)^{\alpha}}
\end{aligned}$$

Where $\lambda = e^{-x_i\beta}$ is a parameter of covariates. The Weibull density allows for hazard rates that are increasing, constant, or decreasing over survival time, while the log-logistic density also can fit rates that have a peak at a particular survival time.

Given the density distribution, the main quantity of interest is the conditional hazard $h(t, \pi)$, where both the risk/cure probabilities and hazard are conditional on survival to time t:

$$h\left(t, \pi\right) = \frac{f\left(t, \pi\right)}{S\left(t, \pi\right)} = \frac{\pi\left(t\right) \times f\left(t\right)}{\left(1 - \pi\left(t\right)\right) + \pi\left(t\right) \times S\left(t\right)} \tag{4}$$

$$\pi\left(t\right) = \frac{1 - \pi}{S\left(t\right) + \left(1 - \pi\right)\left(1 - S\left(t\right)\right)} \tag{5}$$

For a given unconditional risk rate π, the probability that a subject with survival time t is in the risk set decreases over time, because an increasing number of surviving cases consist of immune or cured $(1 - \pi)$ cases that will never fail. In the hazard rate, the failure rate in the numerator is conditional on the probability that a case is in the risk set, given survival up to time t, and the numerator is an adjusted survivor function that accounts for the fraction of cured cases by time t, which is $1 - \pi(t)$.

Fit a split-population model on coups data

In order to illustrate the package functionality, we examine a model of coups d'etat in Belkin and Schofer (2003). Belkin and Schofer's paper lends itself to re-analysis with a split-population duration model because they explicitly distinguish long-term structural risk factors for coups from more short-term triggering causes that can explain the timing of a coup in an at-risk regime. They argue that many countries never experience coups because coups are effectively impossible due to structural factors, while others that never experience coups are nevertheless at risk due to a different configuration of those same factors. Using language which fits nicely with the class of models described above, they write "[t]riggers are not the source of the original risk, and in the absence of structural causes, the presence of triggering factors alone cannot lead to a coup. Hence, triggers should not be equated with

[2] We use "spell" to designate all time periods observed for a subject up to the failure time. Subjects can theoretically have multiple spells, e.g., cancer patients who go into remission and relapse more than once, or states that experience multiple civil wars over their history.

coup risk. Rather, they are factors that may determine the exact timing of a coup in regimes that suffer from high coup risk" (Belkin and Schofer, 2003).

For their empirical test Belkin and Schofer develop a measure of "structural coup risk" which incorporates factors such as the strength of democratic institutions and civil society, and a recent history of successful coups. However, they implement a logistic regression model, which does not capture the process as described in their quote above. This is because the logistic regression model assumes that all observations are at risk for a coup (i.e., the probability of a coup is non-zero for all observations). Since their structural coup risk indicator is developed precisely to distinguish between at risk and "immune" cases, the split-population model allows one to examine whether the indicator effectively does so.

We begin by loading the package and the Belkin and Schofer replication data, and formatting the data to add several variables needed by the split-population duration model.

```
library("spduration")
data(bscoup)
?bscoup
head(bscoup[, 1:5])
  countryid  year  couprisk recentcoups rwar
1       909  1960 0.2200166           0    0
2       909  1961 0.2200166           0    0
3       909  1962 0.2200166           0    0
4       909  1963 0.2200166           0    0
5       909  1964 2.5702426           1    0
6       909  1965 2.5702426           2    0
```

The data are documented in '?bscoup', which also gives a reference to the source citation. It consists of a little more than 5,000 observations of 162 countries from 1960 to 2000. Each row corresponds to a country c in year t. Excluding the country and year identifiers, the data include 12 variables, including a binary indicator for successful coups.

```
str(bscoup)
'data.frame':        5463 obs. of  14 variables:
$ countryid  : num  909 909 909 909 909 909 909 920 920 920 ...
$ year       : num  1960 1961 1962 1963 1964 ...
$ couprisk   : num  0.22 0.22 0.22 0.22 2.57 ...
$ recentcoups: num  0 0 0 0 1 2 4 0 0 0 ...
$ rwar       : num  0 0 0 0 0 0 1 0 0 0 ...
$ milreg     : num  0 0 0 1 0 1 1 0 0 0 ...
$ wealth     : num  NA NA NA NA NA ...
$ instab     : num  4 2 9 9 12 13 12 NA 0 4 ...
$ coup       : Factor w/ 2 levels "no","yes": 1 1 1 2 2 2 1 1 1 1 ...
$ africa     : num  0 0 0 0 0 0 0 0 0 0 ...
$ eurnam     : num  0 0 0 0 0 0 0 1 1 1 ...
$ samerica   : num  0 0 0 0 0 0 0 0 0 0 ...
$ camerica   : num  0 0 0 0 0 0 0 0 0 0 ...
$ regconf    : num  0 0 0 0.019 0.019 ...
- attr(*, "datalabel")= chr ""
- attr(*, "time.stamp")= chr "30 Jul 2003 12:15"
- attr(*, "formats")= chr  "%9.0g" "%9.0g" "%9.0g" "%9.0g" ...
- attr(*, "types")= int  102 102 102 102 102 102 102 102 102 102 ...
- attr(*, "val.labels")= chr  "" "" "" "" ...
- attr(*, "var.labels")= chr  "country code .pr89" "year coded" "" "" ...
- attr(*, "version")= int 7
- attr(*, "label.table")=List of 1
..$ hadcoup: Named int  0 1
.. ..- attr(*, "names")= chr  "no" "yes"
```

Altogether the data include 290 coups, which makes for a positive rate of slightly more than 5% of all observations.

```
bscoup$coup <- ifelse(bscoup$coup == "yes", 1, 0)
table(bscoup$coup)
0    1
5173  290
```

Country	Year	Coup	Spell ID	Failure	Censor	At risk	Duration
Portugal	1960	0	1	0	0	1	1
...	...	0	1	0	0	1	...
...	1975	1	1	1	0	1	16
Portugal	1976	0	2	0	0	0	1
...	...	0	2	0	0	0	...
...	2000	0	2	0	1	0	20
Canada	1960	0	3	0	0	0	1
...	...	0	3	0	0	0	...
...	2000	0	3	0	1	0	41

Table 1: Example data frame with select duration variables.

Before we can estimate a split-population duration model with the data, we need to add several variables that capture the survival characteristics of the data.

```
bscoup <- add_duration(bscoup, "coup", unitID = "countryid",
+    tID = "year", freq = "year", ongoing = FALSE)
Warning message:
In attempt_date(data[, tID], freq) :
Converting to 'Date' class with yyyy-06-30
```

The `add_duration` function takes as input a data frame with a binary response variable—`coup`—that measures the occurrence of the event, or failure, recorded over discrete time periods. We assume that the data frame is in a cross-sectional time-series format that consists of multiple observations over time for a number of subjects, in our case countries.

Within the framework of duration modeling, these data conceptually consist of "spells," which are repeated observations of a unit from the time they enter the data (left-censoring) or after they experienced the event of interest, until the next event or the end of observation (right censoring). Table 1 shows how a single country could have two spells during the observation period. Observations for Portugal make up two spells, one that begins in 1960, when our data start, to a coup event in 1975, when a second spell starts and continues until either the next coup or the data end. Canada on the other hand experienced no coups and so observations for that country from 1960 to 2000 make up one spell.

The `add_duration` functions identifies spells in the data and returns the data frame with several additional variables needed for estimation. The most important of these are shown in Table 1. Using coups as the outcome, failure is coded when a coup occured. This also triggers a new spell if there are more observations for a country. If a spell ends without a failure event it is coded as right censored (`censor`). This can occur either because we have reached the end of observed data—the year 2000, or because a country dropped out of the data, e.g., if it was subsumed by another country. Duration is then coded for each spell by counting the number of time periods until a spell ends. The at risk coding depends on whether a spell ended in failure—spells that end in failure are coded as being at risk for their entire duration, otherwise they are coded as not being at risk.

The `add_duration` function also adds several other variables that should be of less concern in typical use. If the data contain outcomes that can occur over multiple time periods, like civil wars, `add_duration` should be used with the `ongoing = TRUE` flag so that only the first time period is treated as a failure, rather than as repeated failures. In that case `ongoing` is a marker for ongoing events beyond the initial onset. The variable `cured` is the inverse of `atrisk`, i.e., $1-$ `atrisk`. Lastly, `t.0` marks the starting time for the row. It is by default `duration` -1, but is retained to allow extension, if needed in the future, for observations that do not span a fixed time period, e.g. if observations are taken at irregular time intervals.

Though the data used in this example are recorded annually, the function supports annual, monthly, or daily data and will try to convert the `tID` input to class `Date` given the provided argument in `freq`, as indicated by the warning it returned in the example above. This can be avoided by first converting any numeric dates using R's `"Date"` class.

Another important question is how to handle consecutive 1s in the response variable. This is controlled with the `ongoing` argument. In the case of civil war occurrence, the y variable records all years during which a country experienced a civil war. Here `ongoing` should be set to `TRUE` so that the models try to predict the onset of the civil war, but disregard ongoing conflicts (`failure` is set to `NA` for

these cases, dropping them from analysis). With ongoing set to FALSE, successive 1s in y are treated as distinct new failures, and kept in the analysis. This makes sense for discrete, short-term events like coups. Countries can experience distinct coups in successive years.

The spdur function is the primary function in the package and produces a regression model object of class "spdur" which can then be used with further methods. We begin by fitting first a Weibull (the default) and then a log-logistic split-population duration model using the coups data, including the measure of coup risk in the logit (risk) equation.

```
weib_model <- spdur(
+    duration ~ milreg + instab + regconf,
+    atrisk ~ couprisk + wealth + milreg + rwar + regconf + samerica +
+      camerica,
+    data = bscoup, distr = "weibull", silent = TRUE)

loglog_model <- spdur(
+    duration ~ milreg + instab + regconf,
+    atrisk ~ couprisk + wealth + milreg + rwar + regconf + samerica +
+      camerica,
+    data = bscoup, distr = "loglog", silent = TRUE)
```

Using the summary function on either model object will produce standard output showing the model formula, estimates for the duration and risk equations, and test statistics with p-values. The estimates are based on maximum likelihood estimation and standard errors are derived using the Hessian matrix of the optimization results. The spdur parameter estimates can be exported by calling xtable (see Dahl, 2016) directly on the "spdur" object to produce Table 2.

```
library("xtable")
tbl <- xtable(loglog_model, caption = "Coup model with log-logistic hazard",
+    label = "loglog_table")
print(tbl, caption.placement = "top", comment = FALSE,
+    include.rownames = FALSE)
```

Parameter	Estimate	Std. Error	t value	Pr(>|t|)
Dur_(Intercept)	2.40	0.21	11.43	0.00
Dur_milreg	−1.13	0.21	−5.46	0.00
Dur_instab	−0.09	0.02	−4.79	0.00
Dur_regconf	−2.52	2.16	−1.16	0.24
log(alpha)	−0.45	0.06	−7.09	0.00
Risk_(Intercept)	2.93	1.87	1.57	0.12
Risk_couprisk	0.59	0.32	1.82	0.07
Risk_wealth	−0.36	0.28	−1.29	0.20
Risk_milreg	10.82	9.31	1.16	0.25
Risk_rwar	−0.52	0.94	−0.56	0.58
Risk_regconf	−5.43	5.62	−0.97	0.33
Risk_samerica	2.09	1.45	1.45	0.15
Risk_camerica	−0.39	0.73	−0.53	0.59

Table 2: Coup model with log-logistic hazard.

Table 2 shows estimates from the duration equation, beginning with the intercept, and then estimates from the risk equation. The duration component of the model is in accelerated failure time format and the coefficient estimates are the log of expected time to failure. The negative coefficient for military regimes, for example, means that the expected time to a coup is shorter in military regimes than non-military regimes, holding all other factors constant. In the risk equation, positive coefficients mean higher risk of coup. Thus, military regimes have a higher risk of experiencing a coup.

One may also use the AIC and BIC function to calculate the information criterion statistics for spdur objects. In our example, both models are close enough in both statistics to be indistinguishable, so we will continue to focus on the log-logistic form.

```
matrix(c(
+    AIC(weib_model), AIC(loglog_model), BIC(weib_model), BIC(loglog_model)
+    ), ncol = 2, dimnames = list(c("Weibull", "Loglog"), c("AIC", "BIC")))
             AIC      BIC
```

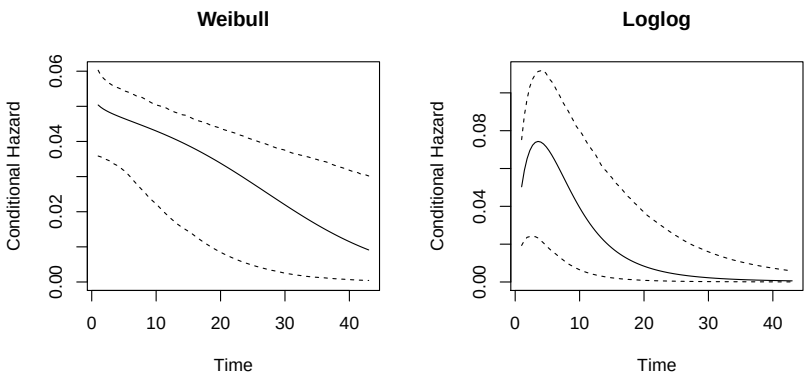


Figure 1: Hazard rate plots for the Weibull and log-logistic coup models.

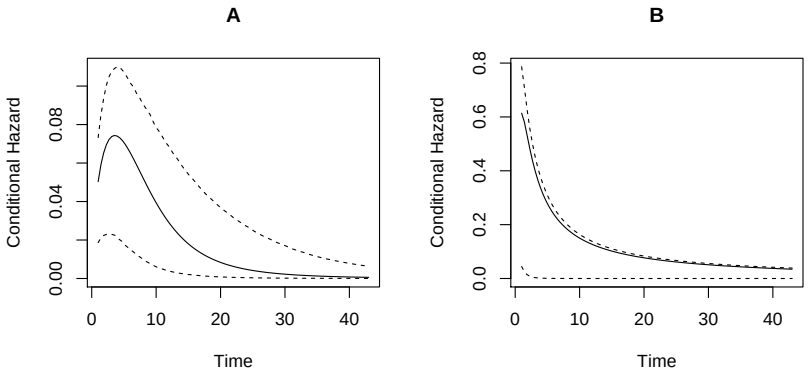


Figure 2: Plots of the hazard rate for the log-logistic model of coups. Graph A uses the default mean values for covariates, while graph B uses user-specified variable values for a high-risk military regime.

```
Weibull 1329.908 1348.972
Loglog  1331.214 1350.278
```

The package includes two types of plots that show the estimated hazard rates and predictive performance, respectively. The hazard rates can be plotted by either calling `plot_hazard` directly, or with `plot(x,type = "hazard")`, on a fitted `spdur` object. It will produce a plot of the conditional hazard, which is the probability of survival conditional on the covariates in the risk and duration equations, and conditional on survival up to time t, when holding all covariates at their sample means. The function calculates the average hazard rate as well as 90% confidence intervals, which are produced by simulating values from the estimated sampling distributions of the model parameters.

```
plot(weib_model,   type="hazard", main="Weibull")
plot(loglog_model, type="hazard", main="Loglog")
```

By default the `plot_hazard` function uses the mean values of the covariates during the simulations, but users can choose specific covariate values by entering them as vectors in the arguments `xvals` and `zvals`, which correspond to the covariates in the duration and risk equations, respectively. The command below creates the graph A in Figure 2.

```
plot(loglog_model, type = "hazard", main = "A")
```

As mean values are maybe not that interesting, we can also generate scenarios by entering values manually. The values are used in the same order that variables are specified in the equations used to estimate the models, which also corresponds to the order of variables in Table 2. Note the inclusion of an intercept term. This produces plot B in Figure 2

```
plot(loglog_model, type = "hazard", xvals = c(1, 1, 10, 0.05),
+    zvals = c(1, 7, 8.64, 1, 1, 0.05, 0, 0), main = "B")
```

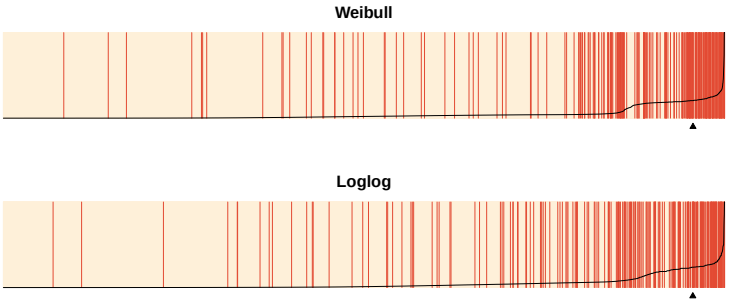


Figure 3: In-sample separation plots of Weibull and log-logistic model conditional hazard predictions.

While `plot(x,type = "hazard")` will produce a hazard rate plot, without a type argument, `plot.spdur` will produce a separation plot. Separation plots are a graphical display for evaluating model predictions (Greenhill et al., 2011). The code below produces Figure 3:

```
plot(weib_model, lwd1 = 1, lwd2 = 1)
title(main="Weibull")
plot(loglog_model, lwd1 = 1, lwd2 = 1)
title(main="Loglog")
```

The option `endSpellOnly` is set to `FALSE` so that every observation, not only those at the end of a spell, is used in the plot. By default, the `plot` function will calculate the conditional hazard for each observation. The separation plot sorts observations from left to right according to the predicted probability assigned by the model (higher values to the right), and shows each event/failure as red line, with non-events shown in beige. This makes it easy to see whether the model is assigning high probabilities of failure to actual cases of failure, and low probabilities to non-failures.

Underlying both plotting functions is the `predict` function, which can be used on an object of class `spdur` to generate several kinds of predictions, including the probability that an observation is "at-risk" and the probability of failure for a given time period. By default `predict` calculates the conditional hazard, but the probability that an observation is at risk may also be of interest. These models are most appropriate in cases where there is a strong theoretical reason to suspect some units are not at risk, as in the canonical applications (not everyone smokes, some cancer patients are cured, some convicts do not relapse, etc.). Whether an observation is cured or at-risk is ultimately unknowable, but estimating cure probabilities can give users some idea of whether a split-population model is appropriate for their application. Here we use `predict` to estimate conditional cure probabilities from the Weibull model.

```
bscoup$ccure_prob <- predict(weib_model, type = "conditional cure",
+    newdata = bscoup, na.action = "na.exclude")
mean(bscoup$ccure_prob, na.rm = TRUE)
[1] 0.4333912
```

Out-of-sample testing

Finally, we demonstrate how to evaluate a model's out-of-sample predictions. We will use the data from 1996 onwards as the test set, and the prior data for training purposes. The `add_duration` function retrospectively codes the risk variable based on how a spell ended, and we therefore need to take care in how we add the duration variables for each data set. For the training data we need to subset the training set first, so that coups in the tests set don't influence the risk coding in the training data.

```
data(bscoup)
bscoup$coup <- ifelse(bscoup$coup == "yes", 1, 0)
coup_train <- bscoup[bscoup$year < 1996, ]
coup_train <- add_duration(coup_train, "coup", unitID = "countryid",
+    tID = "year", freq = "year", ongoing = FALSE)
```

For the test set it is recommended to add the duration variables and *then* subset the test set. Since the test set is later in time than the training set we do not have to worry about contamination of the risk coding, but if we subset the data before building the duration variables we will start all duration counters at 1996, when in fact we can safely use the previous historic coup information. To do this we need to build the duration variables first:

```
coup_test  <- add_duration(bscoup, "coup", unitID = "countryid",
+    tID = "year", freq = "year", ongoing = FALSE)
coup_test  <- coup_test[coup_test$year >= 1996, ]
```

Now we can fit new models using the training data, and calculate predictions from these models for the test data, using `predict(...,newdata = coup_test)`.

```
weib_model2   <- spdur(
+    duration ~ milreg + instab + regconf,
+    atrisk ~ couprisk + wealth + milreg + rwar + regconf + samerica +
+      camerica,
+    data = coup_train, distr = "weib", silent = TRUE)

loglog_model2 <- spdur(
+    duration ~ milreg + instab + regconf,
+    atrisk ~ couprisk + wealth + milreg + rwar + regconf + samerica +
+      camerica,
+    data = coup_train, distr = "loglog", silent = TRUE)

weib2_test_p   <- predict(weib_model2, newdata = coup_test, na.action = "na.omit")
loglog2_test_p <- predict(loglog_model2, newdata = coup_test, na.action = "na.omit")
```

Since we are predicting for data that is not contained in the `spdur` model objects, we have to use the `separationplot` functions directly from the package to produce Figure 4.

```
library("separationplot")
obs_y <- coup_test[complete.cases(coup_test), "coup"]

par(mfrow=c(2,1),mar=c(2,2,2,2))
separationplot(weib2_test_p,   obs_y, newplot = FALSE)
separationplot(loglog2_test_p, obs_y, newplot = FALSE)
```

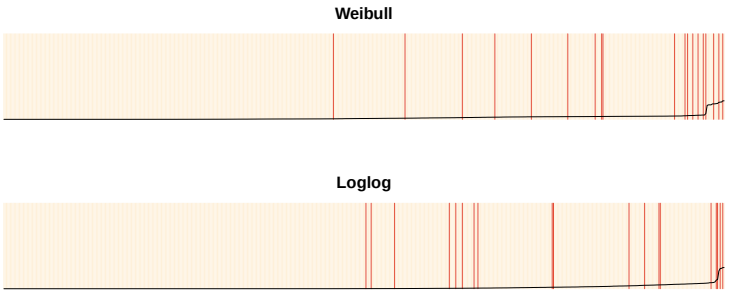


Figure 4: Out-of-sample separation plots.

Additional discussion

Censoring considerations

Truncation and censoring are problematic for split-population duration models, as they are for standard duration regression, and pose some additional considerations. In left-truncation we do not observe data for a spell prior to some date, and thus have incomplete and inaccurate values for the duration or time to failure for a spell. Since immune spells in the sample are over time going to distinguish themselves with exceptionally long survival times compared to spells at risk which fail periodically, left-censoring also makes it more difficult to distinguish the immune and at risk subpopulations.

Sometimes information about previous failures in the data is available beyond the time period over which covariates are observed, making it possible to ameliorate or eliminate left-censoring by using the information of previous failures when constructing the necessary duration variables with `add_duration()`.

Right-censoring, where spells end before outcomes are observed, also poses a unique problem in the split-population framework. Although right-censored spells themselves are accommodated in the modeling function, they impact the coding of at risk vs. immune spells. The `add_duration()` function retroactively codes all observations in a spell as at risk if the spell itself ended in failure. Right-censored spells are coded as immune over their entire duration. This can lead to some misclassification of observations as immune even though they experience failure at some point in the unobserved future.

Furthermore, as the example above shows, in out-of-sample testing based on some kind of data partitioning scheme, this coding scheme can lead to unintentional contamination of in-sample cases with knowledge of out-of-sample failures through the risk coding for failed spells. This leads to two recommendations. First, data should be partitioned spell or block-wise, e.g., by withholding the last x years of data, and not randomly. Second, given the two concerns of left-censoring and non-independence induced through the duration and risk coding, care should be taken to ensure that duration data are built without access to future information in another data partition.

Comparison with other software

The **survival** library (Thernau, 2000, 2015), arguably the most well-established library for survival and event history analysis in R, allows for the estimation of a wide variety of semi-, non-, and parametric survival models, and provides facilities for handling and providing descriptive summaries of survival and event history data. It does not include cure rate or split-population mixture models of the type we have implemented though.

One useful extension of our project in the future would be to integrate and adapt the `"Surv"` class for survival data in **survival** to the **spduration** library. This would require some modification of the class, as our model also requires information on the risk status of spells, but would give access to the much broader functionality, especially for descriptive summaries of data, in the **survival** library.

Other currently available R routines for estimation of split-population/cure rate duration models include the packages **smcure** (Cai et al., 2012b), described in Cai et al. (2012a), and **nltm** (Garibotti and Tsodikov, 2010). The **smcure** package implements semi-parametric proportional hazards and accelerated failure time cure models using estimation procedures presented in Peng (2003) and Zhang and Peng (2007).[3] The package **nltm** will estimate a semi-parametric proportional hazard cure model, and a number of other survival models, using the approach developed in Tsodikov (2003) and Tsodikov and Garibotti (2007). Semi-parametric estimation of the hazard function is attractive because it requires no assumption about its shape, and such assumptions can be difficult in practice to empirically evaluate. The main advantage **spduration** offers is the ability to include time varying covariates, which gives it a broader range of practical applications, as **smcure** and **nltm** will only accommodate case-based data. **spduration** also features more post-estimation methods than alternative packages, including those discussed above to create and evaluate model predictions.

Conclusion

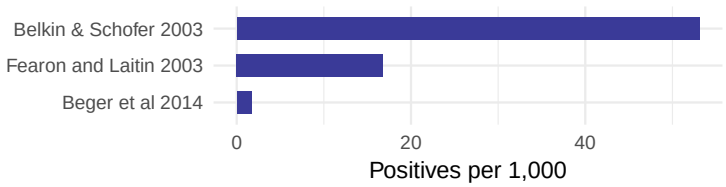


Figure 5: Rates of positive outcomes in select publications with binary outcomes.

Many outcomes of interest are rare events. For example, coups, war onset, or mass killings are exceedingly rare events when considering that most countries do not experience high levels of violence. Figure 5 shows a few examples of published research that models binary outcomes. Fearon and Laitin (2003) is a widely cited study of civil war onset that uses yearly observations of all major countries; Beger et al. (2014) is an example of the positive rates when moving to monthly data for a similar set of countries. The positive rates range from 5 to less than 0.2% of all data points.[4]

[3] There is also the older **gfcure** package, which estimates the parametric accelerated failure time cure model discussed in Peng et al. (1998). A version for Windows is available at `http://post.queensu.ca/~pengp/software.html`.

[4] For a discussion of the difficulties rare events can pose for prediction see King and Zeng (2001a) and King and Zeng (2001b).

In the language of machine learning, we are dealing with highly imbalanced classes. It is a well-recognized problem and has led to the development or use of several specialized mixture models like zero-inflated Poisson and negative binomial regression for count data, and a zero-inflated ordered probit for ordinal outcomes (Bagozzi et al., 2015). Split-population duration regression provides another principled solution to the challenges posed by data in this domain, but, unlike other solutions to the sparse outcome problem, also addresses underlying temporal dynamics that are an important part of the non-independent data political scientists and other social scientists commonly use.

Split-population duration models are not only appealing in a technical sense, but they also match the logic or intuition many social scientists use when they distinguish long-term risk factors from more fleeting triggering causes. The example we have used, Belkin and Schofer (2003), is a particularly clear illustration of how well the language of theorists maps onto the model intuition.

Bibliography

B. E. Bagozzi, D. W. Hill, W. H. Moore, and B. Mukherjee. Modeling two types of peace the zero-inflated ordered probit (ziop) model in conflict research. *Journal of Conflict Resolution*, 59(4):728–752, 2015. URL https://doi.org/10.1177/0022002713520530. [p484]

A. Beger, C. L. Dorff, and M. D. Ward. Ensemble forecasting of irregular leadership change. *Research & Politics*, 1(3):1–7, 2014. URL https://doi.org/10.1177/2053168014557511. [p475, 483]

A. Beger, D. Chiba, D. Hill, N. Metternich, and S. Minhas. *Spduration: Split-Population Duration (Cure) Regression*, 2017. URL https://CRAN.R-project.org/package=spduration. R package version 0.16.0. [p474]

A. Belkin and E. Schofer. Toward a structural understanding of coup risk. *Journal of Conflict Resolution*, 47(5):594–620, 2003. URL https://doi.org/10.1177/0022002703258197. [p476, 477, 484]

J. Berkson and R. P. Gage. Survival curve for cancer patients following treatment. *Journal of the American Statistical Association*, 47(259):501–515, 1952. URL https://doi.org/10.1080/01621459.1952.10501187. [p474]

J. W. Boag. Maximum likelihood estimates of the proportion of patients cured by cancer therapy. *Journal of the Royal Statistical Society B*, 11(1):15–53, 1949. [p474]

C. Cai, Y. Zou, Y. Peng, and J. Zhang. Smcure: An R-package for estimating semiparametric mixture cure models. *Computer Methods and Programs in Biomedicine*, 108(3):1255–1260, 2012a. URL https://doi.org/10.1016/j.cmpb.2012.08.013. [p483]

C. Cai, Y. Zou, Y. Peng, and J. Zhang. *Smcure: Fit Semiparametric Mixture Cure Models*, 2012b. URL https://CRAN.R-project.org/package=smcure. R package version 2.0. [p483]

D. B. Dahl. *Xtable: Export Tables to LaTeX or HTML*, 2016. URL https://CRAN.R-project.org/package=xtable. R package version 1.8-2. [p479]

R. DeYoung. The failure of new entrants in commercial banking markets: A split-population duration analysis. *Review of Financial Economics*, 12(1):7–33, 2003. URL https://doi.org/10.1016/S1058-3300(03)00004-1. [p475]

S. Douglas and G. Hariharan. The hazard of starting smoking: Estimates from a split population duration model. *Journal of Health Economics*, 13(2):213–230, 1994. URL https://doi.org/10.1016/0167-6296(94)90024-8. [p475]

J. D. Fearon and D. D. Laitin. Ethnicity, insurgency, and civil war. *American Political Science Review*, 97 (1):75–90, 2003. URL https://doi.org/10.1017/S0003055403000534. [p483]

M. Forster and A. M. Jones. The role of tobacco taxes in starting and quitting smoking: Duration analysis of British data. *Journal of the Royal Statistical Society A*, 164(3):517–547, 2001. URL https://doi.org/10.1111/1467-985X.00217. [p475]

G. Garibotti and A. Tsodikov. *Nltm: Nonlinear Transformation Models*, 2010. URL https://cran.r-project.org/src/contrib/Archive/nltm/. R package version 1.4.1. [p483]

B. Greenhill, M. D. Ward, and A. Sacks. The separation plot: A new visual method for evaluating the fit of binary models. *American Journal of Political Science*, 55(4):991–1002, 2011. URL https://doi.org/10.1111/j.1540-5907.2011.00525.x. [p481]

G. King and L. Zeng. Explaining rare events in international relations. *International Organization*, 55 (03):693–715, 2001a. URL https://doi.org/10.1162/00208180152507597. [p483]

G. King and L. Zeng. Logistic regression in rare events data. *Political analysis*, 9(2):137–163, 2001b. URL https://doi.org/10.1093/oxfordjournals.pan.a004868. [p483]

Y. Peng. Fitting semiparametric cure models. *Computational Statistics & Data Analysis*, 41(3):481–490, 2003. URL https://doi.org/10.1016/S0167-9473(02)00184-6. [p483]

Y. Peng, K. B. Dear, and J. Denham. A generalized F mixture model for cure rate estimation. *Statistics in Medicine*, 17(8):813–830, 1998. URL https://doi.org/10.1002/(SICI)1097-0258(19980430)17:8<813::AID-SIM775>3.0.CO;2-#. [p483]

P. Schmidt and A. D. Witte. Predicting criminal recidivism using "split population" survival time models. *Journal of Econometrics*, 40(1):141–159, 1989. URL https://doi.org/10.1016/0304-4076(89)90034-1. [p475]

M. Svolik. Authoritarian reversals and democratic consolidation. *American Political Science Review*, 102 (2):153–168, 2008. URL https://doi.org/10.1017/S0003055408080143. [p475]

T. M. Thernau. *Modeling Survival Data: Extending the Cox Model*. Springer-Verlag, New York, 2000. [p483]

T. M. Thernau. *A Package for Survival Analysis in S*, 2015. URL http://CRAN.R-project.org/package=survival. R package version 2.38. [p483]

A. Tsodikov. Semiparametric models: a generalized self-consistency approach. *Journal of the Royal Statistical Society B*, 65(3):759–774, 2003. URL https://doi.org/10.1111/1467-9868.00414. [p483]

A. Tsodikov and G. Garibotti. Profile information matrix for nonlinear transformation models. *Lifetime Data Analysis*, 13(1):139–159, 2007. URL https://doi.org/10.1007/s10985-006-9023-z. [p483]

M. D. Ward, N. W. Metternich, C. L. Dorff, M. Gallop, F. M. Hollenbach, A. Schultz, and S. Weschle. Learning from the past and stepping into the future: Toward a new generation of conflict prediction. *International Studies Review*, 15(4):473–490, 2013. URL https://doi.org/10.1111/misr.12072. [p475]

J. Zhang and Y. Peng. A new estimation method for the semiparametric accelerated failure time mixture cure model. *Statistics in Medicine*, 26(16):3157–3171, 2007. URL https://doi.org/10.1002/sim.2748. [p483]

Andreas Beger
Ward Associates / Predictive Heuristics
Tallinn
Estonia
adbeger@gmail.com

Daniel W. Hill, Jr.
Department of International Affairs
University of Georgia
Athens, GA
United States of America
dwhill@uga.edu

Nils W. Metternich
Department of Political Science
University College London
London
United Kingdom
n.metternich@ucl.ac.uk
Nils W. Metternich acknowledges support from the Economic and Social Research Council (ES/L011506/1)

Shahryar Minhas
Department of Political Science
Duke University
Durham, NC
United States of America
sfm12@duke.edu

Michael D. Ward
Department of Political Science
Duke University
Durham, NC
United States of America
michael.d.ward@duke.edu

Bayesian Regression Models for Interval-censored Data in R

by Clifford Anderson-Bergman

Abstract The package **icenReg** provides classic survival regression models for interval-censored data. We present an update to the package that extends the parametric models into the Bayesian framework. Core additions include functionality to define the regression model with the standard regression syntax while providing a custom prior function. Several other utility functions are presented that allow for simplified examination of the posterior distribution.

Introduction

Interval-censoring occurs when observations are not known exactly, but rather up to an interval. For example, suppose a component of a machine is inspected at time c_1 and c_2. The component is observed to be operational at c_1, but broken at c_2. In such a case, while the exact failure time is not known, it is known that the event occurred inside the interval $(c_1, c_2]$. In some cases, these intervals are small and the interval-censored aspect of the data can be ignored with only minor biases. For example, if age is reported in years, it is likely to be interval-censored due to binning, i.e. reporting "28 years old" typically implies age is in the interval $[28, 29)$. Similarly, if the intervals are non-overlapping, such as reported income brackets, one can simply treat the data as ordinal data and use appropriate models. However, if the data set contains moderate sized overlapping intervals, then interval-censoring methods should be used for valid inference. Note that a right censored observation can be represented as (C, ∞), where C is the censoring time, left censoring can be represented as $[0, C)$ and an uncensored observation occurring at time t can be represented as $[t, t]$.

Although interval-censoring is not strictly a survival analysis problem (for example, the Tobit model (Tobin, 1958)), this work focuses on the survival analysis setting in which the outcome of interest is time to event. A common assumption in many interval censored models, including those provided by **icenReg**, is that the distribution of the inspection times is independent of the event time of interest (Gruger et al., 1991). This can be framed as each subject having an event time of interest, t_i, and a set of inspections $c_{i0} = 0 < c_{i1} < ... < c_{ik_i} = \infty$ where the subject is inspected to determine if the event has occurred. The interval $[c_{ij}, c_{ij+1})$ such that $t_i \in [c_{ij}, c_{ij+1})$ is then recorded as the interval for subject i. The independence assumption states that t_i is independent of c_{ij}.

The standard univariate estimator is the non-parametric maximum likelihood estimator (NPMLE) (Turnbull, 1976), which can be viewed as a generalization of the Kaplan-Meier curves (Kaplan and Meier, 1958) that allow for interval-censoring (Ng, 2002). Many of the standard survival regression models can be extended to the interval-censored such as the proportional hazards, accelerated failure time (AFT) model and proportional odds. Semi-parametric models in which the baseline distribution is fit with the NPMLE are often used to avoid the need to specify the baseline distribution (Finkelstein, 1986), (Rossini and Tsiatis, 1996). While it has been shown that the regression coefficients are asymptotically normal and bootstrap procedures can be used for inference on the regression parameters (Huang, 1995), it is also noted that the asymptotic distribution of the baseline survival curve is currently an open question. This implies that while standard errors can be produced for the regression coefficients, quantifying the uncertainty in estimated survival probabilities when using the semi-parametric models is not currently available; even the bootstrap estimator has been shown to be inconsistent (Sen and Xu, 2015). It has also been noted that while the regression coefficients are consistent, a non-trivial upward bias in the coefficient estimates has been observed (Pan, 1999). Fully parametric regression survival models can also be used and are fairly straightforward to implement (Rabinowitz et al., 1995). In contrast to semi-parametric models, fully parametric models provide more efficient inference and allow for quantification of uncertainty of survival estimates at the cost of requiring assumptions of the family of baseline distribution, although it has been shown empirically that inference is fairly robust to mis-specification of the baseline distribution (Lindsey, 1998). Fully parametric models can be easily extended to the Bayesian framework (Gómez et al., 2004). For a thorough review of the non-parametric, semi-parametric and fully-parametric models in the interval-censoring context, see (Sun, 2007). In this work, we focus on parametric regression models in the Bayesian framework.

In general, interval-censored data is less informative than uncensored data. As such, incorporating prior information into an analysis using Bayesian methods can be especially useful. Recent additions to the R package **icenReg** (Anderson-Bergman, 2017) allow for simplified Bayesian analysis using standard regression formulas and user written prior functions. In Section F.2, the regression models available in **icenReg** are mathematically formulated. In Section F.3, the general form of the posterior

distribution is presented and the MCMC sampler is briefly discussed. In Section F.4, the core Bayesian functions in **icenReg** are presented. In Section F.5, an example analysis on a classic dataset is presented.

Regression models

To completely define a parametric survival regression model, one needs to specify the

- Baseline distribution
- Effect of the covariates on the baseline distribution

In **icenReg**, several classic survival baseline distributions are included: Weibull, gamma, exponential, log-normal and log-logistic.

At this time, three classic regression models are supported in **icenReg**: proportional hazards, AFT and proportional odds. In describing these regression models, we use several standard survival definitions. Defining $f(t)$ and $F(t)$ to represent the probability density function and cumulative density function for a given distribution, the survival distribution is defined as $S(t) = 1 - F(t)$ and the hazard function $h(t) = \frac{f(t)}{S(t)}$. The functions $h_o(t)$ and $S_o(t)$ represent the baseline hazard and survival function; i.e. the corresponding functions if all covariates are equal to 0. The vector X represents a subject's covariates, α represents a vector of parameters defining the baseline distribution and β represents a vector of regression coefficients.

The proportional hazards model can be defined as having the relation

$$h(t|\alpha, \beta, X) = h_o(t|\alpha) \exp(X^T \beta).$$

This definition can be used to interpret a regression coefficient β_j as a one unit increase in x_j is associated with an $\exp(\beta_j)$ fold increase in the hazard at any time.

The proportional odds model is defined as the relation

$$\frac{S(t|\alpha, \beta, X)}{1 - S(t|\alpha, \beta, X)} = \exp(X^T \beta) \frac{S_o(t|\alpha)}{1 - S_o(t|\alpha)}.$$

This definition can be used interpret a regression coefficient β_j as a one unit increase in x_j is associated with $\exp(\beta_j)$ fold increase in the odds of survival at any given time.

The AFT model is defined by the relation

$$S(t|\alpha, \beta, X) = S_o(t \exp(X^T \beta)|\alpha).$$

This definition can be used to interpret a regression coefficient β_j as a one unit increase in x_j is associated with events occuring $\exp(\beta_j)$ fold faster.

To define the likelihood function, we let n_1 be the number of uncensored subjects, n_2 be the number of interval-censored subjects (note that this can include left and right censored subjects), t_i be subject i's event time if subject i is uncensored, $\{L_i, R_i\}$ be the left and right side of the interval containing subject i's event time if subject was censored and X_i be a vector of subject i's covariates. Then the likelihood can be written as

$$\prod_{i=1}^{n_1} f(t_i|\alpha, \beta, X_i) \times \prod_{i=n_1+1}^{n_1+n_2} S(L_i|\alpha, \beta, X_i) - S(R_i|\alpha, \beta, X_i)$$

under the implication that if n_1 or n_2 are equal to 0, the corresponding term of the likelihood function reduces to 1.

Bayesian inference

To perform Bayesian inference, the prior is multiplied by the likelihood function to form the posterior distribution. For the Bayesian models included in **icenReg**, the posterior distribution is proportional to

$$p(\alpha, \beta) \times \prod_{i=1}^{n_1} f(t_i|\alpha, \beta, X_i) \times \prod_{i=n_1+1}^{n_1+n_2} S(L_i|\alpha, \beta, X_i) - S(R_i|\alpha, \beta, X_i)$$

where p is the prior distribution on the α and β parameters. Because the posterior is not in closed form for these models, Markov Chain Monte Carlo (MCMC) methods are used to draw samples from the posterior distribution.

In **icenReg**, MCMC sampling is carried out by an adaptive block updater (Haario et al., 2001). Default behavior is to first calculate the maximum likelihood estimator (MLE)[1], use the MLE point estimates as initial values and the inverse Fisher's information as an initial estimate for the posterior covariance. During the burn-in period, the posterior covariance is updated. A default target acceptance rate of 0.25 is used, as suggested in (Gelman et al., 1996).

Alternatively, the user can specify *not* to use the MLE and Fisher's information to build the starting proposal distribution. In this case, the starting proposal covariance matrix will be the identity matrix multiplied by a user-provided scalar (by default 0.1), which then has the option to adaptively learn the covariance matrix. While this is generally not recommended for efficiency purposes, it may be beneficial when the prior is strongly informative compared with the likelihood function. As an extreme example, if all the data were right censored, the MLE would be degenerate but an informative prior can still lead to valid Bayesian inference. In such cases, starting at the MLE would cause the MCMC algorithm to fail.

Core functionality

Function Name	Basic Description
`ic_bayes()`	Fit Bayesian interval-censored regression model
`bayesControls()`	Outputs control parameters for MCMC algorithm
`sampleSurv()`	Draws samples *of* the posterior survival distribution
`ic_sample()`	Draws samples *from* the posterior survival distribution
`imputeCens()`	Draw samples *from* the distribution, conditional on censoring interval
`survCIs()`	Credible intervals for survival curve
`plot()`	Plots posterior median survival curve

The workhorse for fitting Bayesian regression models is `ic_bayes()`. The arguments are defined as the following.

```
ic_bayes(formula, data,
         logPriorFxn = function(x) return(0),
         model = "ph", dist = "weibull",
         weights = NULL, controls = bayesControls(),
         useMCores = F)
```

The `formula` argument declares the likelihood function in the same manner as other **icenReg** model functions, to be demonstrated in the following section. The `logPriorFxn` argument allows the user to write a custom prior function that takes in a vector of parameters and returns the log prior density (or a value equal up to an additive constant). The order of the values should be the same order as the parameters returned when a user calls `coef()` on a model. Default behavior is to use a flat prior. The `model` argument declares the regression model, with choices `"ph"` (proportional hazards), `"po"` (proportional odds) and `"aft"` (accelerated failure time). The `dist` defines the baseline distribution, with options `"exponential"`, `"weibull"`, `"gamma"`, `"lnorm"` (log-normal) and `"loglogistic"`. The function argument `controls` accepts a list of control parameters for the MCMC sampler, see `?bayesControls` for details of options. The argument `useMCores` is a logical variable indicating whether the multiple chains should be run in parallel. If set to `TRUE`, a cluster must be registered in advance; this is demonstrated in Section F.5.

The output from `ic_bayes()` provides a list of samples from the posterior of α and β. Users are often interested in the survival probabilities for subjects with different sets of covariates, which requires a decent amount coding and double checking differing distribution parameterization. To simplify this process, the `sampleSurv()` function allows a user to take draws of the posterior survival distribution for a given set of covariates. The arguments are defined as

```
sampleSurv(fit, newdata = NULL,
           p = NULL, q = NULL,
           samples = 100)
```

The argument `fit` is a fit returned from `ic_bayes()`. The argument `newdata` is a `"data.frame"` which includes the set of covariates from which we would like to draw the posterior probabilities from.

[1]The MLE and *not* the maximum a posterior (MAP) is used, as the likelihood and its derivatives are hard coded into **icenReg**, but priors are allowed to be generically supplied by user without derivatives.

If newdata is NULL, the baseline distribution is used. A user should either provide a numeric vector p of percentiles to sample *or* a numeric vector q, a set of times to sample the cumulative probabilities at.

The function ic_sample() allows a user to take posterior samples of event times for a given set of covariates. The arguments are defined as

```
ic_sample(fit, newdata = NULL,
          sampleType = "fullSample",
          samples = 5)
```

The argument sampleType has two options: "fullSample", in which event times are sampled from the full posterior and "fixedParSample", in which event times are sampled conditional on the MAP estimates.

In some cases, a user may wish to impute posterior samples of the exact event times for the response variables in their dataset. This may be for the purpose of inferring the distribution of the exact event time for a specific subject, or for passing the data to an analysis tool that does not account for interval-censoring. This can be done with imputeCens(). The arguments are defined as

```
imputeCens(fit, newdata = NULL,
           imputeType = "fullSample",
           samples = 5)
```

The arguments are the same as ic_sample(), except that the newdata "data.frame" must include a pair of columns that contain the lower and upper bounds of the response variable. If newdata is set to NULL, imputeCens() will impute all the rows from the original dataset.

The function survCIs() returns credible intervals for the survival distribution, along with the posterior mean and posterior median estimates. The arguments for survCIs() are

```
survCIs(fit, newdata = NULL,
        p = NULL, q = NULL,
        ci_level = 0.95,
        MC_samps = 40000)
```

Finally, the plot() function accepts the following arguments

```
plot(x, newdata = NULL,
     plot_legend = T, lgdLocation = "topright",
     cis = T, ci_level = 0.9,
     ...)
```

In this case, x should be a fit from ic_bayes(), newdata is a "data.frame" with a set of covariates to determine the survival functions to plot, plot_legend() is a logical argument indicating whether to include a legend with labels provided by the rownames of newdata, cis is a logical indicator for whether credible intervals should be included, ci_level is the credible levels for the credible intervals, and ... is additional arguments to be passed to the base plot() function. Note that if the col argument is supplied, each color will be matched to the corresponding row of newdata. The solid lines plotted are the posterior median survival probabilities, with dashed lines representing the upper and lower limits of the credible interval.

Example analysis

To demonstrate the use of Bayesian regression models in **icenReg**, we will use the miceData dataset included in **icenReg** (Hoel and Walburg, 1972). This dataset examined occurrances of lung cancer in RFM mice (bred for high rates of cancer) kept in two different environments; conventional environment (ce) or germ-free environment (ge). At different ages, mice are sacrificed and examined for lung tumors. If mouse i is inspected at age C_i and a tumor is found, then time of onset is recorded as being in the interval $[0, C_i]$. If no tumor is found, then the time of onset is recorded as being in the interval (C_i, ∞). Note that this form of data is referred to as *current status data*.

We first load the **icenReg** library along with **foreach** (Revolution Analytics and Weston, 2014b) and **doParallel** (Revolution Analytics and Weston, 2014a), which are required to run MCMC chains in parallel.

```
library(icenReg)
library(foreach)
library(doParallel)
```

We then load and examine the `miceData` dataset.

```
data(miceData)
head(miceData)
##   l   u grp
## 1 0 381  ce
## 2 0 477  ce
## 3 0 485  ce
## 4 0 515  ce
## 5 0 539  ce
## 6 0 563  ce
summary(miceData)
##       l             u         grp
## Min.   :  0   Min.   :381.0   ce:96
## 1st Qu.:  0   1st Qu.:809.5   ge:48
## Median :439   Median :  Inf
## Mean   :343   Mean   :  Inf
## 3rd Qu.:644   3rd Qu.:  Inf
## Max.   :986   Max.   :  Inf
```

The column "l" and "u" represent the lower and upper end of the intervals containing the onset time for each mouse. We note that there are 96 mice in the ce group and 48 mice in the ge group. Because current status data is fairly uninformative per subject, this dataset contains limited information about the distribution of time to onset.

For the sake of demonstration, suppose that we had expert information regarding onset of lung cancer. An expert tells us that (a) after two years in the conventional environment, the expert is 50% certain that between 10-30% of the mice will have developed lung tumors and (b) hazard rates are non-decreasing with age. To incorporate (a), we can set a $Beta(\alpha = 1.5, \beta = 5.5)$ prior onto the probability of an event occuring before $t = 730$ for the CE group. For (b), we note the fact that for the Weibull distribution, a shape parameter below 1 implies a decreasing hazard, while a shape parameter above 1 implies an increasing hazard. To enforce a non-decreasing hazard, we will set zero probability mass to the shape parameter below 1. We note that this is an improper prior: we have put a flat prior of the regression coefficient.

To demonstrate how to incorporate this into `ic_bayes()`, we first look at the parameters that will be handed to our prior function. This will be vector of parameters given in the same form and order as returned by `coef()`, for either a Bayesian model or maximum likelihood model (`ic_par()`).

```
mle_fit <- ic_par(cbind(l, u) ~ grp,
                  model = "ph",
                  dist  = "weibull",
                  data = miceData)
coef(mle_fit)
## log_shape log_scale     grpge
## 0.7071843 6.9481420 0.7861709
```

All the syntax used for defining models for `ic_par()` is shared with `ic_bayes()`. In the formula, we define the response by calling `cbind(l,u)`, where `l` and `u` represent the lower and upper ends of the interval. We see that we will be given the baseline log shape parameter, baseline log scale and the coefficient for the dummy variable indicating belonging to the GE group. We then write our log prior density function as such:

```
expertPrior <- function(x){
   # Extracting parameters from input
   shape <- exp(x[1])
   scale <- exp(x[2])
   ge_coef <- x[3]

   # ans is log-density of the prior
   ans <- 0
   # First prior: S(730) ~ beta(1.5, 5.5)
   # Note that we are using a Weibull distribution
   s_730 <- 1 - pweibull(730, shape = shape, scale = scale)
   ans <- ans + dbeta(s_730, 1.5, 5.5, log = TRUE)
   # Second prior: shape >= 1
   if(shape < 1) ans <- -Inf
```

```
   return(ans)
}
```

This prior is then provided to the `ic_bayes()` function. We use **doParallel**'s `registerDoParallel()` to sample from the 4 chains in parallel.

```
myClust <- makeCluster(4)
registerDoParallel(myClust)
bayes_fit <- ic_bayes(cbind(l,u) ~ grp,
                      data = miceData,
                      model = "ph", dist = "weibull",
                      logPriorFxn = expertPrior,
                      useMCores = TRUE)
stopCluster(myClust)
```

We can examine the results using the `summary()` method.

```
summary(bayes_fit)
```

```
## Model:  Bayesian Cox PH
## Baseline:  weibull
## Call: ic_bayes(formula = cbind(l, u) ~ grp, data = miceData, logPriorFxn = expertPrior,
##     model = "ph", dist = "weibull", useMCores = T)
##
##
## Iterations = 1001:5996
## Thinning interval = 5
## Number of chains = 4
## Sample size per chain = 1000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##             Mean     SD Naive SE Time-series SE
## log_shape 0.6337 0.3220 0.005092       0.008220
## log_scale 6.9570 0.1829 0.002892       0.005239
## grpge     0.6916 0.3014 0.004765       0.007971
##
## 2. Quantiles for each variable:
##
##              2.5%    25%    50%    75% 97.5%
## log_shape 0.04927 0.3860 0.6316 0.8685 1.228
## log_scale 6.68745 6.8180 6.9221 7.0694 7.384
## grpge     0.08428 0.4928 0.6967 0.8965 1.274
##
## 3. MAP estimates:
## log_shape log_scale     grpge
##    0.9118    6.8550    0.7530
```

We can access the raw MCMC samples from the `$mcmcList` field. This is a `"mcmcList"` object (Plummer et al., 2006), and as such all the standard **coda** methods can be used directly on this object. For example, if we want traceplots and marginal density estimates of the samples, we can directly call `plot()`. The results are plotted on Figure 1.

```
plot(bayes_fit$mcmcList)
```

We can examine a plot of the posterior survival distribution using the `plot()` method. If we do not provide any new data, the baseline survival distribution will be plotted. This is demonstrated on Figure 2. The solid line is the median posterior survival probability at any given time, with the dashed lines representing upper and lower credible intervals for the survival probabilities.

```
plot(bayes_fit,
     main = "Posterior Baseline Survival",
     col = "blue",
     lwd = 2)
```

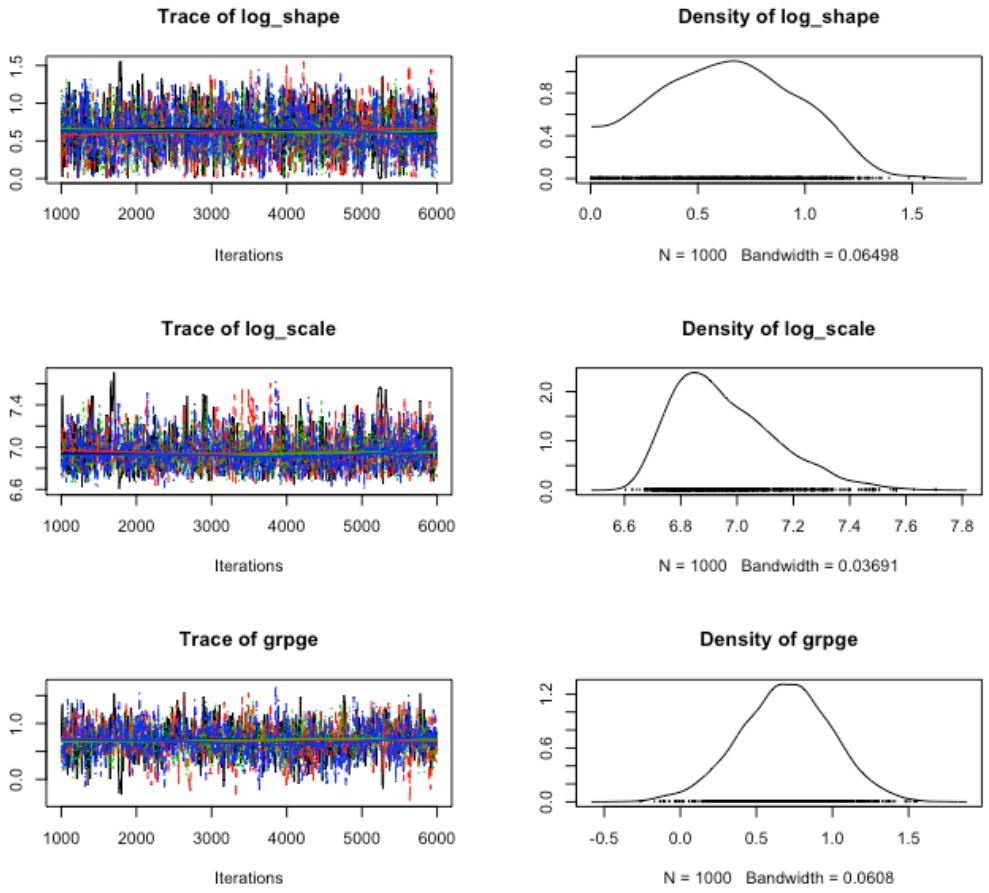


Figure 1: Posterior samples

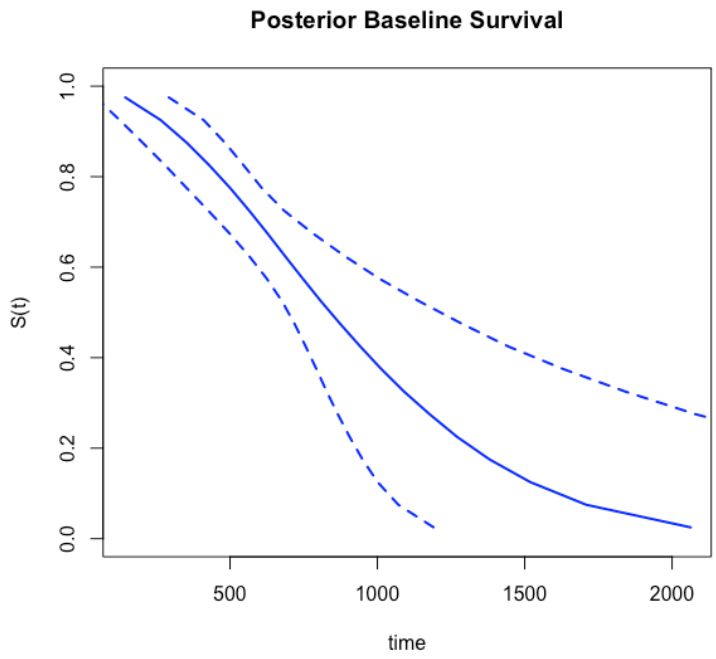


Figure 2: Posterior survival probabilities for baseline distribution

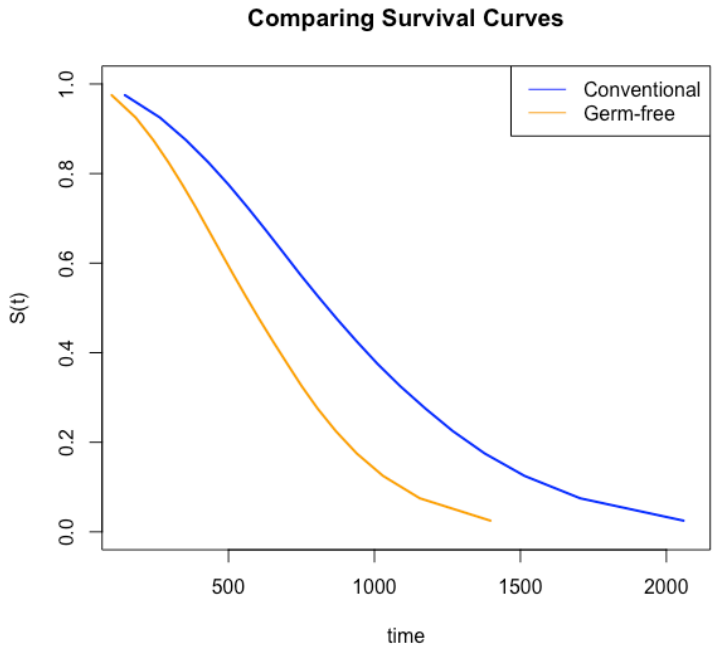


Figure 3: Comparing survival curves between groups

More often, we would like to plot the survival distribution for a given set of covariates, or compare the survival distribution for two different sets of covariates. This can be done by providing a new data set with the covariates of interest, as is done in the code below. If more than one row of data is provided, a legend is added with the row names of the new data set. The location of the legend can be changed using the `lgdLocation` argument. To keep the plot from looking overly cluttered, we will remove the credible bands in this example. The plot can be seen on Figure 3.

```
plot(bayes_fit,
     newdata = newdata,
     main = "Comparing Survival Curves",
     col = c("blue", "orange"),
     lwd = 2,
     cis = F,
     lgdLocation = "topright")
```

Using the `survCIs()` function, we can extract credible intervals for the survival function with a given set of covariates, along with the posterior mean and posterior medians.

```
survCIs(bayes_fit,
        newdata = newdata,
        p = seq(from = 0.1, to = 0.9, by = .2),
        ci_level = 0.95)
## Model call:
##   ic_bayes(formula = cbind(l, u) ~ grp, data = miceData, logPriorFxn = expertPrior,
##     model = "ph", dist = "weibull", useMCores = T)
## Credible Level = 0.95
## Rowname:  Conventional
##      Percentile estimate (mean) estimate (median)     lower     upper
## [1,]        0.1        308.7645          310.4224  165.6698  450.7731
## [2,]        0.3        598.7705          597.9676  471.6719  736.6549
## [3,]        0.5        876.7032          843.0287  700.4579 1242.3366
## [4,]        0.7       1217.7391         1133.4531  849.1271 2013.4947
## [5,]        0.9       1816.5917         1604.8638 1036.7680 3629.9967
## Rowname:  Germ-free
##      Percentile estimate (mean) estimate (median)     lower     upper
## [1,]        0.1        217.1181          211.0069  71.82923  402.7130
## [2,]        0.3        403.0740          405.2846 219.64849  579.6520
```

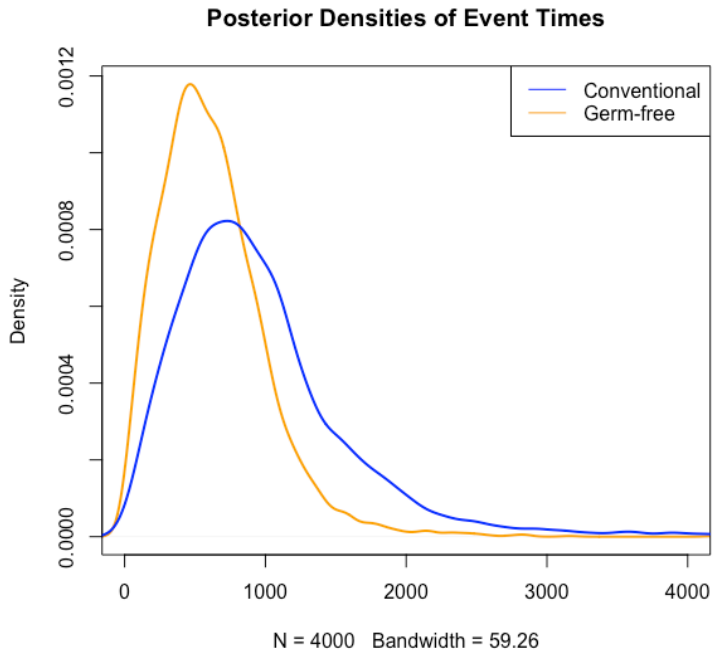


Figure 4: Posterior densities of event times for each group

```
## [3,]        0.5        574.8200          582.3774 393.62514  723.9960
## [4,]        0.7        780.1581          778.2784 618.57113  957.4698
## [5,]        0.9       1131.2446         1088.8539 899.39371 1573.0770
```

Now suppose we wanted to draw posterior samples of the event time distribution for each group. For example, we may wish to construct density plots for event time from each group. This can be done with `ic_samples()` and is demonstrated in the code below. The generated plot can be found on Figure 4.

```
eventTimeSamples <- ic_sample(bayes_fit,
                              newdata = newdata,
                              samples = 4000)
ce_dens <- density(eventTimeSamples["Conventional",],
                   from = 0)
ge_dens <- density(eventTimeSamples["Germ-free",],
                   from = 0)
plot(ge_dens,
     main = "Posterior Densities of Event Times",
     col = "orange",
     xlim = c(0, 4000),
     lwd = 2)
lines(ce_dens,
      col = "blue",
      lwd = 2)
legend("topright",
       c("Conventional", "Germ-free"),
       col = c("blue", "orange"),
       lwd = 1)
```

Finally, we can draw posterior samples of the event time, given that it occurs within some specified interval, with `imputeCens()`. To demonstrate, suppose we were interested in the exact event time for mice in each group that were sacrificed at one year and found to have no tumors, implying the event time was right censored at $t = 365$. This can be expressed in the interval censoring format as $t \in [365, \infty)$. Below, we use `imputeCens()` to draw posterior samples of event times conditional on being greater than 365 and plot the estimate posterior density in Figure 5.

```
# Adding event time intervals
newdata$l <- c(365, 365)
```

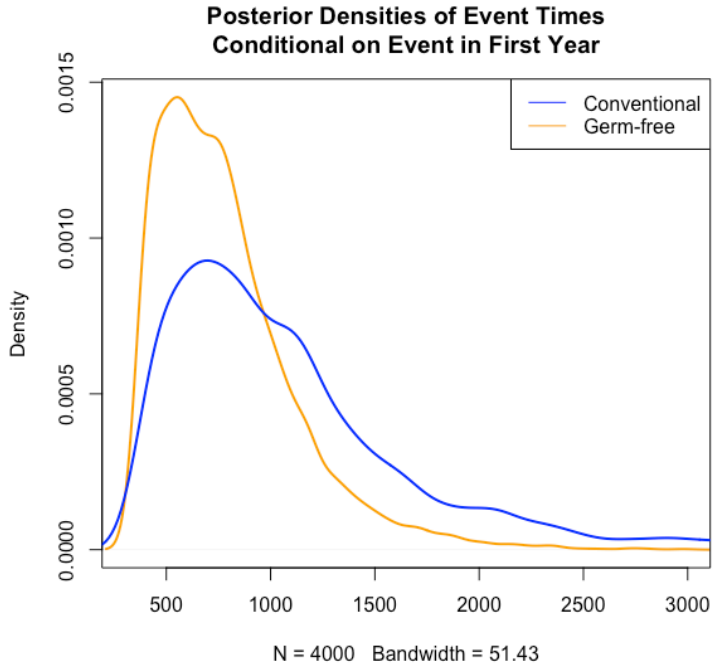


Figure 5: Posterior densities of event time conditional on occurring after first year

```
newdata$u <- c(Inf, Inf)
imputedTimes <- imputeCens(bayes_fit,
                           newdata = newdata,
                           samples = 4000)

ce_dens <- density(imputedTimes["Conventional",],
                   from = 365)
ge_dens <- density(imputedTimes["Germ-free",],
                   from = 365)

plot(ge_dens,
     main = "Posterior Densities of Event Times\nConditional on Event in First Year",
     col = "orange",
     xlim = c(300, 3000),
     ylim = c(0, 0.0015),
     lwd = 2)
lines(ce_dens,
      col = "blue",
      lwd = 2)
legend("topleft",
       c("Conventional", "Germ-free"),
       col = c("blue", "orange"),
       lwd = 1)
```

Summary

Interval-censoring occurs when event times are not known exactly, but rather only up to an interval. Naturally, this results in less informative data than if the event time were observed exactly. The potentially weakly informative data further motivates using prior information about the data generating process to provide a more informative analysis of a given data set. Bayesian methodology provides a straightforward framework for incorporating such prior information. The addition of `ic_bayes()` to the **icenReg** package allows for simple, efficient interval-censored regression models with generic user provided prior distributions and a variety of tools to simplify analyses.

Acknowledgements

The **icenReg** package uses **Rcpp** (Eddelbuettel and Francois, 2011) to interface R objects with C++. Linear algebra at the C++ level is handled by the Eigen library (Guennebaud et al., 2010), which is interfaced with **RcppEigen** (Bates and Eddelbuettel, 2013).

Bibliography

C. Anderson-Bergman. icenReg: Regression models for interval censored data. *Journal of Statistical Software*, 81(12), 2017. [p487]

D. Bates and D. Eddelbuettel. Fast and elegant numerical linear algebra using the RcppEigen package. *Journal of Statistical Software*, 52(5):1–24, 2013. URL https://doi.org/10.18637/jss.v052.i05. [p497]

D. Eddelbuettel and R. Francois. Rcpp: Seamless R and C++ integration. *Journal of Statistical Software*, 40(8):1–18, 2011. URL http://www.jstatsoft.org/v40/i08/. [p497]

D. M. Finkelstein. A proportional hazards model for interval-censored failure time data. *Biometrika*, 42:845–854, 1986. [p487]

A. Gelman, G. O. Roberts, W. R. Gilks, and others. Efficient metropolis jumping rules. *Bayesian statistics*, 5(599-608):42, 1996. [p489]

J. Gruger, R. Kay, and M. Schumacher. The validity of inferences based on incomplete observations in disease state models. *Biometrics*, pages 595–605, 1991. [p487]

G. Guennebaud, B. Jacob, and others. *Eigen V3*, 2010. [p497]

G. Gómez, M. L. Calle, and R. Oller. Frequentist and bayesian approaches for interval-censored data. *Statistical Papers*, 45(2):139–173, 2004. URL https://doi.org/10.1007/bf02777221. [p487]

H. Haario, E. Saksman, J. Tamminen, and others. An adaptive metropolis algorithm. *Bernoulli*, 7(2): 223–242, 2001. URL https://doi.org/10.2307/3318737. [p489]

D. G. Hoel and H. E. Walburg. Statistical analysis of survival experiments. *The Annals of Statistics*, 18: 1259– 1294, 1972. [p490]

J. Huang. Efficient estimation for the proportional hazards model with interval censoring. *The Annals of Statistics*, 24:540–568, 1995. [p487]

E. Kaplan and P. Meier. Nonparametric estimation from incomplete observations. *Communications in Statistics - Theory and Methods*, 27:1961 – 1977, 1958. [p487]

J. Lindsey. A study of interval censoring in parametric regression models. *Lifetime data analysis*, 4(4): 329–354, 1998. [p487]

M. Ng. A modification of peto's nonparametric estimation of survival curves for interval-censored data. *Biometrics*, 58:439–442, 2002. URL https://doi.org/10.1111/j.0006-341x.2002.00439.x. [p487]

W. Pan. Extending the iterative convex minorant algorithm to the Cox model for interval-censored data. *Journal of Computational and Graphical Statistics*, 8:109–120, 1999. [p487]

M. Plummer, N. Best, K. Cowles, and K. Vines. CODA: Convergence diagnosis and output analysis for MCMC. *R News*, 6(1):7–11, 2006. URL https://journal.r-project.org/archive/. [p492]

D. Rabinowitz, A. Tsiatis, and J. Aragon. Regression with interval-censored data. *Biometrika*, 82(3): 501–513, 1995. [p487]

Revolution Analytics and S. Weston. *doParallel: Foreach Parallel Adaptor for the Parallel Package*, 2014a. URL http://CRAN.R-project.org/package=doParallel. R package version 1.0.8. [p490]

Revolution Analytics and S. Weston. *foreach: Foreach Looping Construct for R*, 2014b. URL http://CRAN.R-project.org/package=foreach. R package version 1.4.2. [p490]

A. Rossini and A. Tsiatis. A semiparametric proportional odds regression model for the analysis of current status data. *Journal of the American Statistical Assocation*, 91:713–721, 1996. [p487]

B. Sen and G. Xu. Model based bootstrap methods for interval censored data. *Computational Statistics & Data Analysis*, 81:121–129, 2015. URL https://doi.org/10.1016/j.csda.2014.07.007. [p487]

J. Sun. *The Statistical Analysis of Interval-Censored Failure Time Data*. Springer-Verlag, 2007. URL https://doi.org/10.1007/0-387-37119-2. [p487]

J. Tobin. Estimation of relationships for limited dependent variables. *Econometrica: journal of the Econometric Society*, pages 24–36, 1958. [p487]

B. Turnbull. The empirical distribution with arbitrarily grouped and censored data. *Journal of the Royal Statistical Society B*, 38:290–295, 1976. [p487]

Clifford Anderson-Bergman
Sandia National Labs
7011 East Avenue
USA
ciande@sandia.gov

openEBGM: An R Implementation of the Gamma-Poisson Shrinker Data Mining Model

by Travis Canida and John Ihrie

Abstract We introduce the R package **openEBGM**, an implementation of the Gamma-Poisson Shrinker (GPS) model for identifying unexpected counts in large contingency tables using an empirical Bayes approach. The Empirical Bayes Geometric Mean (EBGM) and quantile scores are obtained from the GPS model estimates. **openEBGM** provides for the evaluation of counts using a number of different methods, including the model-based disproportionality scores, the relative reporting ratio (RR), and the proportional reporting ratio (PRR). Data squashing for computational efficiency and stratification for confounding variable adjustment are included. Application to adverse event detection is discussed.

Introduction

Contingency tables are a common way to summarize events that depend on categorical factors. DuMouchel (1999) describes an application of contingency tables to adverse event reporting databases. The rows represent products, such as drugs or food, and the columns represent adverse events. Each cell in the table represents the number of reports that mention both that product and that event. Overreported product-event pairs might be of interest.

One naïve approach for analyzing counts in this table is to calculate the observed *relative reporting ratio* (*RR*)—which DuMouchel (1999) calls *relative report rate*—for each cell. While *RR* is easy to compute, it has the drawback of being highly variable, and thus unreliable, for small counts (DuMouchel, 1999; Madigan et al., 2011). To combat the high variability of *RR*, DuMouchel (1999) created an *empirical Bayes* (EB) data mining model for finding "interestingly large" counts. The model-based EB scores are measures of disproportionality, similar to *RR*. However, the model uses Bayesian shrinkage to correct for the high variability in *RR* associated with small counts.

After the EB model was introduced, Evans et al. (2001) created another disproportionality approach called the *proportional reporting ratio* (*PRR*). The *PRR* compares "the proportion of all reactions to a drug which are for a particular medical condition of interest...to the same proportion for all drugs in the database" (Evans et al., 2001). Like *RR*, *PRR* is easy to calculate, but has the same drawback of high variability (Madigan et al., 2011). Almenoff et al. (2006) found that *PRR* finds more false positives than DuMouchel's model; however, it also finds more true positives. Another difference between DuMouchel's model and the *PRR* metric is that while DuMouchel's model considers the event of interest when calculating the expected count for the pair, *PRR* does not (Duggirala et al., 2015).

The **openEBGM** (Ihrie and Canida, 2017) package implements the model described in DuMouchel (1999) and the computational efficiency improvement techniques described in DuMouchel and Pregibon (2001). Our goal was to create a general-purpose, flexible, efficient implementation of DuMouchel's model, but we also include *PRR* in case users want to compare the results. Other disproportionality approaches exist (Madigan et al., 2011), but our goal for this article is not to provide an exhaustive list or to compare DuMouchel's model to other methods. Instead, we focus on our implementation of DuMouchel's model and provide some comparisons with other R packages. Users can refer to **openEBGM**'s vignettes to follow future developments and modifications to the package.

While **openEBGM** was developed mainly for adverse event detection, we structured the code to be general enough for any similar application. DuMouchel describes multiple applications of his EB model, which can find statistical associations but not necessarily causal relationships (DuMouchel, 1999; DuMouchel and Pregibon, 2001). Thus, the model is used primarily for signal detection. Regardless of the application, subject-matter experts should investigate using other means to determine if any causal relationships might exist.

Attenuating the relative reporting ratio

The relative reporting ratio compares a table cell's actual count, N, to its expected count, E, under the assumption of independence between rows and columns: $RR = {}^{N}/_{E}$. Thus, $RR = 1$ if the actual count is equal to the expected count. When $RR > 1$, more events are observed than expected. Therefore, large RR scores may indicate interesting row-column pairs. This approach to analyzing contingency

table counts works well for large cell counts, but small cell counts result in unstable RR values. Since the expected counts can be close to zero, RR can be very large (DuMouchel, 1999) for small actual counts which could have easily occurred simply by chance. The EB approach shrinks large RRs with small Ns to a value much closer to 1. The shrinkage is less for larger counts. Shrinkage produces more reliable results than the simple RR score.

Model description

The EB model uses a Poisson(μ_{ij}) data distribution (i.e. likelihood) for contingency table cell counts in row i and column j, where $i = 1, \ldots, I$ and $j = 1, \ldots, J$. We are interested in the ratio $\lambda_{ij} = {}^{\mu_{ij}}/_{E_{ij}}$. The prior distribution on λ (Equation 1) is a mixture of two gamma distributions, resulting in gamma-mixture posterior distributions (Equation 2). Thus, the model is sometimes referred to as the *Gamma-Poisson Shrinker* (GPS) model.

The prior is a single distribution that models all cell counts; however, each cell has a separate posterior distribution determined both by that cell's actual and expected counts and by the distribution of actual and expected counts in the entire table. The λ_{ij}s are assumed to come from the prior distribution with hyperparameter $\theta_{prior} = (\alpha_1, \beta_1, \alpha_2, \beta_2, P)$, where P is the mixture fraction. The posterior distributions have parameters $\theta_{post,ij} = \left(\alpha_1 + n_{ij},\ \beta_1 + E_{ij},\ \alpha_2 + n_{ij},\ \beta_2 + E_{ij},\ Q_{n,ij}\right)$, where $Q_{n,ij}$ are the mixture fractions. The posterior distributions are, in a sense, Bayesian representations of the relative reporting ratios (note the similarity in the equations $RR_{ij} = {}^{N_{ij}}/_{E_{ij}}$ and $\lambda_{ij} = {}^{\mu_{ij}}/_{E_{ij}}$). DuMouchel (1999, Eqs. 4, 7) summarized the model with row and column subscripts suppressed:

$$prior:\ \pi(\lambda;\ \alpha_1, \beta_1, \alpha_2, \beta_2, P) = Pg(\lambda;\ \alpha_1, \beta_1) + (1 - P)g(\lambda;\ \alpha_2, \beta_2) \tag{1}$$

$$posterior:\ \lambda | N = n \sim \pi(\lambda;\ \alpha_1 + n,\ \beta_1 + E,\ \alpha_2 + n,\ \beta_2 + E,\ Q_n) \tag{2}$$

where $g(\cdot) \sim \Gamma(\alpha, \beta)$ is the gamma distribution with shape and rate parameters α and β, respectively.

Model-based scores

The EB scores are based on the posterior distributions and used in lieu of RR. The *Empirical Bayes Geometric Mean* ($EBGM$) score is the geometric mean of a posterior distribution. The 5^{th} and 95^{th} percentiles of a posterior distribution create a two-sided 90% credibility interval for the EB disproportionality score. Alternatively, since we are primarily interested in the lower bound, we could create a one-sided 95% credibility interval (Szarfman et al., 2002). Bayesian shrinkage causes the EB scores to be smaller than RR scores, but the shrinkage amount decreases as N increases (Figure 1).

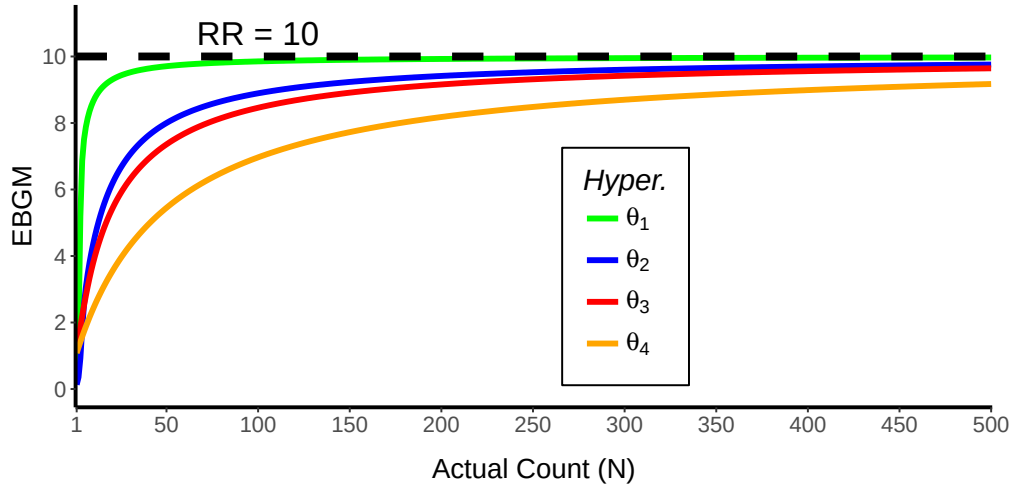


Figure 1: Asymptotic behavior of $EBGM$ for various hyperparameter values. $\theta_1 = (.01, .1, 5, 20, .5)$; $\theta_2 = (.1, 1.2, .4, 8, .05)$; $\theta_3 = (2, 2, 5, 5, .8)$; $\theta_4 = (5, 5, 5, 5, .5)$

Example application

The U.S. Food and Drug Administration (FDA) monitors regulated products (such as drugs, vaccines, food and cosmetics) for adverse events. For food, FDA's Center for Food Safety and Applied Nutrition (CFSAN) mines data from the CFSAN Adverse Events Reporting System (CAERS) (U.S. Food and Drug Administration, 2017). Adverse events are tabulated in contingency tables (separate tables for separate product types). The FDA previously used the GPS model to analyze these tables (Szarfman et al., 2002). Later, we show an example of GPS using the CAERS database.

Existing open-source implementations

Existing open-source implementations of the GPS model include the R packages **PhViD** (Ahmed and Poncet, 2016) and **mederrRank** (Venturini and Myers, 2015). Each of the existing implementations has its own feature set and drawbacks.

The **PhViD** package does not offer data squashing, stratification, or a means of counting events from raw data. Nor does it account for unique event identifiers to eliminate double counting of reports when calculating expected counts. **PhViD** does, however, offer features (Ahmed et al., 2009) not found in **openEBGM**, such as false discovery rate decision rules. **PhViD**'s approach to implementation of the GPS model seems focused on adding multiple comparison techniques. We used the **PhViD** package as a starting point to write our own code. However, we focused on creating a tool that simply implements the GPS model in a flexible and efficient manner. See Appendix H.10 for a timing comparison between **openEBGM** and **PhViD**.

The **mederrRank** package adapts the GPS model to a specific medication error application. Although **mederrRank** does not appear to offer data squashing or a means of counting events from raw data, it does allow for stratification by hospital (Venturini et al., 2017). Hyperparameters are estimated using an expectation-maximization (EM) algorithm, whereas **openEBGM** uses a gradient-based approach. **mederrRank** seems focused on comparing the GPS model to the Bayesian hierarchical model suggested by Venturini et al. (2017).

Main functions

Table 1 shows a listing of **openEBGM**'s main functions:

Function	Description
`processRaw()`	Counts unique reports; calculates expected counts, RR, and PRR.
`squashData()`	Squashes points (N, E) to reduce computational burden for hyperparameter estimation.
`autoHyper()`	Calculates hyperparameter estimates.
`ebgm()`	Calculates $EBGM$ scores.
`quantBisect()`	Calculates quantile scores.
`ebScores()`	Creates an object with $EBGM$ and quantile scores.

Table 1: The main functions in **openEBGM**.

Hardware and software specifications

We ran the code presented in this article on a 64-bit Windows 7 machine with 128 GB of DDR3 RAM clocked at 1866 MHz and two 6-core Intel(R) Xeon(R) CPU E5-2630 v2 @ 2.60 GHz processors. We used R v3.4.1, **openEBGM** v0.3.0, and **PhViD** v1.0.8.

Data preparation

Use of the **openEBGM** package requires that the data be formatted in a tidy way (Wickham, 2014); the data must adhere to the standards of one column per variable and one row per observation. The columns may be of class `"factor"`, `"character"`, `"integer"` or `"numeric"`. Although zero counts can

exist in the contingency table, missing values are not allowed in the unprocessed data and must be replaced with appropriate values or removed before using **openEBGM**'s functions.

Column names

The input data frame must contain some specific columns. In particular, these are: 'var1', 'var2' and 'id'. 'var1' and 'var2' are simply the row and column variables of the contingency table, respectively. The identifier ('id') column allows **openEBGM** to properly handle marginal totals and remove the effect of double counting. For example, many CAERS reports contain multiple products and events. However, if an application does use a unique cell for each event of interest, the user can then create a column of unique sequential identifiers with `df$id <-1:nrow(df)`, where `df` is the input data frame.

In addition to the columns mentioned above, a column which stratifies the data (by, for example, gender, age, race, etc.) may be of interest as it can help reduce the effects of confounding variables (DuMouchel, 1999). If stratification is used, any column whose name contains the case-sensitive substring 'strat' will be treated as a stratification variable. If a continuous variable (e.g. age) is used for stratification, it should be appropriately categorized so that it is no longer continuous. Additional columns besides those mentioned above are allowed, but ignored by **openEBGM**.

CAERS data example

One example use case of the **openEBGM** package is adverse event signal detection. This application is utilized by FDA's CFSAN with data from CAERS, which collects adverse event reports on consumer products such as foods and dietary supplements and is freely available and updated quarterly (U.S. Food and Drug Administration, 2017). An example of data preparation with the CAERS dataset is shown below. We start by downloading the data from FDA's website and selecting dietary supplement (*Industry Code 54*) reports before the year 2017.

```
> Sys.setlocale(locale = "C")  #locale can affect sorting order, etc.
> site   <- "https://www.fda.gov/downloads/Food/ComplianceEnforcement/UCM494018.csv"
> dat    <- read.csv(site, stringsAsFactors = FALSE, strip.white = TRUE)
> dat$yr <- dat$RA_CAERS.Created.Date
> dat$yr <- substr(dat$yr, start = nchar(dat$yr) - 3, stop = nchar(dat$yr))
> dat$yr <- as.integer(dat$yr)
> dat    <- dat[dat$PRI_FDA.Industry.Code == 54 & dat$yr < 2017, ]
```

Next we rename the columns:

```
> dat$var1 <- dat$PRI_Reported.Brand.Product.Name
> dat$var2 <- dat$SYM_One.Row.Coded.Symptoms
> dat$id   <- dat$RA_Report..
> dat$strat_gen <- dat$CI_Gender
> vars <- c("id", "var1", "var2", "strat_gen")
> dat  <- dat[, vars]
```

Gender needs to be recategorized:

```
> dat$strat_gen <- ifelse(dat$strat_gen %in% c("Female", "Male"),
+                         dat$strat_gen, "unknown")
```

We can remove rows with non-ASCII characters in case they cause problems in certain locales:

```
> dat2 <- dat[!dat$var1 %in% tools::showNonASCII(dat$var1), ]
```

Each row has one product, but multiple adverse events:

```
> head(dat2, 3)
```

```
     id                    var1                                        var2 strat_gen
7 65353     HERBALIFE RELAX NOW PARANOIA, PHYSICAL EXAMINATION, DELUSION    Female
8 65353 HERBALIFE TOTAL CONTROL PARANOIA, PHYSICAL EXAMINATION, DELUSION    Female
9 65354                 YOHIMBE                 BLOOD PRESSURE INCREASED      Male
```

We can use the **tidyr** (Wickham, 2017) package to reshape the data:

```
> dat_tidy <- tidyr::separate_rows(dat2, var2, sep = ", ")
> dat_tidy <- dat_tidy[, vars]
```

```
> head(dat_tidy, 3)

     id                var1                 var2 strat_gen
1 65353 HERBALIFE RELAX NOW             PARANOIA    Female
2 65353 HERBALIFE RELAX NOW PHYSICAL EXAMINATION    Female
3 65353 HERBALIFE RELAX NOW             DELUSION    Female
```

Data cleaning

As a small digression, it is worth noting that the quality of the data can greatly impact the results of the **openEBGM** package. For instance, **openEBGM** considers the drug-symptom pairs 'drug1-symp1', 'Drug1-symp1', and 'DRUG1-symp1' to be distinct *var1-var2* pairs. Even minor differences (e.g. spelling, capitalization, spacing, etc.) can influence the results. Thus, we recommend cleaning data before using **openEBGM** to help improve hyperparameter estimates and signal detection in general.

While the issue above could be fixed simply by using the R functions `tolower()` or `toupper()`, other issues such as punctuation, spacing, string permutations in the 'var1' or 'var2' variables, or some combination of these must first be vetted and repaired by the user.

Counts and simple disproportionality measures

openEBGM contains functions which take the prepared data and output *var1-var2* counts, as well as some simple disproportionality measures for these *var1-var2* pairs. As mentioned in other sections, there are a number of disproportionality measures of interest that this package concerns, including the EB scores, *RR* as well as the *PRR*. The `processRaw()` function in the **openEBGM** package takes the prepared data and outputs the *var1-var2* pair counts (N), the expected number of counts for the *var1-var2* pair (E), as well as the *RR* and *PRR* for the *var1-var2* pair.

Data processing function

The function `processRaw()` takes the prepared data and returns a data frame with one row for each *var1-var2* pair. Each row contains the simple disproportionality measures (*RR*, *PRR*) as well as the counts (N, E) for that pair. In the case that the calculation for *PRR* involves division by zero, a default value of 'Inf' is returned.

The user may decide if stratification should be used to calculate E and whether zero counts (i.e. a given *var1-var2* pair is never observed in the data) should be included. Stratification affects *RR*, but not *PRR*. For the purpose of reducing computational burden, zero counts should not typically be included for hyperparameter estimation; however, they may help when convergence issues are encountered. If included, the points should be squashed (discussed in later sections) in order to reduce the computation time. These zero counts should not typically be used for EB scores since they are meaningless for studying larger-than-expected counts (which is usually the goal).

Data processing example

Here, the **tidyr** package is only needed for the pipe (`%>%`) operator.

```
> library("tidyr")
> library("openEBGM")
> data("caers")  #small subset of publicly available CAERS data
> head(caers, 3)

      id     var1                    var2 strat1
1 147289 PREVAGEN          BRAIN NEOPLASM Female
2 147289 PREVAGEN CEREBROVASCULAR ACCIDENT Female
3 147289 PREVAGEN          RENAL DISORDER Female
```

First, we process the data without using stratification or zeroes:

```
> processRaw(caers) %>% head(3)

                    var1                  var2 N            E      RR      PRR
1         1-PHENYLALANINE  HEART RATE INCREASED 1 0.0360548272   27.74    27.96
2 11 UNSPECIFIED VITAMINS               ASTHMA 1 0.0038736591  258.15   279.58
3 11 UNSPECIFIED VITAMINS CARDIAC FUNCTION TEST 1 0.0002979738 3356.00      Inf
```

Now, using stratification:

```
> processRaw(caers, stratify = TRUE) %>% head(3)

stratification variables used: strat1
there were 3 strata

                     var1                   var2 N            E      RR      PRR
1         1-PHENYLALANINE  HEART RATE INCREASED 1 0.0287735849   34.75    27.96
2 11 UNSPECIFIED VITAMINS               ASTHMA 1 0.0047169811  212.00   279.58
3 11 UNSPECIFIED VITAMINS CARDIAC FUNCTION TEST 1 0.0004716981 2120.00      Inf
```

Finally, we use stratification and a cap on the number of strata (useful for preventing excessive data slimming, especially with uncategorized continuous variables):

```
> processRaw(caers, stratify = TRUE, max_cats = 2) %>% head(3)

stratification variables used: strat1
Error in .checkStrata_processRaw(data, max_cats) :
  at least one stratification variable contains more than 2 categories --
  did you remember to categorize stratification variables?
  if you really need more categories, increase 'max_cats'
```

Hyperparameter estimation

The marginal distribution of each N_{ij} is a negative binomial mixture (DuMouchel, 1999, p. 180, Eq. 5) and is a function of the hyperparameter (θ_{prior}), the actual observed counts (n_{ij}), and the expected counts (E_{ij}). The prior distribution's maximum likelihood hyperparameter estimate, $\hat{\theta}_{prior}$, is obtained by minimizing the negative log-likelihood from these marginal distributions. Global optimization is needed to estimate θ_{prior}. **openEBGM**'s approach to global optimization is to simply use local optimization with multiple starting points. The user can manually optimize the likelihood function using another approach if desired. **openEBGM**'s optimization functions are wrappers for functions from the **stats** package. (**Note:** Results might vary slightly by operating system and version of R.) Users are encouraged to explore many optimization approaches because the accuracy of a global optimization result is difficult to verify, convergence is not guaranteed, and some approaches may outperform others.

Data squashing

DuMouchel and Pregibon (2001) used a method of reducing computational burden they called *data squashing*, which reduces the number of data points (n_{ij}, E_{ij}) used for hyperparameter estimation. A very large table with I rows and J columns can require immense computational resources. Data squashing reduces this set of points to a much smaller set of K points (n_k, E_k, W_k), where $k = 1, \ldots, K < I \times J$ and W_k is the weight of the k^{th} squashed point.

For a given n, `squashData()` bins points with similar Es and uses the average E within each bin as the expected count for that squashed point. The new points are weighted by bin size. For example, the points $(1, 1.1)$ and $(1, 1.3)$ could be squashed to $(1, 1.2, 2)$. To minimize information loss, we recommend only squashing points in close proximity. By default, `squashData()` does not squash the points with the highest Es for a given n since those points tend to have more variability (at least for small n; see Figure 2). The user can call `squashData()` repeatedly on different values of n (e.g. $n = 1$, then $n = 2$). We recommend squashing the data to less than 20,000 points.

Likelihood functions

The likelihood function for θ_{prior} (DuMouchel, 1999, p. 181, Eq. 12) must be adjusted (DuMouchel and Pregibon, 2001) when using data squashing or removing small counts (often just zeroes) from the estimation procedure, so **openEBGM** offers 4 likelihood functions: `negLL()`, `negLLsquash()`, `negLLzero()`, and `negLLzeroSquash()`. Since the GPS model was developed to study large datasets, `negLLsquash()` will usually be used since it allows for both data squashing and the removal of smaller counts. The user will not call the likelihood functions directly if using the wrapper functions described in the next section.

Optimization wrapper functions

`exploreHypers()` requires the user to choose one or more starting points (i.e. guesses) for θ_{prior}. For each starting point, the corresponding estimate, $\hat{\theta}_{prior}$, is returned if the algorithm converges. Examining estimates from multiple starting points allows the user to study the consistency of the results and reduces the chances of false convergence or getting trapped in a local minimum. Hyperparameter estimates are calculated using an implementation of one of three Newton-like or quasi-Newton methods from the **stats** package: `nlminb()`, `nlm()`, or `optim()` (using `method = "BFGS"` for `optim()`). The `N_star` argument defines the smallest actual count (usually 1) used for hyperparameter estimation (DuMouchel and Pregibon, 2001). Setting the `std_errors` argument to `TRUE` calculates estimated standard errors using the observed Fisher information as discussed in DuMouchel (1999, p. 183).

`autoHyper()` uses a semi-automated approach that returns a list including a final $\hat{\theta}_{prior}$ after running some verification checks on the estimates returned by `exploreHypers()`. From the solutions that converge inside the parameter space, `autoHyper()` chooses the $\hat{\theta}_{prior}$ with the smallest negative log-likelihood. By default, at least one other convergent solution must be similar to the chosen $\hat{\theta}_{prior}$ (i.e. within a specified tolerance defined by the `tol` argument). Each of the three methods available in `exploreHypers()` are attempted in sequence until these conditions are satisfied. If all methods are exhausted without consistent convergence, `autoHyper()` returns an error message. Setting the `conf_ints` argument to `TRUE` returns standard errors and asymptotic normal confidence intervals. `exploreHypers()` is called internally, so `autoHyper()` may be used without first calling `exploreHypers()`.

Hyperparameter estimation example

We start by counting item pairs and squashing the counts twice:

```
> proc <- processRaw(caers)
```

Figure 2 illustrates why squashing the largest Es for each N could result in a large loss of information.

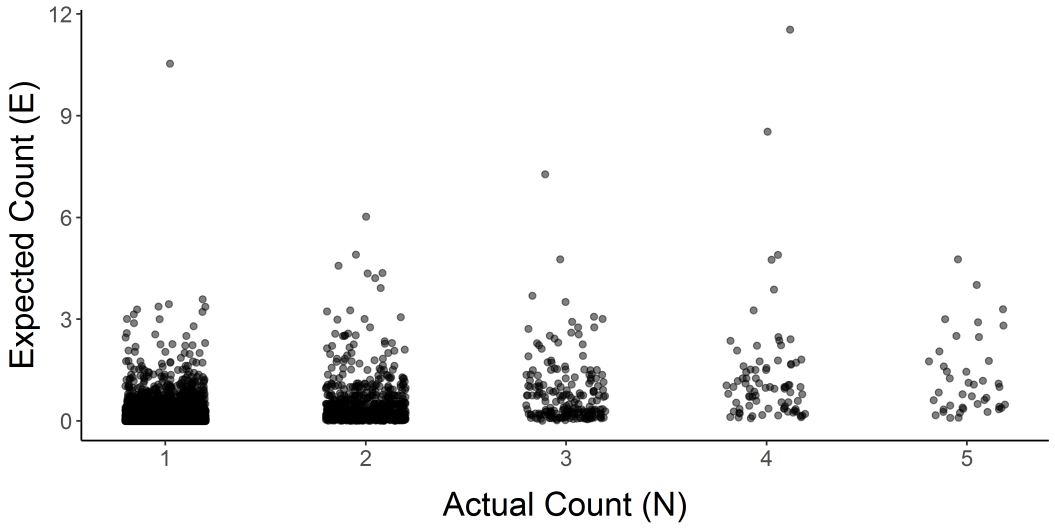


Figure 2: Larger expected counts are generally more spread out; by default, they are not squashed (to prevent information loss).

```
> squashed <- squashData(proc)
> squashed <- squashData(squashed, count = 2, bin_size = 10)
> head(squashed, 3); tail(squashed, 2)

  N            E weight
1 1 0.0002979738     50
2 1 0.0002979738     50
3 1 0.0002979738     50
     N        E weight
946 53 14.30095      1
947 54 16.05721      1
```

We can optimize the likelihood function directly:

```
> theta_init1 <- c(alpha1 = 0.2, beta1 = 0.1, alpha2 = 2, beta2 = 4, p = 1/3)
> stats::nlminb(start = theta_init1, objective = negLLsquash,
+               ni = squashed$N, ei = squashed$E, wi = squashed$weight)$par

    alpha1      beta1     alpha2      beta2          p
3.25120698 0.39976727 2.02695588 1.90892701 0.06539504
```

Or we can use **openEBGM**'s wrapper functions:

```
> theta_init2 <- data.frame(
+   alpha1 = c(0.2, 0.1, 0.5),
+   beta1  = c(0.1, 0.1, 0.5),
+   alpha2 = c(2,   10,  5),
+   beta2  = c(4,   10,  5),
+   p      = c(1/3, 0.2, 0.5)
+ )

> exploreHypers(squashed, theta_init = theta_init2, std_errors = TRUE)

$estimates
  guess_num   a1_hat    b1_hat   a2_hat   b2_hat      p_hat ... minimum
1         1 3.251207 0.3997673 2.026956 1.908927 0.06539504 ... 4161.921
2         2 3.251187 0.3997670 2.026961 1.908933 0.06539553 ... 4161.921
3         3 3.251243 0.3997702 2.026965 1.908933 0.06539509 ... 4161.921

$std_errs
  guess_num    a1_se     b1_se     a2_se     b2_se       p_se
1         1 2.280345 0.1434897 0.4515784 0.4328318 0.03575796
2         2 2.280247 0.1434851 0.4515718 0.4328231 0.03575709
3         3 2.280786 0.1435108 0.4516005 0.4328767 0.03576430

There were 11 warnings (use warnings() to see them)

> (theta_hat <- autoHyper(squashed, theta_init = theta_init2, conf_ints = TRUE))

$method
[1] "nlminb"

$estimates
    alpha1      beta1     alpha2      beta2          P
3.25118662 0.39976698 2.02696130 1.90893277 0.06539553

$conf_int
       pt_est     SE   LL_95  UL_95
a1_hat 3.2512 2.2802 -1.2180 7.7204
b1_hat 0.3998 0.1435  0.1185 0.6810
a2_hat 2.0270 0.4516  1.1419 2.9120
b2_hat 1.9089 0.4328  1.0606 2.7573
p_hat  0.0654 0.0358 -0.0047 0.1355

$num_close
[1] 2

$theta_hats
  guess_num   a1_hat    b1_hat   a2_hat   b2_hat      p_hat ... minimum
1         1 3.251207 0.3997673 2.026956 1.908927 0.06539504 ... 4161.921
2         2 3.251187 0.3997670 2.026961 1.908933 0.06539553 ... 4161.921
3         3 3.251243 0.3997702 2.026965 1.908933 0.06539509 ... 4161.921

There were 11 warnings (use warnings() to see them)
```

Warnings are often produced. Global optimization results are not guaranteed even when no warnings are produced and convergence is reached. The starting points can have an impact on the results. DuMouchel (1999) and DuMouchel and Pregibon (2001) provide some recommendations for starting points. The user must decide if the results seem reasonable.

EB disproportionality scores

The empirical Bayes scores are obtained from the posterior distributions. Thus, the posterior functions calculate the EB scores. Casual users will rarely call these functions directly since they are called internally by the `ebScores()` function described in the next section. However, we still exported these functions for users that want added flexibility.

Posterior functions

`Qn()` calculates the mixture fractions for the posterior distributions using the hyperparameter estimates ($\hat{\theta}_{prior}$) and the counts (n_{ij}, E_{ij}). The values returned by `Qn()` correspond to "the posterior probability that λ came from the first component of the mixture, given $N = n$" (DuMouchel, 1999, p. 180). Recall that a posterior distribution exists for each cell in the table. Thus, `Qn()` returns a numeric vector with the same length as the number of (usually non-zero) *var1-var2* combinations. Use the counts returned by `processRaw()`—not the squashed dataset—as inputs for `Qn()`.

`ebgm()` finds the *EBGM* scores. These scores replace the *RR* scores and represent the geometric means of the posterior distributions. Scores much larger than 1 indicate *var1-var2* pairs that occur at a higher-than-expected rate. `quantBisect()` finds the quantile scores (i.e. credibility limits) using the bisection method and can calculate any percentile between 1 and 99. Low percentiles (e.g. 5^{th} or 10^{th}) can be used as conservative disproportionality scores.

EB scores example

Continuing with the previous example:

```
> theta_hats <- theta_hat$estimates
> qn <- Qn(theta_hats, N = proc$N, E = proc$E)
> proc$EBGM <- ebgm(theta_hats, N = proc$N, E = proc$E, qn = qn)
> proc$QUANT_05 <- quantBisect(5, theta_hat = theta_hats,
+                              N = proc$N, E = proc$E, qn = qn)
> proc$QUANT_95 <- quantBisect(95, theta_hat = theta_hats,
+                              N = proc$N, E = proc$E, qn = qn)
> head(proc, 3)
```

```
                    var1                  var2 ...      RR ... EBGM QUANT_05 QUANT_95
1         1-PHENYLAL...  HEART RATE INCREASED ...   27.74 ... 2.23     0.49    13.85
2 11 UNSPECIFIED VIT...                ASTHMA ...  258.15 ... 2.58     0.52    15.78
3 11 UNSPECIFIED VIT... CARDIAC FUNCTION TEST ... 3356.00 ... 2.63     0.52    16.02
```

Object-oriented features

In addition to the capabilities described above, **openEBGM** can create S3 objects of class `"openEBGM"` to aid in the calculation and inspection of disproportionality scores, as well as reduce the number of direct function calls needed. When using an `"openEBGM"` object, the generic functions `print()`, `summary()` and `plot()` dispatch to methods written specifically for objects of this class.

Object creation

The `ebScores()` function instantiates an object, which includes the EB scores (*EBGM* and chosen posterior quantile scores). After calculating the hyperparameters, `ebScores()` can instantiate an object by calling the posterior distribution functions (previous section), thus simplifying the analyst's work. Generic functions can then be used to quickly review the results. An example is provided below.

```
> proc2 <- processRaw(caers)
> ebScores(proc2, hyper_estimate = theta_hat, quantiles = 10)$data %>% head(3)
```

```
                    var1                  var2 N ... EBGM QUANT_10
1         1-PHENYLALANINE  HEART RATE INCREASED 1 ... 2.23     0.67
2 11 UNSPECIFIED VITAMINS                ASTHMA 1 ... 2.58     0.71
3 11 UNSPECIFIED VITAMINS CARDIAC FUNCTION TEST 1 ... 2.63     0.72
```

We can also calculate upper and lower credibility limits for the EB disproportionality score:

```
> obj <- ebScores(proc2, hyper_estimate = theta_hat, quantiles = c(10, 90))
> head(obj$data, 3)

                    var1                  var2 N ... EBGM QUANT_10 QUANT_90
1         1-PHENYLALANINE  HEART RATE INCREASED 1 ... 2.23     0.67    10.79
2 11 UNSPECIFIED VITAMINS               ASTHMA 1 ... 2.58     0.71    12.62
3 11 UNSPECIFIED VITAMINS CARDIAC FUNCTION TEST 1 ... 2.63     0.72    12.83
```

Or we can specify *EBGM* scores without limits, which may reduce computation time:

```
> ebScores(proc2, hyper_estimate = theta_hat, quantiles = NULL)$data %>% head(3)

                    var1                  var2 N ... EBGM
1         1-PHENYLALANINE  HEART RATE INCREASED 1 ... 2.23
2 11 UNSPECIFIED VITAMINS               ASTHMA 1 ... 2.58
3 11 UNSPECIFIED VITAMINS CARDIAC FUNCTION TEST 1 ... 2.63
```

Like all S3 objects in R, class "openEBGM" objects are list-like. The first element, 'data', contains the *var1-var2* pair counts, simple disproportionality scores, and EB scores. Other elements describe the hyperparameter estimation results and the quantile choices, if present.

Simple descriptive analysis

As stated previously, there are some generic functions included in **openEBGM**, two of which assist with descriptive analysis. These functions provide textual summaries of the disproportionality scores. Examples are provided below.

```
> obj <- ebScores(proc2, hyper_estimate = theta_hat, quantiles = c(10, 90))
> obj

There were 157 var1-var2 pairs with a QUANT_10 greater than 2

Top 5 Highest QUANT_10 Scores
                           var1                        var2  N ... QUANT_10
13924              REUMOFAN PLUS          WEIGHT INCREASED 16 ...    17.22
8187  HYDROXYCUT REGULAR RAPID...        EMOTIONAL DISTRESS 19 ...    12.69
13886              REUMOFAN PLUS                  IMMOBILE  6 ...    11.74
4093  EMERGEN-C (ASCORBIC ACID...                     COUGH  6 ...    10.29
7793       HYDROXYCUT HARDCORE... CARDIO-RESPIRATORY DISTRESS  8 ...    10.23
```

When the `print()` function is executed on the object, the textual output gives a brief overview of the highest EB scores for the lowest quantile calculated. In the absence of quantiles, the highest *EBGM* scores are returned with their associated *var1-var2* pairs. In addition, it states how many *var1-var2* pairs with a minimal quantile score above 2 existed in the data. Two is used as a rule of thumb in determining whether a pair is observed more than would be expected.

When `summary()` is called on an "openEBGM" object, the output includes a numerical summary on the EB scores. One may use the `log.trans=TRUE` argument to $\log_2$ transform the *EBGM* scores beforehand, in order to get information on the "Bayesian version of the information statistic" (DuMouchel, 1999, p. 180).

```
> summary(obj)

Summary of the EB-Metrics
      EBGM           QUANT_10          QUANT_90
 Min.   : 0.200   Min.   : 0.0900   Min.   : 0.43
 1st Qu.: 2.010   1st Qu.: 0.6500   1st Qu.: 9.19
 Median : 2.390   Median : 0.6900   Median :11.62
 Mean   : 2.355   Mean   : 0.7266   Mean   :10.42
 3rd Qu.: 2.580   3rd Qu.: 0.7100   3rd Qu.:12.57
 Max.   :23.260   Max.   :17.2200   Max.   :31.07
```

Graphical analysis

In addition to the above descriptive analysis, plots are exceedingly helpful when analyzing disproportionality scores. There are a number of different plot types included in the **openEBGM** package, all of which utilize the **ggplot2** package (Wickham and Chang, 2016; Wickham, 2009). The plots may

be used to diagnose the performance of the package, as well as to aid in analysis and identification of interesting *var1-var2* pairs. All of the plots are created by using the generic `plot()` function when called on an `"openEBGM"` object. The plot types include histograms, bar plots and shrinkage plots, and may be specified using the `plot.type` parameter in the generic function.

For all of the plots that follow, they may be created for the entire dataset in general, or with a specified event. When one wishes to look at the *EBGM* scores (and corresponding quantiles, etc.) for 'var1' observations corresponding to a specific 'var2' variable, the argument 'event' may be used in the `plot()` function call. An example is provided for bar plots using this feature.

Bar plots

It may be of interest to the researcher to see the comparison of *var1-var2* pairs in the disproportionality scores. For this reason, the generic `plot()` function's default plot type is a bar plot showing *var1-var2* pairs by their *EBGM* scores, with error bars when appropriate. Counts for each pair are also displayed as an additional layer of information. When quantiles were requested in the `ebScores()` function call, then the error bars on the bar plot represent the bounds of the quantiles specified. That is, the lower end of the error bar is the lowest quantile requested, and the upper end is the highest quantile requested. When no quantiles or only one quantile is requested, then error bars are not printed onto the plot. The bars are colored by the magnitude of the *EBGM* score.

Continuing from the last example, 'obj' is an object of class `"openEBGM"`, with quantile specification of the 10^{th} and 90^{th} percentiles. Figure 3 then displays the bar plot on the data in general, corresponding to the highest *EBGM* scores for the *var1-var2* pairs, and Figure 4 displays the bar plot when only considering the event terms which match the regular expression 'CHOKING'.

```
> plot(obj)  #Figure 3
```

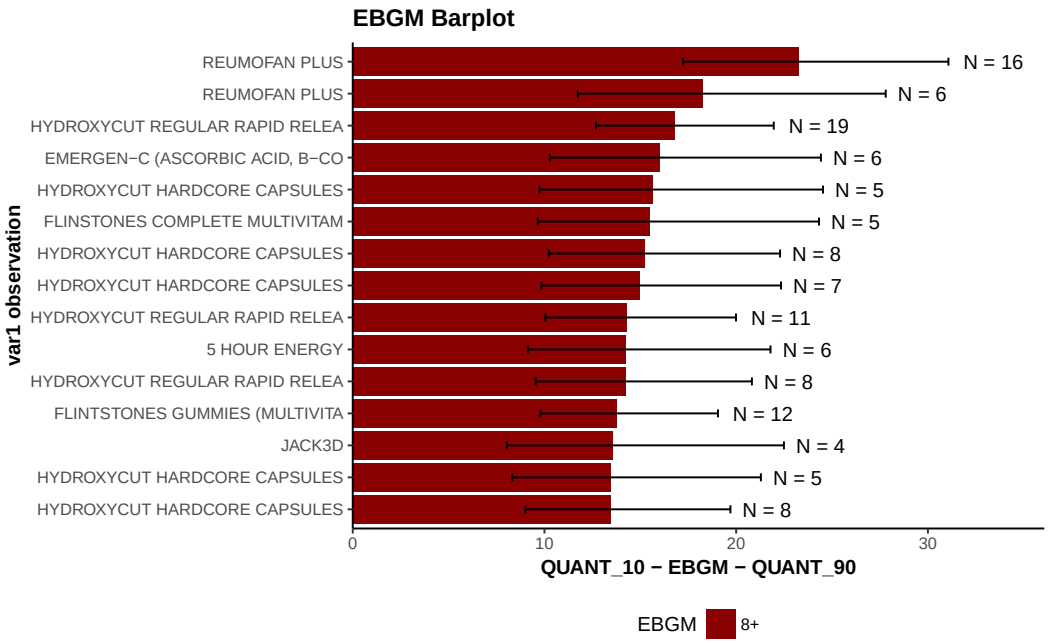


Figure 3: Top 15 product-event EBGM scores on entire dataset

```
> plot(obj, event = "CHOKING")  #Figure 4
```

Histograms

It may also be of interest to the researcher to see the distribution of *EBGM* scores. This distribution may provide the researcher with valuable insight regarding the extent of high-scoring *var1-var2* pairs, as well as their relative magnitudes. For this reason, a histogram may be created by the generic `plot()` function. The histogram output is always the *EBGM* score, regardless of whether or not quantiles were specified in the `ebScores()` function call. Figure 5 demonstrates that most of the EBGM scores are relatively small, with far fewer more interesting large scores representing unusual occurrences, illustrating why the GPS model is useful for signal detection.

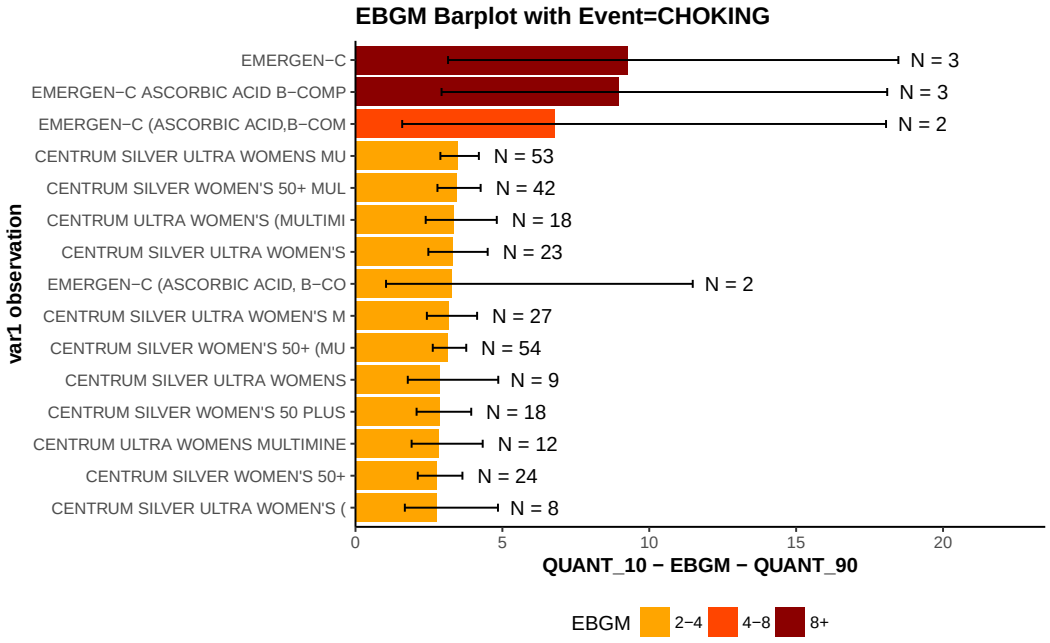


Figure 4: Top 15 product-event EBGM scores for choking events only

```
> plot(obj, plot.type = "histogram")  #Figure 5
```

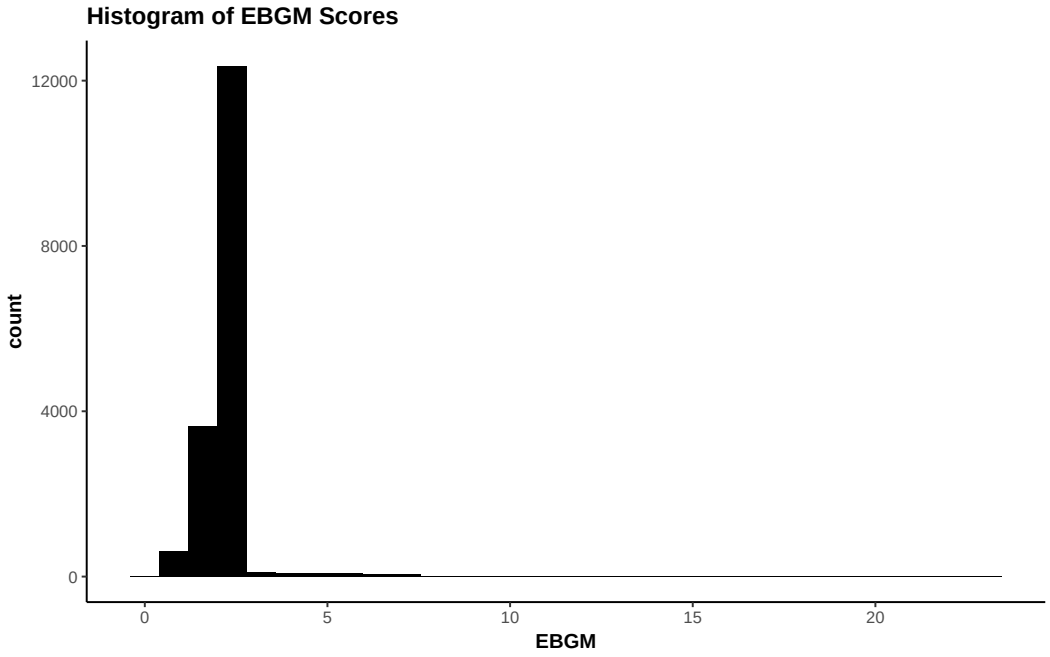


Figure 5: Histogram of EBGM scores on entire dataset

Shrinkage plots

Finally, the last plot that is included in the **openEBGM** package is a *Chirtel Squid Shrinkage Plot* (Chirtel, 2012; Duggirala et al., 2015), similar to Madigan et al. (2011, Fig.1), which shows the performance of the shrinkage algorithm on the data. In particular, it plots the *EBGM* score versus the natural log transformation of the *RR*. This plot may be useful by allowing the researcher to investigate the extent of the shrinkage that is being performed on the data, and how this shrinkage changes with varying *N* counts. The plot is colored by the count of each individually displayed *var1-var2* pair. Figure 6 illustrates that scores for product/event pairs that only occur once are greatly shrunk, but the

shrinkage lessens quickly as the number of occurrences increases. For this reason, single counts are typically not considered signals, effectively eliminating many of the false signals that occur with the simple relative reporting ratio.

```
> plot(obj, plot.type = "shrinkage")  #Figure 6
```

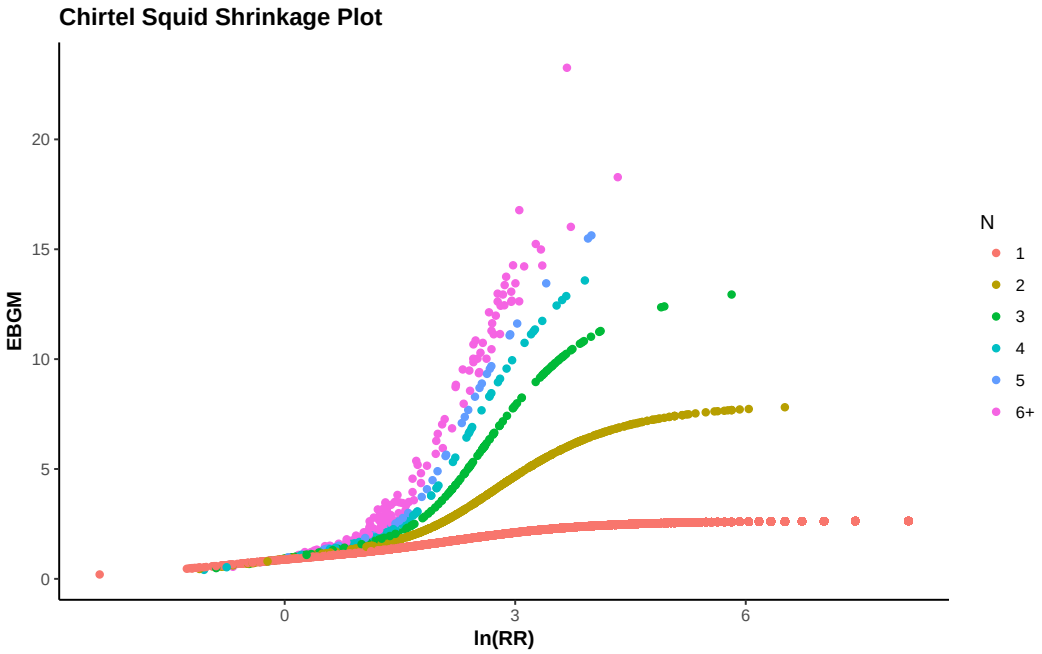


Figure 6: Shrinkage performance plot on entire dataset

Computational efficiency

As mentioned earlier, the **openEBGM** package implements the GPS model, taking into account the computational requirements that are involved when working with contingency tables, as well as when exploring a parameter space for the purposes of log-likelihood maximization (here, minimization of negative log-likelihood).

Efficient processing

As seen in the data squashing section, methodologies may be implemented that provide more efficient data processing in order to reduce overall computation time without significant loss in posterior estimates. An example is provided below showing the difference in time in hyperparameter estimation for data with and without zeroes, along with the utilization of data squashing and no data squashing.

```
> library("openEBGM")
> data("caers")
> proc_zeroes <- processRaw(caers, zeroes = TRUE)
> proc_no_zeroes <- processRaw(caers)
> squash_zeroes <- squashData(proc_zeroes, count = 0)
> squash_no_zeroes <- squashData(proc_no_zeroes)
> theta_init <- data.frame(alpha1 = c(0.2, 0.1),
+                          beta1  = c(0.1, 0.1),
+                          alpha2 = c(2,   10),
+                          beta2  = c(4,   10),
+                          p      = c(1/3, 0.2))

> system.time(hyper_zeroes <- autoHyper(squash_zeroes,
+                                       theta_init = theta_init, squashed = TRUE,
+                                       zeroes = TRUE, N_star = NULL,
+                                       max_pts = nrow(squash_zeroes)))
```

```
> system.time(hyper_no_zeroes <- autoHyper(squash_no_zeroes,
+                                          theta_init = theta_init,
+                                          squashed = TRUE, zeroes = FALSE,
+                                          N_star = 1))

> qn_zeroes <- Qn(theta_hat = hyper_zeroes$estimates,
+                 N = proc_no_zeroes$N, E = proc_no_zeroes$E)
> ebgm_zeroes <- ebgm(theta_hat = hyper_zeroes$estimates,
+                     N = proc_no_zeroes$N, E = proc_no_zeroes$E, qn = qn_zeroes)

> qn_no_zeroes <- Qn(theta_hat = hyper_no_zeroes$estimates,
+                    N = proc_no_zeroes$N, E = proc_no_zeroes$E)
> ebgm_no_zeroes <- ebgm(theta_hat = hyper_no_zeroes$estimates,
+                        N = proc_no_zeroes$N, E = proc_no_zeroes$E, qn = qn_no_zeroes)

> hyper_zeroes$estimates
> hyper_no_zeroes$estimates
```

We may look at the output of the above code to see how long it took to estimate the hyperparameters with and without zeroes, as well as their associated estimates. In the output below, for both the time elapsed in calculation as well as the estimates, the data including zeroes is displayed first.

```
   user  system elapsed
 311.73    10.09  321.83

   user  system elapsed
   2.58    0.00    2.58

   alpha1     beta1    alpha2     beta2         P
0.1874269 0.2171502 2.3409023 1.6725397 0.4582590

    alpha1      beta1     alpha2      beta2          P
3.25091549 0.39971566 2.02580114 1.90804807 0.06539048
```

A Bland-Altman plot comparing the *EBGM* scores when using zero counts vs. strictly nonzero counts for hyperparameter estimates can be seen in Figure 7. We can also compare the difference between squashed data and non-squashed data on the estimates of the hyperparameters, as well as how long it takes to estimate them.

```
> system.time(hyper_squash <- autoHyper(data = squash_no_zeroes,
+                                       theta_init = theta_init, squashed = TRUE,
+                                       zeroes = FALSE, N_star = 1))
> system.time(hyper_no_squash <- autoHyper(data = proc_no_zeroes,
+                                          theta_init = theta_init, squashed = FALSE,
+                                          zeroes = FALSE, N_star = 1))
> hyper_squash$estimates
> hyper_no_squash$estimates

   user  system elapsed
   2.54    0.00    2.54

   user  system elapsed
  24.62    0.17   24.79

    alpha1      beta1     alpha2      beta2          P
3.25091549 0.39971566 2.02580114 1.90804807 0.06539048

    alpha1      beta1     alpha2      beta2          P
3.25599765 0.39999135 2.02374652 1.90612584 0.06530695
```

As we see in the example above, the difference in *EBGM* estimates when comparing the use of squashing and not squashing for hyperparameter estimation is minimal, and by analysis of the Bland-Altman plot of the *EBGM* scores (see Figure 8), the results are nearly identical.

We see that not including zero counts makes the estimation of the hyperparameters far more computationally feasible than when one uses zeroes, which would occur if we used contingency table

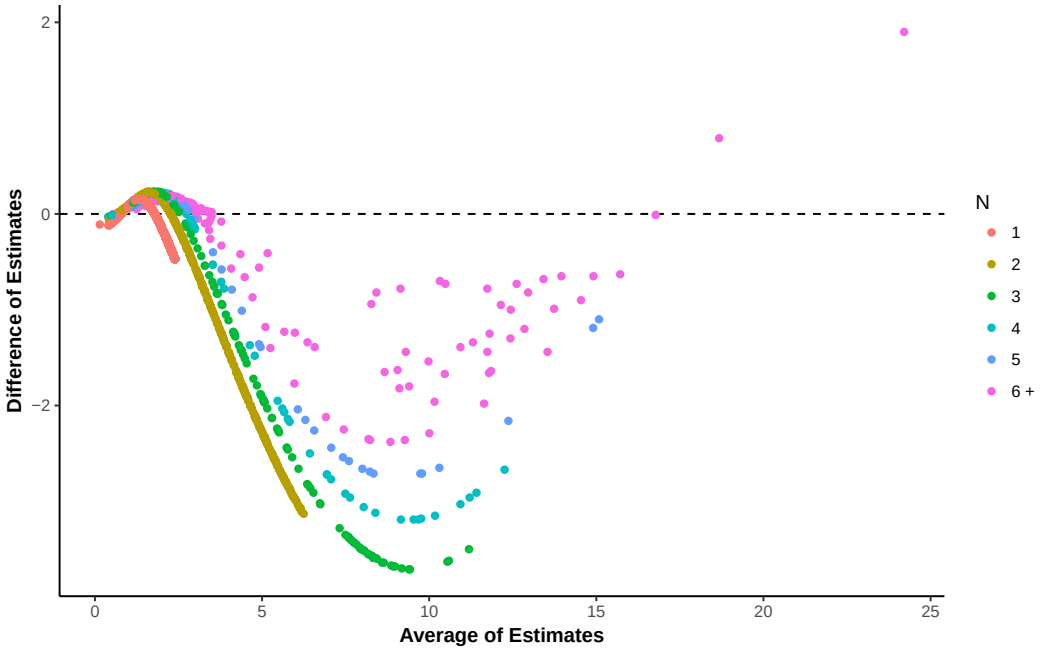


Figure 7: Bland-Altman plot of *EBGM* scores with and without zeroes (zeroes - no zeroes)

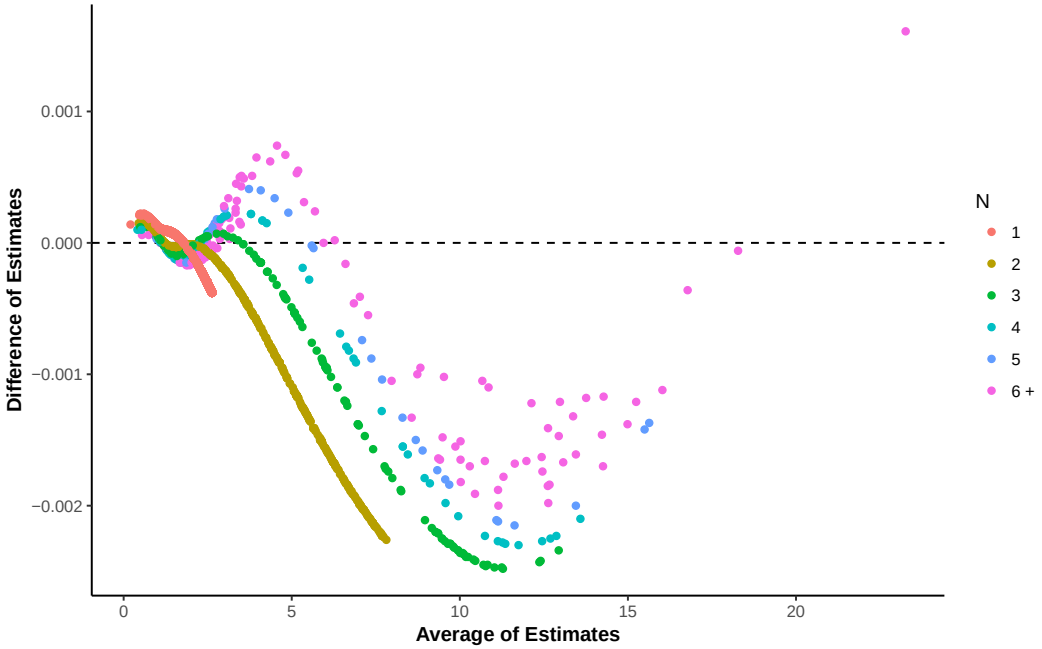


Figure 8: Bland-Altman plot of *EBGM* scores with and without data squashing (data squashing - no data squashing)

data structures instead of tidy data where zeroes are generally not needed. Appendix H.11 further illustrates efficiency gains from removing zeroes.

Additionally, **openEBGM** utilizes other strategies to reduce the amount of unnecessary computation required to employ the GPS model. In particular, the package uses certain data structures which represent the sometimes large contingency table of *var1-var2* pairs as a `"data.table"` (Dowle and Srinivasan, 2017) in tidy form (Wickham, 2014). This results in more efficient data squashing and computation of marginal totals and expected counts. Representing data in tidy form instead of a matrix resembling a contingency table avoids inefficient row-wise operations involving many unnecessary zeroes. Furthermore, the documentation for **data.table** claims fast merging of `"data.table"` objects. While developing the code, we noticed a sizable speed improvement for the merging and aggregation methods for `"data.table"` objects over the standard methods for traditional data frames. Appendix

H.12 demonstrates that **openEBGM** can process very large data sets in a reasonable amount of time, even when including zeroes for hyperparameter estimation.

Conclusion and discussion

Our package greatly simplifies the process of generating disproportionality scores for large contingency tables. In addition, the analysis of these scores is simplified using built-in functions which summarize and display the results. Furthermore, **openEBGM**'s implementation of the GPS model aims for greater efficiency (see Appendix H.10), flexibility and usability compared to other open-source implementations of the same model.

Further applications

Adverse event reporting is not the only application of the GPS methodology. DuMouchel (1999) provides other examples, including natural language processing. GPS can find word pairs that appear more frequently than expected, which could prove to be useful for crawling the web in search of trends (e.g. document searches). DuMouchel also suggests using the model to find supermarket products that are often purchased together (stratified by store location).

Multi-item gamma-Poisson shrinker

openEBGM currently implements the GPS model, which can only examine single *var1-var2* pairs. An extension of this model exists, allowing the study of interactions. The FDA currently uses (Duggirala et al., 2016; Szarfman et al., 2002) the *Multi-item Gamma-Poisson Shrinker* (MGPS) model (DuMouchel and Pregibon, 2001) to find higher-than-expected reporting of adverse events associated with specific products or product groups. MGPS can model Drug-Drug-Event or Drug-Event-Event triples (with higher order interactions also being possible). DuMouchel and Pregibon (2001) use a log-linear model in the MGPS approach to account for lower-order associations. More discussion on the details of MGPS can be found in Szarfman et al. (2002). Existing proprietery implementations of MGPS include Oracle Health Science's Empirica Signal (Oracle, 2017) and some versions of JMP® Clinical software (SAS Institute Inc., 2017). Future improvements to **openEBGM** could include the addition of the MGPS model.

Acknowledgements

We thank Stuart Chirtel for introducing us to the GPS model, helping us understand its use, and encouraging us to implement it in R. We thank Hugh Rand for valuable insight into numerical techniques and for encouraging us to add our code to the CRAN repository and write this article. We also thank Hugh, Stuart, John Bowers, and the journal reviewers for valuable comments and suggestions. We thank James Pettengill for helping us test the code. We thank Lyle Canida and Ella Smith for helping us obtain and understand the CAERS data; thanks also to everyone else involved in creating, maintaining, and disseminating the CAERS data. Finally, we thank the authors of the **PhViD** package, whose code gave us a starting point to develop our own code.

Bibliography

I. Ahmed and A. Poncet. *PhViD: PharmacoVigilance Signal Detection*, 2016. URL https://CRAN.R-project.org/package=PhViD. R package version 1.0.8. [p501]

I. Ahmed, F. Haramburu, A. Fourrier-Réglat, F. Thiessard, C. Kreft-Jais, G. Miremont-Salamé, B. Bégaud, and P. Tubert-Bitter. Bayesian pharmacovigilance signal detection methods revisited in a multiple comparison setting. *Statistics in Medicine*, 28(13):1774–1792, 2009. ISSN 1097-0258. URL https://doi.org/10.1002/sim.3586. [p501]

J. S. Almenoff, K. K. LaCroix, N. A. Yuen, D. Fram, and W. DuMouchel. Comparative performance of two quantitative safety signalling methods. *Drug Safety*, 29(10):875–887, October 2006. ISSN 1179-1942. URL https://doi.org/10.2165/00002018-200629100-00005. [p499]

S. Chirtel. Disproportionality analyses for detection of food adverse events. ENAR 2012 Spring Meeting, Washington, D.C., USA, 2012. URL https://www.enar.org/meetings/meetings2012/Meeting_ABSTRACTS.pdf. [p510]

M. Dowle and A. Srinivasan. *data.table: Extension of 'data.frame'*, 2017. URL https://CRAN.R-project.org/package=data.table. R package version 1.10.4. [p513]

H. Duggirala, J. Tonning, E. Smith, R. Bright, J. Baker, R. Ball, C. Bell, K. Bouri, S. J. Bright-Ponte, T. Botsis, M. Boyer, K. Burkhart, G. Condrey, J. Chen, S. Chirtel, R. Filice, H. Francis, H. Jiang, J. Levine, D. Martin, T. Oladipo, R. O'Neill, L. Palmer, A. Paredes, G. Rochester, D. Sholtes, H. Wong, Z. Xu, A. Szarfman, and T. Kass-Hout. Data mining at FDA. Technical report, U.S. Food and Drug Administration, November 2015. URL https://www.fda.gov/scienceresearch/dataminingatfda/ucm446239. [p499, 510]

H. Duggirala, J. Tonning, E. Smith, R. Bright, J. Baker, R. Ball, C. Bell, S. J. Bright-Ponte, T. Botsis, K. Bouri, M. Boyer, K. Burkhart, G. Condrey, J. Chen, S. Chirtel, R. Filice, H. Francis, H. Jiang, J. Levine, D. Martin, T. Oladipo, R. O'Neill, L. Palmer, A. Paredes, G. Rochester, D. Sholtes, A. Szarfman, H. Wong, Z. Xu, and T. Kass-Hout. Use of data mining at the Food and Drug Administration. *Journal of the American Medical Informatics Association*, 23(2):428–434, 2016. URL https://doi.org/10.1093/jamia/ocv063. [p514]

W. DuMouchel. Bayesian data mining in large frequency tables, with an application to the FDA spontaneous reporting system. *The American Statistician*, 53(3):177–190, August 1999. URL https://doi.org/10.1080/00031305.1999.10474456. [p499, 500, 502, 504, 505, 506, 507, 508, 514]

W. DuMouchel and D. Pregibon. Empirical bayes screening for multi-item associations. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '01, pages 67–76, New York, NY, USA, 2001. ACM. ISBN 1-58113-391-X. URL https://doi.org/10.1145/502512.502526. [p499, 504, 505, 506, 514]

S. J. W. Evans, P. Waller, and S. Davis. Use of proportional reporting ratios (PRRs) for signal generation from spontaneous adverse drug reaction reports. *Pharmacoepidemiology and Drug Safety*, 10(6):483–486, 2001. ISSN 1099-1557. URL https://doi.org/10.1002/pds.677. [p499]

J. Ihrie and T. Canida. *openEBGM: EBGM Scores for Mining Large Contingency Tables*, 2017. URL https://CRAN.R-project.org/package=openEBGM. R package version 0.3.0. [p499]

D. Madigan, P. Ryan, S. Simpson, and I. Zorych. Bayesian methods in pharmacovigilance. In J. Bernardo, M. Bayarri, J. Berger, A. Dawid, D. Heckerman, A. Smith, and M. West, editors, *Bayesian Statistics 9*. Oxford University Press, 2011. ISBN 9780199694587. URL https://doi.org/10.1093/acprof:oso/9780199694587.003.0014. [p499, 510]

Oracle. Oracle Health Sciences Empirica Signal, 2017. URL https://www.oracle.com/us/products/applications/health-sciences/safety/empirica-signal/. [p514]

SAS Institute Inc. JMP Clinical: Clinical data analysis software for ensuring trial safety and efficacy, 2017. URL https://www.jmp.com/en_us/software/clinical-data-analysis-software.html. [p514]

A. Szarfman, S. G. Machado, and R. T. O'Neill. Use of screening algorithms and computer systems to efficiently signal higher-than-expected combinations of drugs and events in the US FDA's spontaneous reports database. *Drug Safety*, 25(6):381–392, 2002. ISSN 1179-1942. URL https://doi.org/10.2165/00002018-200225060-00001. [p500, 501, 514]

U.S. Food and Drug Administration. CFSAN Adverse Event Reporting System (CAERS), 2017. URL https://www.fda.gov/Food/ComplianceEnforcement/ucm494015.htm. [p501, 502]

S. Venturini and J. Myers. *mederrRank: Bayesian Methods for Identifying the Most Harmful Medication Errors*, 2015. URL https://CRAN.R-project.org/package=mederrRank. R package version 0.0.8. [p501]

S. Venturini, J. M. Franklin, L. Morlock, and F. Dominici. Random effects models for identifying the most harmful medication errors in a large, voluntary reporting database. *The Annals of Applied Statistics*, 11(2):504–526, 2017. URL https://doi.org/10.1214/16-AOAS974. [p501]

H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2009. ISBN 978-0-387-98140-6. URL http://ggplot2.org. [p508]

H. Wickham. Tidy data. *Journal of Statistical Software, Articles*, 59(10):1–23, 2014. ISSN 1548-7660. URL https://doi.org/10.18637/jss.v059.i10. [p501, 513]

H. Wickham. *tidyr: Easily Tidy Data with 'spread()' and 'gather()' Functions*, 2017. URL https://CRAN.R-project.org/package=tidyr. R package version 0.6.3. [p502]

H. Wickham and W. Chang. *ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics*, 2016. URL https://CRAN.R-project.org/package=ggplot2. R package version 2.2.1. [p508]

Travis Canida
U.S. Food and Drug Administration
Center for Food Safety and Applied Nutrition
Biostatistics and Bioinformatics Staff
5001 Campus Drive
College Park, MD 20740
United States
Travis.Canida@fda.hhs.gov

John Ihrie
U.S. Food and Drug Administration
Center for Food Safety and Applied Nutrition
Biostatistics and Bioinformatics Staff
5001 Campus Drive
College Park, MD 20740
United States
John.Ihrie@fda.hhs.gov

Appendix A

```
# Purpose: To compare computing times for openEBGM vs. PhViD.
# Notes:   We continue from the data set created in the CAERS data set example.
#          PhViD also computes many multiplicity-adjusted values not shown here.
#          The pre-2017 CAERS data set might change slightly over time.

> library("PhViD")
> dat_tidy$id <- 1:nrow(dat_tidy)  #since PhViD cannot count unique reports
> counts <- processRaw(dat_tidy)
> nrow(counts)  #number of points/var1-var2 pairs

[1] 145257

> theta_init <- c(alpha1 = 0.2, beta1 = 0.06, alpha2 = 1.4, beta2 = 1.8, P = 0.1)

#Start with the PhViD package
> system.time({
+   dat_phvid <- as.PhViD(counts[, 1:3])
+   results_phvid <- GPS(dat_phvid, RANKSTAT = 3, TRONC = TRUE, PRIOR.INIT = theta_init)
+ })

   user  system elapsed
 342.08    4.32  346.46

> results_phvid <- results_phvid$ALLSIGNALS[, 1:6]
> results_phvid <- results_phvid[order(results_phvid$drug, results_phvid$event), ]
> results_phvid$EBGM <- round(2 ^ results_phvid[, 'post E(Lambda)'], 3)
> row.names(results_phvid) <- NULL
> head(results_phvid, 3)

                            drug             event count ...    n11/E  EBGM
1  CENTRUM SILVER WOMEN'S 50+...           CHOKING     2 ... 14.27370 1.639
2  CENTRUM SILVER WOMEN'S 50+...         DYSPHAGIA     1 ... 13.53379 1.153
3  CENTRUM SILVER WOMEN'S 50+... THROAT IRRITATION     1 ... 48.00568 1.187

#Now for the openEBGM package
> theta_init_df <- t(data.frame(theta_init))
> system.time({
+   theta1 <- exploreHypers(counts, theta_init = theta_init_df, squashed = FALSE,
+                           method = "nlm", max_pts = nrow(counts))$estimates
```

```
+   theta1 <- as.numeric(theta1[1, 2:6])
+   results_open1 <- ebScores(counts, list(estimates = theta1), quantiles = NULL)
+ })

   user  system elapsed
  84.18    0.02   84.19

> dat_open1 <- results_open1$data
> head(dat_open1, 3)

                            var1              var2 N ...    RR ... EBGM
1  CENTRUM SILVER WOMEN'S 50+...           CHOKING 2 ... 14.27 ... 1.64
2  CENTRUM SILVER WOMEN'S 50+...         DYSPHAGIA 1 ... 13.53 ... 1.15
3  CENTRUM SILVER WOMEN'S 50+... THROAT IRRITATION 1 ... 48.01 ... 1.19

#Now with data squashing
> system.time({
+   squashed <- squashData(counts, bin_size = 100)
+   squashed <- squashData(squashed, count = 2, bin_size = 10)
+   theta2 <- exploreHypers(squashed, theta_init = theta_init_df,
+                           method = "nlm")$estimates
+   theta2 <- as.numeric(theta2[1, 2:6])
+   results_open2 <- ebScores(counts, list(estimates = theta2), quantiles = NULL)
+ })

   user  system elapsed
   5.64    0.03    5.68

> dat_open2 <- results_open2$data
> head(dat_open2, 3)

                            var1              var2 N ...    RR ... EBGM
1  CENTRUM SILVER WOMEN'S 50+...           CHOKING 2 ... 14.27 ... 1.64
2  CENTRUM SILVER WOMEN'S 50+...         DYSPHAGIA 1 ... 13.53 ... 1.15
3  CENTRUM SILVER WOMEN'S 50+... THROAT IRRITATION 1 ... 48.01 ... 1.19
```

Appendix B

Rows	Without zeroes			With zeroes		
	Unstratified	Stratified	Points	Unstratified	Stratified	Points
50K	0.6	0.8	40K	32	47	18M
100K	1.3	1.9	80K	84	122	44M
150K	2.2	2.9	118K	157	226	72M
200K	3.1	3.8	156K	205	308	103M
250K	3.0	3.9	194K	273	430	139M
300K	3.6	4.6	227K	352	519	171M

Elapsed time (in seconds) and approximate number of points (N, E) for `processRaw()` on various sizes of raw data. Using gender only when stratifying. K = 1,000; M = 1,000,000

Appendix C

```
# Purpose: To assess how quickly openEBGM can process a very large data set.
# Notes: We include all industry codes. We also include zero counts in the
#        hyperparameter estimation step to further illustrate how quickly openEBGM
#        can process a very large number of points (~177 million).
```

```
> site <- "https://www.fda.gov/downloads/Food/ComplianceEnforcement/UCM494018.csv"
> dat  <- read.csv(site, stringsAsFactors = FALSE, strip.white = TRUE)
> dat$yr <- dat$RA_CAERS.Created.Date
> dat$yr <- substr(dat$yr, start = nchar(dat$yr) - 3, stop = nchar(dat$yr))
> dat$yr <- as.integer(dat$yr)
> dat <- dat[dat$yr < 2017, ]  #using all industry codes
> dat$var1 <- dat$PRI_Reported.Brand.Product.Name
> dat$var2 <- dat$SYM_One.Row.Coded.Symptoms
> dat$id   <- dat$RA_Report..
> dat$strat_gen <- dat$CI_Gender
> dat$strat_gen <- ifelse(dat$strat_gen %in% c("Female", "Male"),
+                         dat$strat_gen, "unknown")
> vars <- c("id", "var1", "var2", "strat_gen")
> dat <- dat[, vars]
> dat <- dat[!dat$var1 %in% tools::showNonASCII(dat$var1), ]
> dat_tidy <- tidyr::separate_rows(dat, var2, sep = ", ")
> dat_tidy <- dat_tidy[dat_tidy$var2 != "", ]
> nrow(dat_tidy)  #rows in raw data

 [1] 307719

> system.time(counts <- processRaw(dat_tidy, zeroes = TRUE))

 user system elapsed
 289.07 63.22 352.33

> nrow(counts)  #number of points/var1-var2 pairs in processed data

 [1] 176739728

> system.time({
+   squashed <- squashData(counts, count = 0, bin_size = 50000, keep_bins = 0)
+ })

 user system elapsed
 75.38 12.40 87.80

> nrow(squashed)

 [1] 235734

> system.time({
+   squashed <- squashData(squashed, bin_size = 500, keep_bins = 1)
+   squashed <- squashData(squashed, count = 2, bin_size = 100, keep_bins = 1)
+   squashed <- squashData(squashed, count = 3, bin_size = 50, keep_bins = 1)
+   squashed <- squashData(squashed, count = 4, bin_size = 25, keep_bins = 1)
+   squashed <- squashData(squashed, count = 5, bin_size = 10, keep_bins = 1)
+   squashed <- squashData(squashed, count = 6, bin_size = 10, keep_bins = 1)
+ })

 user system elapsed
 0.16 0.00 0.16

> nrow(squashed)  #number of points used for hyperparameter estimation

 [1] 7039

> theta_init <- c(alpha1 = 0.2, beta1 = 0.06, alpha2 = 1.4, beta2 = 1.8, P = 0.1)
> theta_init_df <- t(data.frame(theta_init))

> system.time({
+   theta_est <- exploreHypers(squashed, theta_init = theta_init_df,
+                              squashed = TRUE, zeroes = TRUE, N_star = NULL,
```

```
+                              method = "nlminb", std_errors = TRUE)
+   theta_hat <- theta_est$estimates
+   theta_hat <- as.numeric(theta_hat[1, 2:6])
+ })

   user  system elapsed
   4.63    0.09    4.72

> theta_hat

[1] 0.036657366 0.006645404 0.681479849 0.605705800 0.020295102

> theta_est$std_errs  #standard errors of hyperparameter estimates

  guess_num       a1_se       b1_se      a2_se       b2_se       p_se
1         1 0.007072148 0.000275861 0.00964448 0.008095735 0.00389658

> system.time({
+   counts_sans0 <- counts[counts$N != 0, ]  #do not need EB scores for zero counts
+   results <- ebScores(counts_sans0, list(estimates = theta_hat), quantiles = NULL)
+ })

 user system elapsed
 2.95 0.02 2.97

> nrow(results$data)  #number of nonzero counts

 [1] 232203
```

rentrez: An R package for the NCBI eUtils API

by David J. Winter

Abstract The USA National Center for Biotechnology Information (NCBI) is one of the world's most important sources of biological information. NCBI databases like PubMed and GenBank contain millions of records describing bibliographic, genetic, genomic, and medical data. Here I present **rentrez**, a package which provides an R interface to 50 NCBI databases. The package is well-documented, contains an extensive suite of unit tests and has an active user base. The programmatic interface to the NCBI provided by **rentrez** allows researchers to query databases and download or import particular records into R sessions for subsequent analysis. The complete nature of the package, its extensive test-suite and the fact the package implements the NCBI's usage policies all make **rentrez** a powerful aid to developers of new packages that perform more specific tasks.

Introduction

The USA National Center for Biotechnology Information (NCBI) is one of the world's largest and most important sources of biological data. At the time of writing, the NCBI PubMed database provided information on 27.5 million journal articles, including 4.6 million full text records. The NCBI Nucleotide Database (including GenBank) had data for 243.3 million different sequences and dbSNP described 997.3 million different genetic variants. Records from all of these databases can be cross-referenced with the 1.3 million species in the NCBI taxonomy, and PubMed entries can be searched using a controlled vocabulary containing 272 thousand unique terms.

The NCBI provides access to a total of 50 databases through a web interface, public FTP sites and an API called Entrez Programming Utilities (eUtils, Sayers and Wheeler (2004)). R packages from the Bioconductor project (e.g., **genomes**, Stubben (2015); **RMassBank**, Stravs et al. (2013) and **MeSHSim**, Zhou and Shui (2015)) or available from CRAN (e.g., **ape**, Paradis et al. (2004); **RISmed**, Kovalchik (2017) and **pubmed.mineR**, Rani et al. (2014)) take advantage of the Eutils API to perform specific tasks. Two packages, **rentrez** and **reutils** (Schöfl, 2016), provide functions that cover the entire API. Here I describe **rentrez**, a package which provides users with a simple and consistent interface to eUtils. This paper discusses the design of the package, illustrates its use in biological research and demonstrates how the provided functions can aid the development of other packages designed to meet more specific goals.

The eUtils API and rentrez

The eUtils API provides endpoints for searching each of the databases it covers, finding cross-references among records in those databases and fetching particular records (in complete or summary form). The design of **rentrez** mirrors that of eUtils, with each of these endpoints represented by a core function that has arguments named to match those used in the API documentation (Table 1). The most important arguments to each R function are documented, and the help pages associated with these functions contain a reference to the relevant section of the eUtils documentation.

Table 1: Core eUtils endpoints and their **rentrez** counterparts

NCBI endpoint	Purpose	Core function
esearch	Locate records matching search criteria.	`entrez_search`
elink	Discover cross-linked records.	`entrez_link`
esummary	Fetch summary data on a set of records.	`entrez_summary`
efetch	Fetch complete records in a variety of formats.	`entrez_fetch`

Typically, a user will begin by using `entrez_search` to discover unique identifiers for database records matching particular criteria. This function requires a database (argument `db`) and a search term (argument `term`). The `term` argument can take advantage of the eUtils search syntax, in which queries can be associated with particular 'search fields' (enclosed in square brackets), to limit the number of matching records. The fields available for a given database can be retrieved with the function `entrez_db_searchable`. The following call demonstrates the use of this search syntax. This query finds scientific papers that contain the phrase 'R Package' in their title while limiting the result to include only papers published in 2017.

```
pubmed_search <- entrez_search(db="pubmed",
                               term="(R package[TITL]) AND 2017[PDAT]",
                               use_history=TRUE)
pubmed_search
```

```
#> Entrez search result with 62 hits (object contains 20 IDs and a web_history object)
#>  Search term (as translated):  R package[TITL] AND 2017[PDAT]
```

For the most part, the objects returned by **rentrez** functions are S3 objects that inherit from `"list"` objects. Each of these classes has its own `print` method, providing users with a succint summary of the object's contents. In this case, `entrez_search` returns a an `"esearch"`, object and the print method describes the number of records matching the query. These objects also contain either a character vector containing the unique identifiers of records that match the query or a `"web_history"` object that serves as a reference to this set of identifiers stored on the NCBI's servers. These `"web_history"` objects are useful when dealing with large numbers of records, as they reduce the number of identifiers that need to be exchanged between a user and NCBI servers. Both identifiers and `"web_history"` objects can be passed to the other core functions to retrieve information about the records they represent. For example, a call to `entrez_summary` returns information about each paper identified in the search above.

```
pkg_paper_summs <- entrez_summary(db="pubmed", web_history=pubmed_search$web_history)
pkg_paper_summs
```

```
#> List of  62 esummary records. First record:
#>
#>  $`28759592`
#> esummary result with 42 items:
#>  [1] uid               pubdate           epubdate
#>  [4] source            authors           lastauthor
#>  [7] title             sorttitle         volume
#> [10] issue             pages             lang
#> [13] nlmuniqueid       issn              essn
#> [16] pubtype           recordstatus      pubstatus
#> [19] articleids        history           references
#> [22] attributes        pmcrefcount       fulljournalname
#> [25] elocationid       doctype           srccontriblist
#> [28] booktitle         medium            edition
#> [31] publisherlocation publishername     srcdate
#> [34] reportnumber      availablefromurl  locationlabel
#> [37] doccontriblist    docdate           bookname
#> [40] chapter           sortpubdate       sortfirstauthor
```

In addition to matching each of the eUtils endpoints, **rentrez** provides utility functions that facilitate common workflows. For example, the function `extract_from_esummary` allows users to extract some subset of the items contained in each of a set of summary records. In this case, the names of the journals that these papers appeared in can be retrieved. The resulting character vector can then be used to identify the PubMed-indexed journals that have published the most papers describing R packages this year.

```
journals <- extract_from_esummary(pkg_paper_summs, "fulljournalname")
journals_by_R_pkgs <- sort(table(journals), decreasing = TRUE)
head(journals_by_R_pkgs,3)
```

```
#> journals
#> Bioinformatics (Oxford, England)              BMC bioinformatics
#>                               16                               9
#>      Molecular ecology resources
#>                                9
```

Demonstration: retrieving unique transcripts for a given gene

Records in the NCBI's various databases are heavily cross-referenced, allowing users to identify and download data related to particular papers, organisms or genes. By providing a programmatic interface to these records **rentrez** allows R users to develop reproducible workflows that either

download particular datasets for further analysis or load them into an R session. Here I demonstrate such a workflow, downloading DNA sequences corresponding to unique mRNA transcripts of a particular gene in a particular species.

Our aim is to retrieve the sequence of mRNA transcripts associated with the gene that encodes Amyloid Beta Precursor Protein in humans. This gene is identified by the gene symbol [1] 'APP'. The NCBI database dealing with genetic loci (rather than particular sequences) is called 'Gene', so the first step to recovering the sequence data is discovering the unique identifier associated with APP in this database. This can be achieved with `entrez_search`, using the gene symbol and species in the search term.

```
app_gene <- entrez_search(db="gene", term="(Homo sapiens[ORGN]) AND APP[GENE]")
app_gene

#> Entrez search result with 1 hits (object contains 1 IDs and no web_history object)
#>  Search term (as translated):  "Homo sapiens"[Organism] AND APP[GENE]
```

We now have a unique identifier for APP in the Gene database. In order to download sequences for this gene we need to find records from the NCBI Nucleotide database that are associated with the Gene record. The function `entrez_link` can be used to find cross-referenced records. In this case, a single call to `entrez_link` can identify human APP sequences in the nucleotide database in general and in a number of restrictive subsets of that database.

```
nuc_links <- entrez_link(dbfrom="gene", id=app_gene$ids, db="nuccore")
nuc_links$links

#> elink result with information from 5 databases:
#> [1] gene_nuccore            gene_nuccore_mgc        gene_nuccore_pos
#> [4] gene_nuccore_refseqgene gene_nuccore_refseqrna
```

The RefSeq RNA subset on the Nucleotide database contains a curated set of mRNA transcripts for different genes. Thus the unique identifiers contained in the `gene_nuccore_refseqrna` element correspond to the sequences we wish to download.The function `entrez_fetch` allows users to retrieve complete records in a variety of formats. Here the sequences are retrieved in the standard 'fasta' format, and returned as a character vector with a single element.

```
raw_recs <- entrez_fetch(db="nuccore",
                         id=nuc_links$links$gene_nuccore_refseqrna,
                         rettype="fasta")
cat(substr(raw_recs, 1,303), "...")

#> >NM_001136131.2 Homo sapiens amyloid beta precursor protein (APP) ...
#> GTCGGATGATTCAAGCTCACGGGGACGAGCAGGAGCGCTCTCGACTTTTCTAGAGCCTCAGCGTCCTAGG
#> ACTCACCTTTCCCTGATCCTGCACCGTCCCTCTCCTGGCCCCAGACTCTCCCTCCCACTGTTCACGAAGC
#> CCAGGTACCCACTGATGGTAATGCTGGCCTGCTGGCTGAACCCCAGATTGCCATGTTCTGTGGCAGA...
```

Sequences retrieved in this way could be written to file to be used by other software.

```
cat(raw_recs, file="APP_transcripts.fasta")
```

Alternatively, the sequences can be analysed within R using packages designed for sequence data. In this case, the data can be represented as a `'DNAbin'` object using the phylogenetics package **ape**.

```
tf <- tempfile()
cat(raw_recs, file=tf)
ape::read.dna(tf, format="fasta")

#> 10 DNA sequences in binary format stored in a list.
#>
#> Mean sequence length: 3477.9
#>    Shortest sequence: 3255
#>     Longest sequence: 3648
#>
#> Labels:
#> NM_001136131.2 Homo sapiens amyloid beta precursor protein (...
```

[1] http://www.genenames.org/

```
#> NM_001136016.3 Homo sapiens amyloid beta precursor protein (...
#> NM_001204303.1 Homo sapiens amyloid beta precursor protein (...
#> NM_001204301.1 Homo sapiens amyloid beta precursor protein (...
#> NM_001204302.1 Homo sapiens amyloid beta precursor protein (...
#> NM_201414.2 Homo sapiens amyloid beta precursor protein (APP...
#> ...
#>
#> Base composition:
#>     a     c     g     t
#> 0.276 0.223 0.258 0.244
```

The workflow detailed above provides a relatively simple example of how functions provided by **rentrez** can be used to identify, retrieve and analyse data from the NCBI's databases. The package includes an extensive vignette which documents each of the eUtils endpoints and demonstrates a number of detailed workflows. This document also describes how analyses can be scaled to analyse much larger datasets than are described in this paper. The tutorial can be accessed from within an R session by typing `vignette(topic="rentrez_tutorial")`.

Demonstration: development of a new package

Development of **rentrez** has deliberately focused on producing a "low-level" package that provides a flexible interface to the entire eUtils API. As a result the package does not provide functions for any particular analysis or return records in any of the object classes made available for biological data by other packages. Rather, it is hoped that by providing a reliable interface to the eUtils API that meets the NCBI's terms of use **rentrez** will help other developers to build packages for more specific use-cases. Indeed, the package has already been used to integrate NCBI data into packages dealing with sequence analysis (**genbankr**, Becker and Lawrence (2017)), retrieval of phylogenetic trees (**rotl**, Michonneau et al. (2016)) and handling of full-text journal articles (**fulltext**, Chamberlain (2016)).

New packages that take advantage of **rentrez** will usually focus on providing a simple interface to users, so researchers will not need to be familiar with the syntax or arguments used in eUtils to perform tasks. Packages may also provide functions to parse files returned by `entrez_fetch`, either to extract particular information from those files or represent them as R objects. Here I present an example of a small package that performs both of these tasks.The software repository for this manuscript (`https://github.com/dwinter/rentrez_ms`) includes the code for a package called 'tidytaxonomy' that can be used to explore the taxonomic diversity of various NCBI databases. This demonstrates how the low-level code in **rentrez** can be used to develop specific applications that have a simpler interface than could be achieved with core **rentrez** functions alone.

The exposed functions from `tidytaxonomy` retrieve data from NCBI, but do not require the user to have any knowledge of the eUtils API. Internal functions

parse the XML formatted records returned from the NCBI Taxonomy database and extract relevant information. The core function `tidy_taxonomy` allows users to retrieve a part of the NCBI Taxonomy database in 'tidy data' format (Wickham, 2014).

```
# install.packages('devtools') # (If not already installed)
devtools::load_all("tidytaxon")

animal_orders <- tidy_taxonomy("animals",
                               lowest_rank="order",
                               higher_ranks=c("phylum", "class"))
head(animal_orders,3)
```

```
#>     phylum       class            order
#> 1 Chordata Actinopteri    Lutjaniformes
#> 2 Chordata Actinopteri     Gerreiformes
#> 3 Chordata Actinopteri Priacanthiformes
```

Once this data is obtained, two additional functions make it easy to include the number of records a given taxon has in a particular database. The function `taxon_children` is specifically for counting taxonomy records which are subordinate to a given taxonomic group. The other function, `taxon_records`, discovers records in any NCBI database.

```
animal_orders$species <- taxon_children(animal_orders$order)
animal_orders$genomes <- taxon_records(animal_orders$order, db="genome")
```

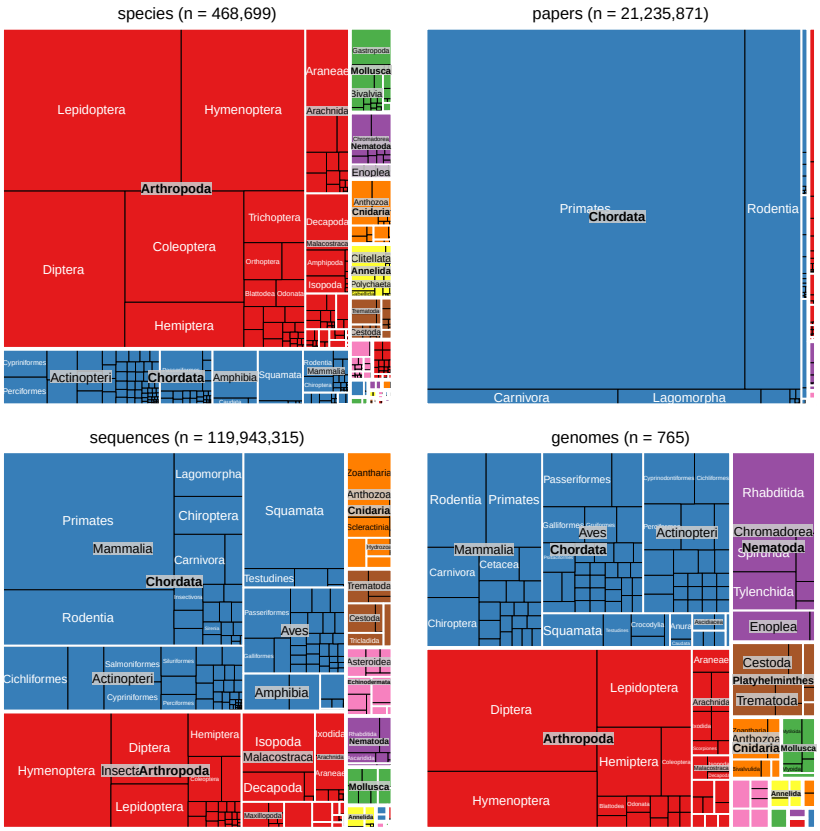


Figure 1: Taxonomic diversity of various NCBI databases (considering only animals). Panels in each plot are scaled to represent the number of database records corresponding to a given taxonomic rank and are shaded to reflect the phylum to which they belong. Starting from the upper left in clockwise direction subplots represent number of species in NCBI Taxonomy, the number of papers in PubMed, the number of sequences in NCBI Nucleotide database and the number of nuclear genome sequences in NCBI Genome database.

```
animal_orders$sequences <- taxon_records(animal_orders$order, db="nuccore")
animal_orders$papers <- taxon_records(animal_orders$order, db="pubmed")
head(animal_orders,3)
```

```
#>     phylum       class           order species genomes sequences papers
#> 1 Chordata Actinopteri    Lutjaniformes     279       0      9315      0
#> 2 Chordata Actinopteri     Gerreiformes      62       0       901      0
#> 3 Chordata Actinopteri Priacanthiformes      34       0       602      0
```

The resulting data can be used to visualise the taxonomic diversity of NCBI databases (Figure 1). The Appendix to this paper includes code that takes advantage of **treemap** (Tennekes, 2017) to produce these visualisations.

Continued development of rentrez

rentrez covers the complete eUtils API, is well-documented at the function and package level and includes an extensive test suite that covers internal functions as well as typical use-cases of the software. The current version of **rentrez** is thus considered a stable release, and it is unlikely any additional functionality will be added. The software is nevertheless still actively maintained to keep pace with CRAN and NCBI policies and to fix any bugs that arise. Software issues, including bug reports and requests for help with particular use-cases, are welcomed at the package's software repository: http://github.com/ropensci/rentrez.

Acknowledgements

Development of **rentrez** has benefited greatly from being part of the rOpenSci project. I am especially grateful to Scott Chamberlain for his guidance. I am also very grateful to everyone who has provided pull-requests or filed issues including Chris Stubben, Karthik Ram, Han Guangchun, Matthew O'Meara, Reed Cartwright and Pavel Fedotov.

Bibliography

G. Becker and M. Lawrence. *Genbankr: Parsing GenBank Files into Semantically Useful Objects*, 2017. R package version 1.2.1. [p523]

S. Chamberlain. *Fulltext: Full Text of 'Scholarly' Articles Across Many Data Sources*, 2016. URL https://CRAN.R-project.org/package=fulltext. R package version 0.1.8. [p523]

S. Kovalchik. *RISmed: Download Content from NCBI Databases*, 2017. URL https://CRAN.R-project.org/package=RISmed. R package version 2.1.7. [p520]

F. Michonneau, J. W. Brown, and D. J. Winter. rotl: An R package to interact with the open tree of life data. *Methods in Ecology and Evolution*, 7(12):1476–1481, 2016. URL https://doi.org/10.1111/2041-210x.12593. [p523]

E. Paradis, J. Claude, and K. Strimmer. APE: Analyses of phylogenetics and evolution in R language. *Bioinformatics*, 20:289–290, 2004. [p520]

J. Rani, S.Ramachandran, and A. R. Shah. *An R Package for Text Mining of PubMed Abstracts.*, 2014. R package version 1.0.5. [p520]

E. Sayers and D. Wheeler. *Building Customized Data Pipelines Using the Entrez Programming Utilities (eUtils)*. NCBI, 2004. [p520]

G. Schöfl. *Reutils: Talk to the NCBI EUtils*, 2016. URL https://CRAN.R-project.org/package=reutils. R package version 0.2.3. [p520]

M. A. Stravs, E. L. Schymanski, H. P. Singer, and J. Hollender. Automatic recalibration and processing of tandem mass spectra using formula annotation. *Journal of Mass Spectrometry*, 48(1):89–99, 2013. [p520]

C. Stubben. *Genomes: Genome Sequencing Project Metadata*, 2015. R package version 3.4.0. [p520]

M. Tennekes. *Treemap: Treemap Visualization*, 2017. URL https://CRAN.R-project.org/package=treemap. R package version 2.4-2. [p524]

H. Wickham. Tidy data. *The Journal of Statistical Software*, 59, 2014. URL http://www.jstatsoft.org/v59/i10/. [p523]

J. Zhou and Y. Shui. *MeSHSim: MeSH(Medical Subject Headings) Semantic Similarity Measures*, 2015. R package version 1.6.0. [p520]

David J. Winter
Institute of Fundamental Sciences, Massey University
Palmerston North 4442
New Zealand
ORCiD: 0000-0002-6165-0029
david.winter@gmail.com

Appendix

Code used to produce Figure 1, using `animal_orders` data generated above.

```
# Format the total number of records for a graph title
make_title <- function(col_name, data){
    n <- sum(data[,col_name])
    with_commas <- formatC(n, format = "d", big.mark = ",")
```

```
    paste0(col_name, " (n = ", with_commas, ")")
}

# Generate a treemap from taxonmic data.frame
# * data= tidy_taxonomy data.frame
# * size_col = name of column for tm tile-size
# * fill_col = name of column for tile-fil
# * row = plot row in 2x2 grid
# * col = plot col in 2x2 grid
# * pal = palette for fill
taxic_diversity_tm <- function(data, size_col, fill_col, row, col, pal){
    treemap(data,
        index=c("phylum", "class", "order"), vSize=size_col, vColor=fill_col,
        palette=pal, type='categorical', position.legend="none",
        title=make_title(size_col, data), border.col=c("white","white","black"),
        vp = viewport(layout.pos.row = row, layout.pos.col = col)
    )
}

library(treemap)
library(grid)
library(gridExtra)
# 24 phyla means some fill-colours will be re-used, ordering phylum factor by spp
# will prevent any "major" phyla from getting the same colour.
spp_per_phylum <- aggregate(species ~ phylum, FUN=sum, data=animal_orders)
phyla_ordered <- spp_per_phylum$phylum[ order(spp_per_phylum$species, decreasing=TRUE)]
animal_orders$phylum<- factor(animal_orders$phylum, levels=phyla_ordered)
pal <-  rep(RColorBrewer::brewer.pal(8, name="Set1"), 3)

grid.newpage()
pushViewport(viewport(layout = grid.layout(2, 2)))

taxic_diversity_tm(animal_orders, "species",   "phylum", 1,1, pal)
taxic_diversity_tm(animal_orders, "papers",    "phylum", 1,2, pal)
taxic_diversity_tm(animal_orders, "sequences", "phylum", 2,1, pal)
taxic_diversity_tm(animal_orders, "genomes",   "phylum", 2,2, pal)
```

An Introduction to Rocker: Docker Containers for R

by Carl Boettiger, Dirk Eddelbuettel

Abstract We describe the Rocker project, which provides a widely-used suite of Docker images with customized R environments for particular tasks. We discuss how this suite is organized, and how these tools can increase portability, scaling, reproducibility, and convenience of R users and developers.

Introduction

The Rocker project was launched in October 2014 as a collaboration between the authors to provide high-quality Docker images containing the R environment (Boettiger and Eddelbuettel, 2014). Since that time, the project has seen both considerable uptake in the community and substantial development and evolution. Here we seek to document the project's objectives and uses.

What is Docker?

Docker is a popular open-source tool to create, distribute, deploy, and run software applications using *containers*. Containers provide a virtual environment (see Clark et al. (2014) for an overview of common virtual environments) requiring all operating-system components an application needs to run. Docker containers are lightweight as they share the operating system kernel, start instantly using a layered filesystem which minimizes disk footprint and download time, are built on open standards that run on all major platforms (Linux, Mac, Windows), and provide an added layer of security by running an application in an isolated environment (Docker, 2015). Familiarity with a few key terms is helpful in understanding this paper. The term "container" refers to an isolated software environment on a computer. R users can think of running a container as analogous to loading an R package; a container is an active instance of a static Docker image. A Docker "image" is a binary archive of that software, analogous to an R binary package: a given version is downloaded only once, and can then be "run" to create a container whenever it is needed. A "Dockerfile" is a recipe, the source-code, to create a Docker image. Pre-built Docker images are publicly available through Docker Hub, which plays a role for central distribution similar to CRAN in our analogy. Development and contributions to the Rocker project focus on the construction, organization and maintenance of these Dockerfiles.

Design principles and use cases

Docker gives users very convenient access to pre-configured and pre-built *binary* images that "just work". This allows R users to access a wider-variety of ready-to-use environments than provided by either the R Project itself or, say, their distribution which will generally focus on one (current) release. For example, R users on Windows may run RStudio® Server or Shiny® Server locally just by launching a single command (once Docker itself is installed). Another common use-case is access to R-devel without affecting the local system. Here, we detail some of the principal use cases motivating these containerized versions of R environments, and the design principles that help make them work.

Portability: From laptop to cloud

One common use case for Rocker containers is to provide a fast and reliable mechanism to deploy a custom R environment to a remote server, such as Amazon Web Services Elastic Compute (AWS EC2), DigitalOcean, NSF's Jetstream servers (Stewart et al., 2015), or private or institutional server hardware. Rocker containers are also easy to run locally on most modern laptops using Windows, MacOS, or Linux-based operating systems. By sharing volumes with the local host, users can still manipulate files with familiar, native tools while performing computation through a reproducible, containerized environhment (Boettiger, 2015). Being able to test code in a predictable, pre-configured R environment on a local machine and to then run the same code in an identical environment on a remote server (*e.g.*, for access to greater RAM, more processors, or merely to free up the local machine from a long-running computation) is essential for low-friction scaling of analysis. Without such containerization, getting code to run appropriately in a remote environment can be a major undertaking, requiring both time and knowledge many would-be users may not have.

For instance, on any platform with Docker installed, the following Docker command will launch a Rocker container providing the RStudio® server environment over a web interface.

```
wget -qO- https://get.docker.com/ | sh
sudo docker run -p 8787:8787 -e PASSWORD=<PICK-A-PASSWORD> rocker/rstudio
```

The `docker run` option `-p` sets the port on which RStudio® will appear, for which 8787 is the default (adding your user to the docker group avoids the need for a `sudo` command to call `docker`: `sudo usermod -g docker $USER`). Many academic and commercial cloud providers make it possible to execute such code snippets when a container is launched, without ever needing to `ssh` into the machine. The user may log into the server merely by pasting its IP address or DNS name (followed by the chosen port, *e.g.*, `:8787`) into a browser and entering the appropriate password. This provides the user with a familiar, interactive environment running on a remote machine while requiring a minimum of expertise.

This portability is also valuable in an instructional context. Requiring students to install all necessary software on personal laptops can be particularly challenging for short workshops, where download and installation time and troubleshooting across heterogeneous machines can prove time consuming and frustrating for students and instructors alike. By deploying a Rocker image or Rocker-derived image (see *Extensibility*) on a cloud machine, an instructor can easily provide all students access to the pre-configured software environment using only the browser on their laptops. This strategy has proven effective in our own experience in both workshops and semester-length courses. Similar Docker-based cloud deployments have been scaled to courses of 100s of students, *e.g.*, at Duke (Cetinkaya-Rundel and Rundel, 2017) and UC Berkeley (UC Berkeley, 2017).

HPC application

The portability of Rocker images can be particularly valuable in High Performance Computing contexts Setting up a specific R environment on High Performance Computing platforms and other centrally administrated multi-user machines or clusters has traditionally been challenging due to restrictions on root access that may be needed to install certain libraries. Versions of R and packages installed by the system administrator may also lag behind the most recent releases. Deploying Docker containers on HPC systems has previously been more very problematic since most system administrators do not want to allow the elevated user permissions the Docker runtime environment requires. To work around this problem, Lawrence Berkeley National Labs (LBNL) has made 'Singularity' (Lawrence Berkeley National Laboratories, 2017): a container runtime environment that users can both install and use to run most Docker containers without requiring root privileges. Singularity has seen rapid adoption in the HPC community (`http://singularity.lbl.gov/install-request`). Rocker containers can be run through Singularity with a single command much like the native Docker commands, *e.g.*

```
singularity exec docker://rocker/tidyverse:latest R
```

More details can be found in the Singularity documentation.

Interfaces

An important aspect of the Rocker project design is the ability for users to interact with the software on the container through either an interactive shell session (such as the R shell or a bash shell), or through a web browser accessing the RStudio® Server integrated development environment (IDE). Traditional remote and high-performance computing workflows for R users have usually required the use of `ssh` and a terminal-only interface, posing a challenge for interactive graphics and a barrier to users unfamiliar with these tools and environments. Accessing an RStudio® container through the browser removes these barriers. Rocker images include the RStudio-server software pre-installed and configured with the explicit permission of RStudio® Inc.

Users can access a `bash` shell running as root within a Rocker container using

```
docker exec -ti <container-id> bash
```

which can be useful for administrative tasks such as installing system dependencies. All Rocker images can also be run as an interactive R, RScript or bash shell without running RStudio, which can be useful for batch jobs or for anyone who prefers that environment.

As with any interactive Docker container, users should specify the terminal (`-t`) and interactive (`-i`) flags, (here combined with interactive as `-ti`), and specify the desired executable environment (*e.g.*, `R`, though other common options could be `Rscript` or `bash`):

```
docker run --rm -ti rocker/tidyverse R
```

This example shows the use of the `--rm` flag to indicate that the container should be removed when the interactive session is finished. Details on sharing volumes, managing user permissions, and more can be found on the Rocker website, https://rocker-project.org.

Sandboxed

Another feature of Rocker containers is the ability to provide a sandboxed environment, isolated from software and potentially from other data on the machine. Many users are reluctant to upgrade their suite of installed packages, which may break their existing code or even their R environment if the installation goes poorly. However, upgrading packages and/or the R environment is often necessary to run analyses from a colleague, or access more recent methods. Rocker offers an easy solution. For instance, a user can run R code requiring the most recent versions of R and related packages inside a Rocker container without having to upgrade their local installations first. Conversely, one could use Rocker to run code on an older R release with prior versions of R packages, again without having to make any alteration to one's local R install. Another common use case is to access a container with support for particular options such as using gcc or clang compiler sanitizers (Eddelbuettel, 2014). These require R itself be built with specialized settings that may not be not available or familiar to many R users on their native system, but can be easily deployed by pulling the Rocker images `rocker/r-devel-san` or `rocker/r-devel-ubsan-clang`.

This sandboxing feature is also valuable in the remote computing context, allowing system administrators to grant users freedom to install software which requires root privileges inside a container, while not granting them root access on the host machine. Root access is required to launch Docker containers, though not to access containers already running and providing some service such as RStudio. Users logging into a container through the RStudio® interface do not by default have root privileges, though are able to install R packages. Granting these users root privileges in the container still leaves them sandboxed from the host container. Sandboxing also serves an important function in reproducible research by making it easier to test a specified environment in isolation from the host machine. Unlike traditional virtual machines, containers do not impose a large footprint of reserved resources as a typical host can easily support 100s of containers (Docker, 2015).

Transparent

Users can easily determine the software stack installed on any Rocker image by examining the associated Dockerfile recipe, which provides a concise, human-readable record of the installation. All Rocker images use automated builds through Docker Hub, which also acts as the central, default repository distributing the images. Using automated builds rather than uploading pre-built image binaries to Docker Hub avoids the potential for the build not to match the recipe. The corresponding Dockerfile is visible both on the Docker Hub and in the linked GitHub repository, which provides a transparent versioned history of all changes made to these recipes, as well as documentation, a community wiki, and issue trackers for discussing proposed changes, bugs, improvements to the Dockerfiles and troubleshoot any issues users may encounter. Having these public source files built automatically by a trusted provider (Docker Hub), rather than built locally and uploaded as binaries, is also useful from a security perspective in avoiding malware.

Community optimized

Having a shared, transparent computational environment created by a publicly hosted, reproducible recipe facilitates community input into configuration details. R and many of its packages and related software can be configured with a wide range of options, compilers, different linear-algebra libraries and so forth. While this flexibility reflects varying needs, many users rely on default settings which are most often are optimized more for simplicity of installation rather than than performance. The Rocker recipes reflect significant community input on these choices. This helps create a more finely tuned, optimized reference implementation of the R environment as well as a platform for comparing and discussing these concerns which are often overlooked elsewhere. Issues and Pull Requests on the Rocker repositories on GitHub attest to some of these discussions and improvements. In particular, input from the Docker Inc. employees through the official approval process for the `r-base` image, expertise from the Debian R maintainer and other Debian developers, and both direct and indirect feedback from the experience and user-generated documentation from many early adopters in the R community has helped shape and strengthen the project over the past few years. Widespread use of the Rocker image helps promote both testing of these choices and contributions, further tweaking the configuration from many members of the R community.

Versioned

Access to specific versions of software can be important for users who need computational reproducibility more than having the latest release of any piece of software, since subsequent releases can alter the behavior of code, introduce errors or otherwise alter previous results. The versioned stack (`r-ver`, `rstudio`, `tidyverse`, `verse`, and `geospatial`) provides images which are intended to build an identical software stack every time, regardless of the release of new libraries and packages. Users should specify an R version tag in the Docker image name to request a version stable image, *e.g.*, `rocker/verse:3.4.0`. If no tag is explicitly requested, Docker will provide the image with the tag `:latest`, which will always have the latest available versions of the software (built nightly).

Users building on the version-tagged images will by default use the MRAN snapshot mirror (Revolution Analytics, 2017) associated with the most recent date for which that image was current. This ensures that a Dockerfile building `FROM rocker/verse:3.4.1` will only install R package versions that were available on CRAN on 2017-06-30, *i.e.*, the day R 3.4.1 was released. This default can of course be overwritten in the standard R manner, *e.g.*, by specifying a different CRAN mirror explicitly in any command to install packages, *e.g.*, `install.packages()`, or by adjusting the default CRAN mirror in `options(repo=<CRAN-MIRROR>)` in an `.Rprofile`. Note that the MRAN date associated with the current release (*e.g.*, `3.4.2` at the time of writing) will continue to advance on the Docker-hub image until the next R release. Software installed from `apt-get` in these images will come from the the stable Debian release (`stretch` or `jessie`) and thus not change versions (though it will receive security patches). Packages installed from BioConductor using the `bioclite()` utility will also install the version appropriate to the version of R found on the system (the Bioconductor semi-annual release model avoids the need for an MRAN mirror). Users installing packages from GitHub or other sources can request a specific git release tag or hash for a more reproducible build, or adopt an alternative approach such as **packrat** (Ushey et al., 2016). A more general discussion of the use and limitations of Docker for computational reproducibility can be found in Boettiger (2015).

Extensible

Any portable computational environment faces an inevitable tension between the "kitchen sink problem" at one extreme, and the "discovery problem" on the other. A kitchen sink image seeks to accommodate too many use cases in a single image. Such images are inevitably very large and thus slow or difficult to deploy, maintain and optimize. At the other extreme, providing too many specialized images makes it more difficult for a user to discover the one they need. The Rocker project seeks to avoid both of these problems by providing a carefully-curated suite of images that an be easily extended by individuals and communities.

To make extensions transparent and persistent, Rocker images can be extended by any user by writing their own Dockerfiles based on an appropriate Rocker image. The Dockerfiles in the Rocker stack should themselves provide a simple example of this, (as described in the following section). A user begins by selecting an appropriate base image for their needs: if the RStudio® interface is desired, a user might start with `FROM rocker/rstudio`; an image for testing an R package with compiled code might use `FROM rocker/r-devel-san`, and an image for reproducing a data analysis will probably select a stable version tag in addition to an appropriate base library, *e.g.*,: `FROM rocker/tidyverse:3.4.1`. Users can easily add additional software to any running Rocker image using the standard R and Debian mechanisms. Details on how to extend Rocker images can be found at `https://rocker-project.org`.

Sharing these Dockerfiles can also facilitate the emergence of extensions tuned to particular communities. For instance, the `rocker/geospatial` image emerged from the input of a number of Rocker users all adding common geospatial libraries and packages on top of the existing Rocker images. This coalescence helped create a more fine-tuned image with broad support for a wide range of commonly-used data formats and libraries. Other community images are developed and maintained independently of the Rocker project, such as the `popgen` image of population-genetics-oriented software developed by the National Evolutionary Synthesis Center (NESCent). Rocker images are also being used as base Docker images in the NSF sponsored Whole Tale project for reproducible computing (Ludaescher et al., 2017), and are heavily used by the **rhub** project in automated package testing (Csárdi, 2017).

Rocker organization and workflow

The Rocker project consists of a suite of images built automatically by and hosted on the Docker Hub, `https://hub.docker.com/r/rocker`. Source Dockerfiles, supporting scripts and documentation are hosted on GitHub under the organization `rocker-org`, `https://github.com/rocker-org`. The

issue tracker and pull requests are used for community input, discussions, and contributions to these images. The Rocker project wiki, https://github.com/rocker-org/rocker/wiki, provides a place to synthesize community-contributed documentation, use-cases, and other knowledge about using the Rocker images.

Images in the Rocker Project

The Rocker project aims to provide a small core of Docker images that serve as convenient 'base' images on which other users can build custom R environments by writing their own Dockerfiles, while also providing a 'batteries included' approach to images that can be used out of the box. The challenges of balancing diverse needs driven by very different use cases against the overarching goals of creating images that are still sufficiently light-weight, easy to use, and easy to maintain is a difficult art. The implementation in both individual Rocker images and image stacks can never perfect that balance for everyone, but today reflects the considerable community input and testing over the past few years.

All Rocker images are based on the Debian Linux distribution. It provides a small base image, the well-known `apt` package management system, and a rich ecosystem of software libraries, making it the base image of choice for Docker images, including many of the "official" images maintained by Docker's own development team. The Debian platform is also perhaps the best-supported Linux platform within the R community, including an active `r-sig-debian` listserve. The relatively long period between stable Debian releases (roughly two years recently) means that software in the Debian stable (*e.g.*, `debian:jessie`, `debian:stretch`) releases can lag significantly behind current releases of popular software, including R. More recent versions of packages can be found in the pre-release distribution, `debian:testing`, while the very latest binary builds can be found on `debian:unstable`. The Rocker project can be largely divided into two stacks which address different needs, reflected in which Debian distribution they are based on. The first stack is based on `debian:testing`. The second, more recently-introduced stack, is based only on Debian stable releases. Rocker images always point to specific stable releases (`jessie`, `stretch`), and do not use the tag `debian:stable`, which is a rolling tag that always points to the most recent stable version. The different Rocker stacks have different aims and thus provide different images, as shown in Tables 1 & 2 below.

The `debian:testing`-based images

The `debian:testing` stack aims to make the most efficient use of upstream builds: the pre-compiled `.deb` binaries provided by the Debian repositories. It is both quicker and easier to install software from binaries, since the package manager (`apt`) manages the necessary (binary) dependencies and bypasses the time-consuming process of compiling from source. Basing this stack on `debian:testing` means that much more recent versions of commonly-used libraries and compilers are available as binaries than would be found in a Debian stable release. In order to provide *optional* access to the most recent available binaries, this stack uses apt-pinning (Debian Project, 2017) to allow the `apt` package manager to *selectively* install binaries from `debian:unstable`, which represents the most recent set of packages built for Debian. Similarly, recent versions of many popular R packages can also be installed pre-built through the package manager, *e.g.*, `apt-get install r-cran-xml`. This can be particularly helpful for packages with external system dependencies (such as `libxml2-dev` in this example) which cannot be installed from the R console as they are system dependencies rather than R packages installed from within R. We should note, however, that only about 500 of the over 11,000 CRAN packages are available as Debian packages.

As the names `testing` and `unstable` imply, particular versions of package can change as packages move from `unstable` into `testing`. New versions are sent to `unstable` during the normal course of Debian development. This can occasionally break a previously-working installation command in a Dockerfile until the maintainer redirects the package manager to install a package from the `unstable` sources that could previously be installed from `testing`, or vice versa (using the `-t` option in `apt`). That said, packages only migrate from `unstable` to `testing` after a period of several days—and if the migration and installation of the particular version is free of interactions with other packages in their dependency graph. That way, `unstable` serves as validation lab which leaves `testing` reasonably stable yet current.

Relative to `stable`, the `testing` stack thus offers some advantages as almost all software can be installed through the package manager. Installation of binary packages from `testing` generally provides the most recent available software, and installs it quickly as a binary. On the other hand, these Dockerfiles may require occasional maintenance when packages migrate and/or versions change. The resulting images are also inherently dynamic: rebuilding the same Dockerfile months or years apart will generate images with significantly different versions of software installed as the pool of

underlying packages changes through time.

Images overview

The `debian:testing`-based stack currently includes seven images actively maintained by the Rocker development team (Table 1). `r-base` builds on `debian:testing`, and the other six in the stack each build directly from `r-base`. The `r-base` image is unique in that it is designated as the official image for the R language by the Docker organization itself. This official image is reviewed and then built by employees of Docker Inc. based on a Dockerfile maintained by the Rocker team. Consequently, users should refer to this image in Docker commands without an organization namespace, *e.g.*, `docker run -ti r-base` to access the official image. All other images in the Rocker project are not individually reviewed and built by Docker Inc. and must be referenced using the `rocker` namespace, *e.g.*, `docker run -ti rocker/r-devel`.

Several of the images in this stack are oriented towards the R development community: `r-devel`, `drd`, `r-devel-san`, and `r-devel-ubsan-clang` which all add a copy of the development version of R side-by-side to the current release of R provided by `r-base`. On these images, the development version is aliased to `RD` to distinguish from the current release, `R`. As the names suggest, each provide slightly different configurations. Of particular interest are the images providing development R built with support for C/C++ address and undefined-behavior sanitizers, which are somewhat difficult to configure (Eddelbuettel, 2014).

As these images focus on developers and/or as base images for custom uses, this stack does not include many specific R packages. Additional dependencies and packages can easily be installed from `apt`. R packages not available in the `apt` repositories can be installed directly from CRAN using either `R` or the `littler` scripts, as described in https://rocker-project.org/use.

This stack also includes the images `shiny` and `rstudio:testing` that provide Shiny server and RStudio® server IDE from RStudio® Inc, built on the `r-base` image. RStudio® and Shiny are registered trademarks of RStudio Inc, and their use and the distribution of their software in binary form on Docker Hub has been granted to the Rocker project by explicit permission from RStudio. Users should review RStudio®'s trademark use policy (http://www.rstudio.com/about/trademark/) and address inquiries about further distribution or other questions to permissions@rstudio.com. The Rocker project also provides images with RStudio® server and Shiny server in the stable versioned stack.

Build schedule: The official `r-base` image is rebuilt by Docker following any updates to the official `debian` images (roughly every few weeks). The rest of the stack uses build triggers that rebuild the images whenever `r-base` is updated or the Dockerfile sources are updated on the corresponding GitHub repository. The only exception in this stack is the `drd` image, which is rebuilt each week by a `cron` trigger.

Table 1: The `debian:testing` image stack

image	description	size	downloads
r-base	official image with current version of R	254 MB	632,000
r-devel	R-devel added side-by-side to r-base (using alias `RD`)	1 GB	4,000
drd	lightweight r-devel, built weekly	571 MB	4,000
r-devel-san	as r-devel, but built with compiler sanitizers	1.1 GB	1,000
r-devel-ubsan-clang	sanitizers, clang c compiler (instead of gcc)	1.1 GB	525
rstudio:testing	rstudio on debian:testing	1.1 GB	1,000
shiny	shiny-server on r-base	409 MB	123,000

Table 2: The `rocker-versioned` stack of images

image	description	size	downloads
r-ver	version-stable base R & src build tools	219 MB	6,000
rstudio	adds rstudio	334 MB	314,000
tidyverse	adds tidyverse & devtools	656 MB	83,000 [1]
verse	adds java, tex & publishing-related packages	947 MB	9,000
geospatial	adds geospatial libraries	1.3 GB	4,000

[1] This figure includes 49,000 downloads under the earlier name `hadleyverse`.

The debian:stable-based stack

This stack emphasizes stability and reproducibility of the Docker build. This stack was introduced much more recently (November 2016) in response to considerable user input and requests. The key feature of this stack is the ability to run older versions of R along with the then-contemporaneous versions of R packages. A user specifies the version desired using an image tag, *e.g.*, `rocker/r-ver:3.3.1` will refer to an image with R version 3.3.1 installed. Omitting the tag is equivalent to using the tag `latest`, which, as the name implies, will always point to an image using the current R release. Thus, users who want to create downstream Dockerfiles, which are based on the current release at the time (but will continue to reconstruct the same environment in the future after newer R versions are released), should explicitly include the corresponding version tag, *e.g.*, `rocker/r-ver:3.4.2` at the time of writing, and not the `latest` tag. Users can also run the current development version of R using the tag `devel`, which is built nightly from R-devel sources from `subversion`.

MRAN archives: To facilitate installation of only contemporaneous versions of R packages on these images, the default CRAN mirror from which to install R packages is fixed to a snapshot of CRAN corresponding to the last date for which that version of R was current (*e.g.*, `3.4.2` was released on 2017-09-28, thus `3.4.1` is pinned to the MRAN snapshot for that date). These snapshots are provided by the MRAN archive created by Revolution Analytics (now part of Microsoft). It archives daily snapshots of all of CRAN from which a user can install packages with the usual `install.packages()` function (Revolution Analytics, 2017). Users can always override this default by passing any current CRAN repository explicitly. Unlike CRAN, Bioconductor only updates its repositories through bi-annual releases aligned to R's spring release schedule. Thus, Bioconductor packages can be installed in the usual way using `bioclite`, which automatically selects the Bioconductor release corresponding to the version of R in use.

Version tags: The version tags are propagated throughout this stack: *e.g.*, `rocker/tidyverse:devel` will provide the currently-released versions of the R packages in the **tidyverse** (Wickham, 2017) installed on the nightly build of R-devel. Developers building packages on this stack are encouraged to tag their images accordingly as well. Table 3 indicates which versions of R are currently available in the stack, going back to `3.1.0`. While older versions may be added to the stack at a later date, we note that the MRAN snapshots began in 2014-09-17 and thus go back only to the R `3.1` era. Each tag must be built from a separate Dockerfile, enabling minor differences in the build instructions to accommodate changing dependencies. Dockerfiles for past versions (*e.g.*, prior to `3.4.2` currently) are intended to remain static over the long term, while the tag for the current version, `latest`, and `devel` may be tweaked to accommodate new features or dependencies. Version tags also obey semantics so that omitting the second or third position of the tag is identical to asking for the most recent version: *i.e.*, `rocker/verse:3.3` is the same as `rocker/verse:3.3.3`, and `rocker/verse:3` is (at the time of writing), `rocker/verse:3.4.2`. This is accomplished using post-build hooks in Docker Hub—see examples at `https://github.com/rocker-org/rocker-versioned/` for details.

Installation: In this stack, the desired version of R is always built directly from source rather than the `apt` repositories. Compilers and dependencies are still installed from the stable `apt` repositories, and thus lag behind the more recent versions found in the `testing` stack. Version tags `3.3.3` and older are based on the Debian 8.0 release, code-named `jessie`, while `3.4.0` - `3.4.2`, `devel`, and `latest` are based on Debian 9.0, `stretch`, (released 2017-06-17, while R was at `3.4.0`), and thus have access to much newer versions of common system dependencies and compilers. Dependencies needed to compile R that are not required at runtime are removed once R is installed, keeping the base images light-weight for faster download times. While most system dependencies required by common R packages can still be installed from the `apt` repositories, occasionally a more recent version must be compiled from source (*e.g.*, the Gibbs Sampling program JAGS (Plummer, 2017), and the geospatial toolkit GDAL, must both be compiled from source on `debian:jessie` images). In this stack, users should avoid installing R packages using `apt` without careful consideration as this will install a second (probably different) version of R from the Debian repositories, and a dated version of the R package since any `r-cran-pkgname` package in the Debian repositories will depend on `r-base` in `apt` as well.

Build schedule: All images are built automatically from their corresponding Dockerfiles (found in the GitHub repositories `rocker-org/rocker-versioned` and `rocker-org/geospatial`). A `cron` job sends nightly build triggers to Docker Hub to rebuild the `latest` and `devel` tagged images throughout the stack. To decrease load on the hub, build triggers for the numeric version tags are sent monthly. Although the Dockerfiles for older R versions install an almost-identical software environment every time, the monthly rebuilding of these images on Docker Hub ensures they continue to receive Debian security updates from upstream, and proves the build recipe still executes successfully. Note that rebuilding images with software from external repositories never produces a bit-wise identical image, and thus the image identifier hash will change at each build.

Images overview

In this stack, each image builds on the previous image, rather than all other images building directly on the base image, as in the `testing` stack. Table 2 lists the names and descriptions of the five images in this stack, along with image size and approximate download counts from Docker Hub. Sizes reflect (compressed) cumulative size: a user who has already downloaded the most recent version of `r-ver` and then pulls a copy of `rstudio` image will only need to download the additional 115 MB in the `rstudio` layers and not the full 334 MB listed. This linear design limits flexibility (no option for `tidyverse` without `rstudio`) but simplifies use and maintenance. While no single environment will be optimal for everyone, both the packages selected in this stack and the stack ordering reflect considerable community input and tuning.

The `rstudio` image includes a lightweight, easy-to-use and docker-friendly `init` system, s6 (Bercot, 2017) for running persistent services, including the RStudio® server. This system provides a convenient way for downstream Dockerfile developers to add additional persistent services (such as an `ssh` server) to a single container, or additional start-up or shutdown scripts that should be run when a container starts up or shuts down. The `rstudio` image uses such a start-up script to configure user settings such as login password and permissions through environmental variables at run time.

The `tidyverse` image contains all required and suggested dependencies of the commonly-used `tidyverse` and `devtools` R packages, including external database libraries (*e.g.*, MariaDB and PostgreSQL). Users should consult the package Dockerfiles or `installed.packages()` list directly for a complete list of installed packages. The `verse` library adds commonly-used dependencies, notably a large but not comprehensive LaTeX environment and Java development libraries. Previously, the Rocker project provided the image `hadleyverse` which has since been divided into `tidyverse` and `verse` based on community input.

Table 3: Available tags in the `rocker-versioned` stack.

tag	apt repos	MRAN date	Build frequency	images with tag
devel	`stretch`	current date	nightly	`r-ver`, `rstudio`, `tidyverse`, `verse`, `geospatial`
latest	`stretch`	current date	nightly	`r-ver`, `rstudio`, `tidyverse`, `verse`, `geospatial`
3.4.2	`stretch`	current date	monthly	`r-ver`, `rstudio`, `tidyverse`, `verse`, `geospatial`
3.4.1	`stretch`	2017-09-28	monthly	`r-ver`, `rstudio`, `tidyverse`, `verse`, `geospatial`
3.4.0	`stretch`	2017-06-30	monthly	`r-ver`, `rstudio`, `tidyverse`, `verse`, `geospatial`
3.3.3	`jessie`	2017-04-21	monthly	`r-ver`, `rstudio`, `tidyverse`, `verse`, `geospatial`
3.3.2	`jessie`	2017-03-06	monthly	`r-ver`, `rstudio`, `tidyverse`, `verse`, `geospatial`
3.3.1	`jessie`	2016-10-31	monthly	`r-ver`, `rstudio`, `tidyverse`, `verse`, `geospatial`
3.3.0	`jessie`	2016-06-21	monthly	`r-ver`
3.2.0	`jessie`	2015-06-18	monthly	`r-ver`
3.1.0	`jessie`	2014-09-17	monthly	`r-ver`

Several images in the `rocker-versioned` stack can be customized on build when built locally (rather than pulling prebuilt images from Docker Hub) by using the `--build-arg` option of `docker build`. In the `r-ver` image, users can set `R_VERSION` and `BUILD_DATE` (MRAN default snapshot). In the `rstudio` image users can set `RSTUDIO_VERSION` (otherwise defaults to the most recent version), and the `PANDOC_TEMPLATES_VERSION`.

This stack also makes use of Docker metadata labels defined by `http://schema-label.org`, indicating image `license` (GPL-2.0), `vcs-url` (GitHub repository), and `vendor` (Rocker Project). These metadata can be altered or extended in downstream images.

Conclusions

Over the past several years, Docker has seen immense adoption across industry and academia. The Open Container initiative (The Linux Foundation: Projects, 2017) now provides an open standard

that has further extended this container approach to research environments through projects such as Singularity (Lawrence Berkeley National Laboratories, 2017), allowing users to deploy containerized environments such as Rocker on machines where they do not have root access, such as clusters or private servers. Containerization promises to solve numerous challenges such as portability and replicability in research computing, which often relies on complex and heterogeneous software stacks (Boettiger, 2015). Yet implementing such environments in containers is not a trivial task, and not all implementations provide the same usability, portability or reproducibility. Here we have detailed the approach taken by the Rocker project in creating and maintaining these environments through an open and community-driven process. This structure of the Rocker project has evolved over three years of operation while drawing in an ever-widening base of academic researchers, university instructors and industry users. We believe this overview will be instructive not only to users and developers interested in the Rocker project, but as a model for similar efforts around other environments or domains.

Bibliography

L. Bercot. *S6: Skarnet.org's Small and Secure Supervision Software Suite*, 2017. URL https://skarnet.org/software/s6/. [p534]

C. Boettiger. An Introduction to Docker for Reproducible Research, with Examples from the R Environment. *ACM SIGOPS Operating Systems Review*, 49(1):71–79, 2015. URL https://doi.org/10.1145/2723872.2723882. [p527, 530, 535]

C. Boettiger and D. Eddelbuettel. Introducing Rocker: Docker for R, 2014. URL https://ropensci.org/blog/blog/2014/10/23/introducing-rocker. [p527]

M. Cetinkaya-Rundel and C. W. Rundel. Infrastructure and tools for teaching computing throughout the statistical curriculum. *PeerJ Preprints*, 5:e3181v1, 2017. ISSN 2167-9843. URL https://doi.org/10.7287/peerj.preprints.3181v1. [p528]

D. Clark, A. Culich, B. Hamlin, and R. Lovett. BCE: Berkeley's Common Scientific Compute Environment for Research and Education. *Proceedings of the 13th Python in Science Conference (SciPy 2014)*, pages 1–8, 2014. [p527]

G. Csárdi. *Rhub: Connect to 'R-Hub', from 'R'*, 2017. URL https://github.com/r-hub/rhub. R package version 1.0.1. [p530]

Debian Project. Apt-preferences overview, 2017. URL https://wiki.debian.org/AptPreferences. [p531]

Docker. What is Docker?, 2015. URL https://www.docker.com/what-docker. [p527, 529]

D. Eddelbuettel. Sanitizers, 2014. URL http://dirk.eddelbuettel.com/code/sanitizers.html. [p529, 532]

Lawrence Berkeley National Laboratories. Singularity, 2017. URL http://singularity.lbl.gov/. [p528, 535]

B. Ludaescher, K. Chard, M. Turk, V. Stodden, and N. Gaffney. The Whole Tale: Merging science and cyberinfrastructure pathways, 2017. URL https://wholetale.org/. [p530]

M. Plummer. JAGS: A program for analysis of Bayesian graphical models using Gibbs sampling, 2017. URL http://mcmc-jags.sourceforge.net/. Version 4.3.0. [p533]

Revolution Analytics. Microsoft R Application Network, 2017. URL https://mran.microsoft.com. [p530, 533]

C. A. Stewart, G. Turner, M. Vaughn, N. I. Gaffney, T. M. Cockerill, I. Foster, D. Hancock, N. Merchant, E. Skidmore, D. Stanzione, J. Taylor, and S. Tuecke. Jetstream: A Self-Provisioned, Scalable Science and Engineering Cloud Environment. In *Proceedings of the 2015 XSEDE Conference on Scientific Advancements Enabled by Enhanced Cyberinfrastructure - XSEDE '15*, pages 1–8, New York, New York, USA, 2015. ACM Press. ISBN 9781450337205. URL https://doi.org/10.1145/2792745.2792774. [p527]

The Linux Foundation: Projects. The Open Container Initiative, 2017. URL https://www.opencontainers.org/. [p534]

UC Berkeley. Curriculum Overview | Division of Data Sciences, 2017. URL http://data.berkeley.edu/education/curriculum-overview. [p528]

K. Ushey, J. McPherson, J. Cheng, A. Atkins, and J. Allaire. *Packrat: A Dependency Management System for Projects and Their R Package Dependencies*, 2016. URL https://CRAN.R-project.org/package=packrat. R package version 0.4.8-1. [p530]

H. Wickham. *Tidyverse: Easily Install and Load 'Tidyverse' Packages*, 2017. URL https://CRAN.R-project.org/package=tidyverse. R package version 1.1.1. [p533]

Carl Boettiger
UC Berkeley
ESPM Department, University of California,
130 Mulford Hall Berkeley, CA 94720-3114, USA
ORCiD 0000-0002-1642-628X
cboettig@berkeley.edu

Dirk Eddelbuettel
Debian and R Projects; Ketchum Trading
Chicago, IL, USA
ORCiD 0000-0001-6419-907X
edd@debian.org

Conference Report: useR!2017

by Tobias Verbeke

Introduction

After a very successful 2016 edition in Stanford (US), the useR conference invited the R community to meet from July 4 to July 7 in Brussels (Belgium), heart of Europe. The response was extraordinary: 1175 people (of 54 nationalities) travelled the globe to join for a week of intense exchange and discussion. The conference was held in the Wild Gallery which was – for the occasion – the exclusive territory of R aficionados with many co-hosted events including DSC 2017, RIOT 2017 and an R Foundation meeting.

An important theme throughout the conference was to be welcoming and inclusive. In this respect 25 diversity scholarships were awarded and newbies were welcomed at a newbies session the evening before the tutorial day. Dedicated childcare was organized for the youngest R enthusiasts and besides the regular hacking and relaxing space, a breast-feeding area and quiet zone were foreseen.

The program consisted of 16 pre-conference tutorials, 6 invited talks, 147 oral presentations, 70 lightning talks and 70 poster sessions. The social program included a welcome reception, poster reception and conference dinner where a stand-up comedian offered an introduction to Belgium and a well known beer sommelier revealed the secrets of Belgian beers to a very receptive audience (914 liters to be precise).

Pre-conference tutorials

The pre-conference tutorials were free and open to all attendees.

- OpenML: Connecting R to the Machine Learning Platform OpenML – Joaquin Vanschoren, Heidi Seibold and Bernd Bischl
- Sports Analytics with R – Stephanie Kovalchik
- Environmental Modeling using R – Karline Soetaert and Thomas Petzoldt
- Introduction to parallel computing with R – Hana Sevcikova
- Introduction to Bayesian inference with JAGS – Martyn Plummer
- R Package Development with R-hub – Gabor Csardi
- purrr – Charlotte Wickham
- Spatial Data in R: New Directions – Edzer Pebesma
- Extending R with C++: Motivation, Introduction and Examples – Dirk Eddelbuettel
- Efficient R Programming – Colin Gillespie
- Introduction to Optimal Changepoint Detection Algorithms – Toby Dylan Hocking and Rebecca Killick
- Data Carpentry: Open and Reproducible Research with R: Colin Rundel, Mine Cetinkaya-Rundel
- data.table for Beginners – Arun Srinivasan
- Dose-response analysis using R – Signe M. Jensen and Christian Ritz
- Geospatial Visualization using R – Bhaskar V. Karambelkar
- Introduction to Natural Language Processing with R – Taylor Arnold and Lauren Tilton

Invited talks

As befits an R conference, the invited talks kept statistics and computing in balance with the following keynote speakers:

- Structural Equation Modeling: Models, Software and Stories – Yves Rosseel
- Teaching Data Science to new useRs – Mine Cetinkaya-Rundel
- Dose-Response Analysis: Considering Dose both as Qualitative Factor and Quantitative Covariate, using R – Ludwig Hothorn
- Parallel Computation in R: What We Want, and How We (Might) Get It – Norm Matloff
- R Tools for the Analysis of Complex Heterogeneous Data – Isabella Gollini
- 20 Years of CRAN – Uwe Ligges

Contributed sessions

Fourty-two sessions were held in six parallel tracks demonstrating the ever rich diversity of R usage with contributions in Bioinformatics, Business and Management, Clustering, Community Data management, Data reproducibility, Education, GIS, Graphics, HPC, Kaleidoscope, Lightning Talks (six sessions!), Machine Learning, Medical statistics, Methods, Methods in Business, Missing Data, Packages, Programming, Shiny, Social, Statistical Modelling, Text Mining and Web.

Conference organizers

The quality of the scientific program of the conference was the achievement of Ziv Shkedy (chair), Heather Turner (chair), Michela Battauz, Przemyslaw Biecek, Roger Bivand, Di Cook, Dirk Eddelbuettel, Bettina Gruen, Torsten Hothorn, Julie Josse, Helena Kotthaus and Tobias Verbeke. The organization was in the hands of Tobias Verbeke (chair), Ziv Shkedy, Heather Turner and Matthias Verbeke.

Additional information

Conference website https://user2017.brussels/

Video Recordings https://channel9.msdn.com/Events/useR-international-R-User-conferences/useR-International-R-User-2017-Conference

Aftermovie https://youtu.be/YWF6nbUTRao

Sponsors https://user2017.brussels/sponsors

Tobias Verbeke
Open Analytics NV
Jupiterstraat 20
2600 Antwerp, Belgium
tobias.verbeke@openanalytics.eu

Conference Report: R in Insurance 2017

by Nicolas Baradel, Christophe Dutang and Caroline Hillairet

Conference summary

The fifth R in Insurance conference took place at Ecole Nationale de la Statistique et de l'Administration Economique (ENSAE, one of the leading French graduate schools in the fields of statistics, economics, finance and actuarial science) Paris on 8 June 2017. This one-day conference focused once more on the wide range of applications of R in insurance, actuarial science and beyond. The conference programme covered topics including reserving, pricing, loss modelling, the use of R in a production environment and also new statistical methods such as big data analysis.

The audience of the conference included both practitioners (around 70%) and academics (30%) who are active or interested in the applications of R in Insurance. The fifth edition was a fair success with 128 participants compared to the 100 participants of the fourth edition. Furthermore, it was a truly international event with speakers and delegates from many different countries, including USA, Belgium, Netherlands, Switzerland, Germany, Italy, Qatar, UK, India and of course France. Overall, there were 17 speakers from USA, Belgium, Netherlands, Switzerland, Italy, UK, India and France. The coffee breaks and the lunch time offered great networking opportunities, while the conference dinner at Musée d'Orsay closed on a high note this enlighting day.

In the first plenary session, Julie Seguela (from Covea) spoke about the textual analysis of expert reports to increase knowledge of technological risks. She used open datasets from the ARIA database (Analysis, Research and Information about Accidents), which has collected more than 40 000 technological accidents, between 1995 and 2015 in France, susceptible to damage public health or safety, etc. Text mining techniques and some helpful visualization packages were used on expert reports detailing circumstances, causes and consequences of these accidents. This master class talk highlighted how various R packages can interact to achieve our goal.

This plenary talk was followed by two sessions to close the morning. The first session focused on big data analytics emphasizing new usage in the insurance industry. Then, the second session consisted of a series of lightning talks about R packages or R modelling. Thereafter, the afternoon started with the third session on non-life insurance with speeches rather theoretical on non-life reserving or vine copulas. The fourth and last session followed with topics in life insurance.

In the closing plenary talk, Katrien Antonio (Professor of Actuarial Science at KU Leuven, Belgium) presented recent development and challenges in non-life reserving. In order to be able to fulfill future liabilities, insurance companies approach micro-level reserving by using granular, detailed data on the development of individual claims. In her talk, she gave an overview of the research on micro-level reserving and presented ongoing developments of statistical modeling and data analytic tools for reserving with granular data. Her talk was illustrated on a large European dataset of liability claims (from private individuals) with monthly exposures using R.

All conference presentations are available on the conference website at `https://rininsurance17.sciencesconf.org/`.

Scientific committee and sponsors

The members of the scientific committee were: Arthur Charpentier (University of Amsterdam and KU Leuven), Christophe Dutang (Université du Maine and Université Paris Dauphine, France), Markus Gesmann (ChainLadder project), Giorgio A. Spedicato (Unipol Gruppo Finanziario), Andreas Tsanakas (Cass Business School).

Finally, we are grateful to our sponsors RStudio, Verisk Insurance Solutions, Barnett

Waddingham, Mirai Solutions, Milliman. This conference would not have been possible without their generous support.

R in Insurance 2018

We are delighted to announce next year's event already on 16 July 2018. Following three years in London, one year in Paris and one year in Amsterdam, the conference will go back to London, UK. Further details will be published on https://insurancedatascience.org/.

Nicolas Baradel
ENSAE, Paris, France
http://www.nicolasbaradel.fr/

Christophe Dutang
Université du Maine, Le Mans, France
Université Paris Dauphine, Paris, France
http://dutangc.free.fr/

Caroline Hillairet
ENSAE, Paris, France
https://sites.google.com/site/carolinehillairet

Forwards Column

by Stella Bollmann, Dianne Cook, Jasmine Dumas, John Fox, Julie Josse, Oliver Keyes, Carolin Strobl, Heather Turner and Rudolf Debelak

Forwards is a task force that was set up by the R Foundation in 2015 to address the under-representation of women that has since widened its scope to encompass other under-represented groups. The task force is organised as a core team comprising leaders from a number of sub-teams that focus on particular aspects:

Community General outreach to help people from under-represented groups get into R and develop as useRs. Members have represented the task force at events such as AlterConf, Trans*Code International Transgender Day of Visibility Hackathon, National Federation of the Blind Convention, and Society for Advancement of Chicanos/Hispanics and Native Americans in Science. The team promotes outreach schemes such as NASA Datanauts and works alongside others in the R community seeking to widen participation, such as R-Ladies or other sub-teams, for example co-ordinating the diversity scholarship scheme for useR! 2017.

Conferences With a particular focus on R Foundation conferences, this team liaises with the organizers/program committee on policies and inclusion initiatives. For example this team initiated the conference buddy scheme for useR! 2017 and collaborated with R-Ladies to host a special session for newcomers.

On-ramps Creating paths for useRs to develop their skills and make contributions to the R/Bioconductor package ecosystem. Activities have included speaking at the useR! 2017 newcomer session and R-Ladies meetings on collaborative coding.

Social Media Managing the Twitter account and the recently started Facebook group to support people from under-represented groups. Maintaining the website and co-ordinating the blog.

Surveys Running and analysing community surveys; publishing corresponding reports and data. For example, this team ran a survey at useR! 2016 to find out about the demographics, programming experience and community involvement of useR! attendees.

Teaching Working on methodology, materials and workshops designed for under-represented groups. In particular, this team have developed materials for two workshops: one aimed at high school girls, on creating a data analysis web application and another aimed at women, on package development. So far these workshops have been run in Australia and New Zealand, with repeats planned for Europe and North America.

This new column provides an outlet for news about the work of the task force as well as more detailed reports. This inaugural column presents an article on the results of the useR! 2016 survey.

A First Survey on the Diversity of the R Community

Abstract The study presented here is a first attempt to capture the demographics and opinions of the R community, starting with the attendees of the useR! conference 2016. One aim of Forwards, the R Foundation taskforce on women and other under-represented groups, is to identify groups that are under-represented in the R community and to further stimulate ideas and take initiatives for widening their participation. Since R is an open-source software with various platforms for exchange, however, it is difficult to obtain information about its community – let alone define this community in the first place. As a starting point, a survey was conducted with the attendees of the useR! conference 2016 to document their sociodemographic and computational backgrounds, experiences and opinions. The present paper gives an overview of the results of this first survey. Most of the analysis focuses on women participants, that are generally under-represented in STEM (Science, Technology, Engineering, Mathematics) disciplines, but the results also show a severe under-representation of minorities. A surprising finding concerns a gender difference with regard to the experience with R and the publication of R packages. We investigated possible reasons for this difference by the means of a logistic regression analysis. The self-evident limitations of this first survey are discussed and directions for future research as well as potential means for improvement are outlined.

The R environment for statistical computing is an open-source project for data analysis. It provides an opportunity for users worldwide to benefit from an extensive number of software tools for a wide spectrum of data analysis and also allows users to participate in the project. This participation may take many forms. Beginners may attend tutorials on data analysis with R and contribute through their feedback while advanced users may write their own code and even publish it on one of the public repositories. As a whole, we will refer to the participants of the R project as the "R community" – but be assured of our awareness that this community is very hard to grasp.

Several studies have already investigated important aspects of the R community. A central place in the development of R is taken by the R Core Team. It is responsible for the development of the basic R software and the maintenance of infrastructure which is necessary for its distribution. Several interesting aspects of the organization and work of the R Core Team have been summarized by Fox (2009), who conducted a series of interviews with its members. Mair et al. (2015) investigated the motivation and values of the authors of R software packages published on the CRAN, Bioconductor and R-Forge repositories by the means of a questionnaire. Like the R Core Team, package authors are a well-defined subgroup of the R community that is quite easily accessible because their names are made public. However, very little is known about the remaining part of the R community, which consists of regular R users as well as developers who have not published R packages on public repositories so far. They are usually anonymous and can not be directly addressed.

Therefore, in a first attempt to start gathering information on the R community, we turned to a subsample of the community that was more clearly defined, namely the attendees of the useR! conference 2016, held in Stanford, CA. The advantage of the useR! conference as a platform for the survey – besides feasibility – was that it is attended by both R users and R developers, which is an important prerequisite for getting a broader insight than the previous studies on R Core members and CRAN package developers.

Yet we are fully aware that the useR! attendees are not a representative sample from the R community as a whole, because conference attendance itself depends on a variety of resources. This is particularly true for an expensive location like Stanford. Therefore, in this study we don't claim that our results can be generalized to other parts of the R community – nevertheless they tell us something about a quite relevant part of this community.

Speaking in terms of study design, we can only consider the useR! attendees – not the R community as a whole – as the population, from which again only a sample answered the questionnaire. We did, however, achieve a rather high return rate of 455/904 and also checked the representativeness of this sample as compared to the population of useR! attendees with respect to demographic information provided at or derived from registration[1]. Like most studies, we found that women were slightly more willing to participate in the survey compared to men, however this imbalance only becomes relevant when men and women differ in their responses, in which case we present results for men and women separately.

All attendees of the useR! conference 2016 received an invitation to participate in the survey. The questionnaire was presented online and contained 26 questions, which concerned demographic information, experience with and opinions about R, and involvement in the R community. Our results will be presented in two sections: In the first section, we report results on the demographic data. In the second section, we summarize the responses concerning the participants' usage of R and their involvement in the R community.

[1] For more detail see our supplementary report Non-Responses in the UseR! 2016 Survey

The main goal of the survey was to obtain information from R users and developers that may help in setting up more inclusive infrastructure for the usage, development and teaching of R. The results may also point out prerequisites for successfully developing R expertise and hopefully initiate a dialogue with under-represented subgroups of the R community in order to help them formulate their needs.

While the initial focus of the survey, as well as the task force itself, was on addressing the under-representation of women, we did also investigate the representation of minority groups identified by race and sexuality. We will show that these minorities are severely under-represented. However, for a detailed statistical analysis the group sizes were so small that conclusions would have been questionable and identifiability of individual respondents would have become an issue.

Given that this survey was the first attempt towards addressing a broader part of the R community, both the structure of the questionnaire and its analysis are of a rather explorative nature. This, together with the above-mentioned fact that the respondents are a representative sample from the attendees, but not necessarily from the R community as a whole, has led us to formulate any conclusions carefully and emphasize effect sizes rather than significance in our statistical analysis.

Based on the work of Cohen (1988), we calculated Cohen's Omega as an effect size measure for describing associations between categorical variables. Cohen also provided guidelines for interpreting these measures: For Cohen's Omega, values around 0.1 correspond to a small effect, values around 0.3 correspond to a medium effect, and values around 0.5 correspond to a large effect. For all results presented here, a Cohen's Omega of 0.13 or higher also corresponded to a p-value below 0.05.

Demographic information

The sample consists of 455 respondents. We first investigated the basic characteristics of this sample, which include the age of the respondents, the number of women and men respondents, and the educational level of the respondents, as well as variables related to under-represented groups. Unless noted otherwise, all reported percentages refer to the relative size compared to the entire sample of 455 respondents.

Most respondents identified as men or women: an extremely small number of attendees had a gender outside the binary, but are not included in this analysis to avoid risking identifying them without their consent. Of the remaining respondents, 169 (37.22%) identified as women. 27 respondents (5.93%) reported to identify as members of the LGBT community, whereas 11 respondents (2.42%) did not reply to this question. When asked about their ethnicity and country of origin, a wide array of responses was given. We used a free text field in order to avoid deficiencies in standard race classification systems, which may not represent the respondents' identity. The open answer format does, however, make summarizing the results more difficult, so for this analysis we did compare the free answers to a standard classification system. There were 409 responses on the subject of racial identity. This number was used as the reference for the following percentages: 302 respondents (73.83%) identified as White/Caucasian. Only 7 attendees (1.71%) identified as Black, Native American (including Pacific Islanders) or Middle Eastern.

To put this in context, more respondents identified primarily as Jewish (8 responses, corresponding to 1.96%), than as Black, Middle Eastern or Native American combined. Of the other attendees, 67 (16.38%) fell within what the standard American Federal criteria would class as Asian, 12 (2.93%) as Hispanic/Latinx, and the remainder as mixed-race.

Given these results, there appears to be a severe under-representation of non-white attendees compared to the general population. These results might further indicate an under-representation of members of the LGBT community, although this question is more difficult to address. In the adult population of the United States, around 3.5% of the adult population consider themselves as part of the LBGT community, as was shown in the report of Gates (2011). However, these figures might constitute a severe underestimation, as was argued by Coffman et al. (2013). We are not able to further investigate these minorities and their possible reasons for not attending the conference for two reasons: First, the small sample sizes of the under-represented groups would make it difficult to generalize any conclusions. Still we do not find it justified to merge ethnic minorities into larger groups just for the sake of the analysis. Second, if a group consists of very few attendees, reporting more detailed findings could compromise the anonymity of individual respondents. Therefore, the remaining analysis will focus only on gender differences.

We do, however, want to point out that the under-representation of non-white attendees we found in the useR! survey was more severe than we would have expected and led to a broadening of the focus of the task force to represent not only the concerns of women but also of other under-represented groups, be they identified by race, gender, sexuality, class, or disability.

With respect to gender differences, we found that women and men respondents differed with

regard to their age (approximated from their birth year, see Figure 1), but not with regard to their educational level (with response options "max. secondary school", "undergraduate degree", "Masters degree", "Doctorate" and "Professional degree").

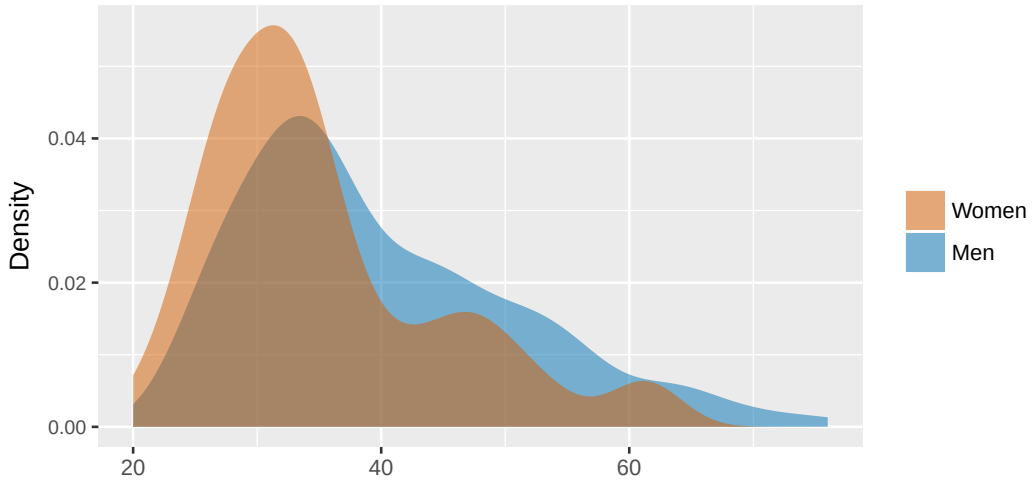


Figure 1: Approximate age of useR! 2016 attendees.

The median birth year of men respondents was earlier than that of women respondents, with men respondents on average being in their early 40s and women respondents being in their mid-30s.

Therefore, when comparing women and men respondents in the remainder of the analysis, care should be taken with the interpretation of gender differences in bivariate analyses, because they might be confounded with age differences.

Gender differences were indeed found in the reported employment status of the respondents (see Figure 2): A higher rate of men respondents tended to be employed in industry or to be permanently employed in academia. On the other hand, a higher rate of women respondents tended to be students. Part of these differences might be due to the age differences reported earlier.

Note that here and in all following figures comparing the answers of men and women respondents, the graphs display conditional relative frequencies of answering in a certain category given the gender. Accordingly, the percentages within one gender add up to 100% (missing values are not considered).

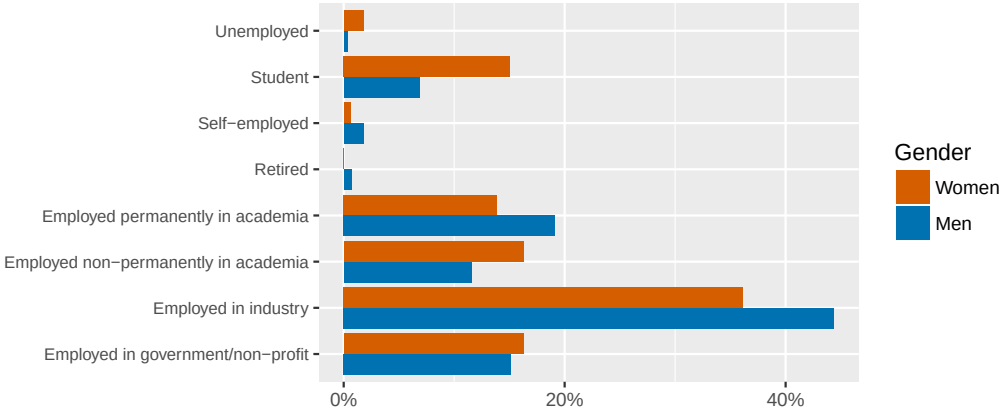


Figure 2: Employment status.

116 respondents (25.49%) reported to be caregivers for children or adult dependents on a regular basis, with men attendees being more likely to be caregivers than women attendees. 28.97% of the men respondents reported to be caregivers, while only 21.83% of the women respondents did. Even though this difference was relatively small (Omega = 0.08), we decided to report it since it demonstrates the self-selection effect inherent in this study: Our sample contained only those members of the R community who were able to attend the useR! conference in Stanford (and were also willing to answer the questionnaire).

At first glance, the fact that men respondents were more likely than women to be caregivers may sound like the R community might have overcome traditional role models already. But the more probable interpretation, that is also supported by the free text answers, is that traditional role models do still make it easier for men to leave children at home with their partner (if applicable) than for women to do the same, resulting in a self-selection effect in which women with children are less likely to be able to attend – and as a result women who do attend are less likely to be caregivers.

This again reflects the general problem of this survey: We cannot draw any conclusion about reasons why women or other under-represented groups did not attend the conference, since we do not have any information on them.

Opinion on R

After having reported demographical information about the useR! sample, we now report our findings on the opinions reported by the respondents towards R and working with R. Since the demographic information reported in the last section suggested that women respondents came from a different professional background than men respondents, we were particularly interested in further gender differences in their opinion towards R.

A first question was whether the respondents would recommend R to friends or colleagues as a programming language to learn. This was agreed to by 418 (91.87%) respondents, while 21 (4.62%) respondents disagreed and 19 (4.18%) respondents did not answer. Asked about their number one argument for or against learning R by selecting one argument from a list, numerous responses were given. We summarized these reponses discarding all answers that were given less than 10 times – this left only arguments for using R. Figure 3 summarizes these most frequently given answers.

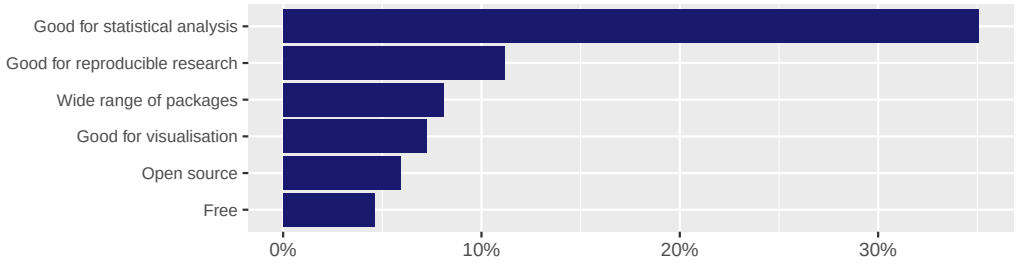


Figure 3: Number one argument for using R

These data seem to suggest that the respondents would recommend learning R because of it being a good tool for statistical analyses, followed by its use as a tool for reproducible research. Men and women respondents did not differ in their willingness to recommend to learn R, or in their arguments for or against R (note, however, that for the arguments the cell frequencies may have been too small for a valid analysis).

Further questions of the survey asked the respondents to indicate to what extent they agreed with certain statements about R. Figure 4 summarizes the responses to the statements that

1. Writing R is fun.
2. Writing R is considered cool or interesting by my peers.
3. Writing R is difficult.
4. Writing R is a monotonous task.

Percentages are in reference to the number of all given answers to the respective question.

We did not find any gender differences in these questions. Attendees of the useR! conference – unsurprisingly – regard R mostly as something fun and interesting and not very monotonous or difficult.

It seems interesting to note that a large part (160 respondents, 35.16%) of the sample reported to use R not only in a professional or educational setting, but also in their free time. When compared with men respondents, women respondents tend to use R less often in their free time, and more often in an educational or professional setting (see Figure 5). Percentages again add up to 100% for each gender respectively, while missing values are not considered.

In a chi square test, we found a medium effect size (Cohen's Omega = 0.25) for this gender difference.

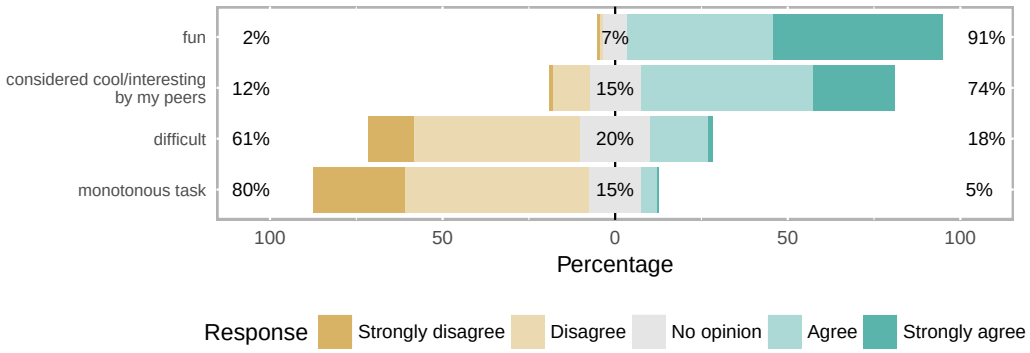


Figure 4: useR! 2016 attendees opinions on writing R.

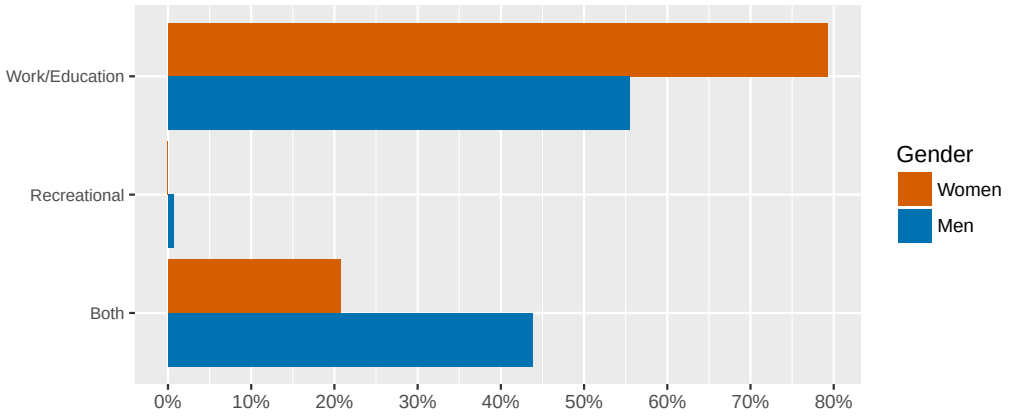


Figure 5: Primary purpose of using R.

Given that the higher share of R usage in their free time might also correspond to longer exposure times for men, which might again affect subjective or objective expertise as well as self-confidence for actively participating in activities like package writing, it might be worth investigating the factors behind this different usage behavior – be they motivational or due to structural differences like inequal distributions of household or childcare duties – in future research.

Experience with R

A significant part of the survey concerned the respondents' experience of working with R. Generally, the respondents tended to be rather experienced R users. 369 respondents (81.10%) reported that they had already worked with R for 2 years or longer, with 338 respondents (74.29%) stating that they had already had programming experience before working with R.

Since women respondents tended to be younger, it could be expected that they would also have shorter experience in working with R. As can be seen in Figure 6, our analysis shows that this is indeed the case (Omega = 0.19). Again, percentages add up to 100 per gender.

There were also gender differences when the respondents were asked about their previous programming experience before using R (Omega = 0.18). While 82.25% of the men who answered this question reported to have previous programming experience, the corresponding percentage among women was 66.06%.

A related question concerned whether the respondents use only existing functions of R or whether they also write and publish their own functions. A majority of the respondents (389, 85.49%) reported to have written R functions for their own use. A smaller part of the sample (253, 55.60% of the respondents who answered) reported to have written their own R package or have contributed to an R package. 155 respondents (33.07% of the respondents who answered) reported to have published their own R packages on CRAN. These results are further illustrated in Figure 7. Percentages in this plot are in reference to the respective gender again. They do not add up to 100% for each group though because multiple answers per person were possible. Notably, men were more predominant when it comes to R package development.

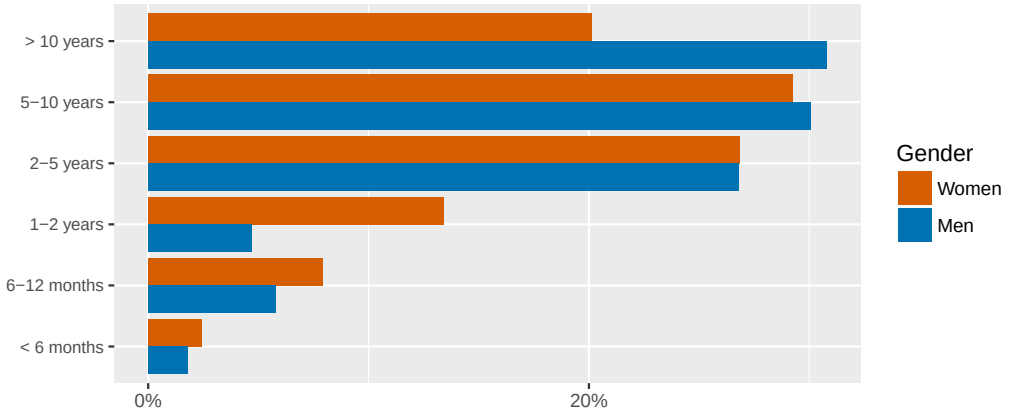


Figure 6: Length of R usage.

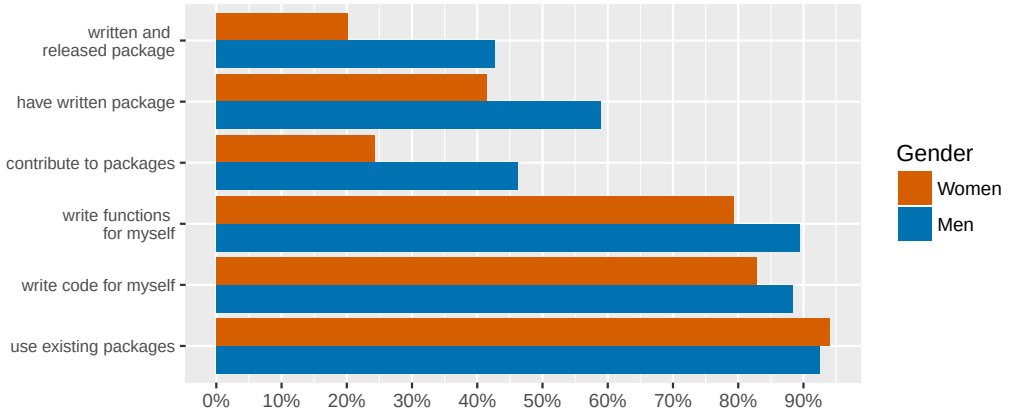


Figure 7: Usage types of R.

So far, our results indicate that women respondents tended to be younger and have used R for a shorter amount of time than men respondents. Furthermore, we found that women respondents have contributed to R packages less often. From these bivariate analyses, however, we cannot assess whether the gender difference in package development is confounded with the usage length and programming experience, or whether there are gender differences beyond these effects, that may again be confounded with the younger age of the women participants. Therefore, in the next section we conduct a multiple logistic regression model to assess the partial effects of gender and the experience variables on package writing.

Modeling gender differences in contribution to R packages

As was outlined in the previous sections, women respondents were less likely to contribute to R packages, but also tended to have less programming experience and a shorter length of R usage. The observed gender differences in the contribution to R packages could be confounded with these variables.

In order to assess the partial effects of these and further variables, we employed a logistic regression model. It should be clearly stated that this was a fully exploratory analysis, as no clear hypotheses about the factors affecting R package contributions in the general R community were available to guide the design of the survey or this statistical anlysis. As will be outlined in the following, we explored the association between package contributions and those survey variables that seemed plausible.

In the logistic regression models, we predicted whether someone had contributed to an R package in any form (i.e., the categories "contribute to packages", "have written package" and "written and released package" from Figure 7 were combined to form response category 1). Different models were compared, with *contribution to R packages* as outcome variable and *gender*, *length of R usage* and *previous programming experience* as a first set of candidate predictors. *Length of R usage* was coded as an ordered

factor variable that consists of 5 response categories that correspond to less than 1 year, 1-2 years, 2-5 years, 5-10 years, and 10 or more years. *Previous programming experience* was coded as a dichotomous variable indicating whether someone had previous programming experience before using R or not.

The results were inconclusive as to whether gender differences remain after accounting for the differences in length of R usage. With respect to AIC, the model with *length of R usage* and *gender* as predictors was the best model, whereas the BIC and the Likelihood ratio test preferred the model with only *length of R usage* as a predictor. The variable *previous programming experience* had no additional effect and was excluded by all criteria. From this first logistic regression analysis, it looked like a large part – but maybe not all – of the gender differences in the contribution to R packages were caused by differences in the length of R usage (with women showing shorter usage lengths, as displayed in Figure 6).

To explore potential effects of additional survey variables, we also included *employment status*, *purpose of R usage* and *community* in the analysis. *Employment status* was coded as a factor variable with eight different categories like in Figure 2. *Purpose of R usage* is the factor variable from Figure 5, and *community* is a binary variable indicating whether someone stated to feel as part of the R community or not (with descriptive statistics for this variable presented in the next section).

The logistic regression model with *gender*, *length of R usage*, *employment status* and *community* had the lowest AIC and was also the best model according to the Likelihood ratio test. The BIC again preferred the more sparse model without *gender*. *Purpose of R usage* did not improve model fit for any of the criteria.

With respect to the interpretation of the effects of the additional variables included in this model, for *employment status* we find that people working permanently and non-permanently in academia contribute most to R packages, whereas those working in government/non-profit and industry are slightly less likely to contribute to R packages. Feeling as part of the R *community* goes along with contributing more to R packages. Of course, the direction of this association may also be the other way round, since people who have already contributed to R packages are more likely to feel as part of the R community.

Again, the analysis does not give a clear answer to the question whether any gender differences in package contributions remain beyond the differences already captured by the other predictor variables. Any remaining differences might depend on a variety of other individual and structural factors, such as differences in motivation or self-confidence, in access to information, or in networking and peer support for contributing to R packages. After this first exploratory study, it would be very interesting to further question women R users that are on the verge of becoming R developers what might be keeping them back – as well as asking men R users that did cross over and become developers what helped them take that step.

The respondents as part of the R community

A final set of questions in the survey concerned the role of the respondents as part of the R community. Asked whether they considered themselves to be part of the R community, 361 (79.34%) respondents agreed, whereas 69 (15.16%) respondents disagreed and 28 (6.15%) respondents did not answer. Men and women respondents did feel as part of the R community to a comparable extent.

The respondents further reported to use a variety of resources to support their work with R. The respective question in the survey provided a list of possible resources, and also allowed the respondents to enter additional resources that were not listed as free text. Among the listed resources, StackOverflow and the R mailing list were the most frequently used resources. We found no gender differences here. Reporting only given answers that were chosen more than 4 times for brevity, the results are displayed in Figure 8.

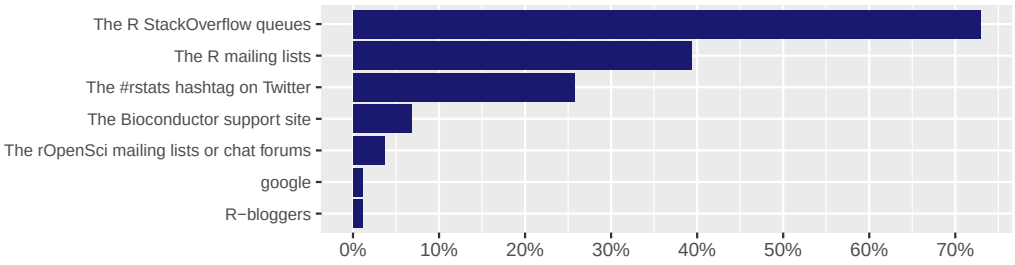


Figure 8: Resources men and women use to support their work with R.

Further asked about their preferred medium for R community news, the respondents most frequently chose the website (24.18%) and the mailing lists (21.10%). Other selected options were blog (18.02%), Twitter (15.16%) and Facebook (4.62%). Given these preferences, communication across a range of media is necessary to be confident in reaching a large proportion of R users and developers. A sizeable proportion (7.91%) did not select any option, so does not have a preference or is not interested in R community news. Nonetheless, as conference participants, they may receive some news in person.

We further investigated whether men and women respondents differed with regard to their preference of individual resources. Therefore, we tested for every single response category of the previous question whether the relative frequency with which it was selected differed between men and women respondents. When correcting for multiple testing, none of them was statistically significant.

Among the 455 respondents of the survey, 163 (35.82%) respondents reported that they attend meetings of a R user group, whereas 268 (58.90%) responded that they did not, and 27 (5.93%) respondents did not reply. Among the 163 respondents attending R user group meetings, 134 (82.21%) respondents indicated that they attended a general user group, whereas 16 (9.82%) respondents reported that they visited a user group within a university. Other types of user groups were less often named.

The respondents who did not visit R user group meetings were further asked about their reasons for this decision. Again, the respondents could answer this question by either selecting statements from a list or by entering new statements. The arguments that were chosen most frequently were that the respondents were too busy or that no active user group was available.

Finally, we investigated whether men and women respondents differed with respect to their attendance of R user group meetings, and how these meetings could be made more attractive for new attendees. In a first step, we found gender differences with regard to the reported attendance of R user group meetings. Men respondents reported to attend R user group meetings more often than women respondents with a small to medium effect (Omega = 0.21).

In a second step, we investigated gender differences to the question what measure would make the respondent more likely to attend user group meetings. For the individual response options, the following differences were found (only those answers are listed that were chosen by at least 10 respondents, with n indicating the total number of respondents that chose the respective option):

- New R user group near me (n = 122): no gender differences (Omega = 0.02)
- New R user group near me aimed at my demographic (n = 31): More positive responses by women respondents with a small effect (Omega = 0.19)
- Free local introductory R workshops (n = 61): no gender differences (Omega = 0.02)
- Paid local advanced R workshops (n = 61): no gender differences (Omega = 0.08)
- R workshop at conference in my domain (n = 73): no gender differences (Omega = 0.08)
- R workshop aimed at my demographic (n = 20): More positive responses by women respondents with a small effect (Omega = 0.12)
- Mentoring (e.g. first CRAN submission/useR! abstract submission/GitHub contribution) (n = 87): no gender differences (Omega = 0.09)
- Online forum to discuss R-related issues (n = 62): More positive responses by women respondents with a small effect (Omega = 0.14)
- Online support group for my demographic (n = 18): More positive responses by women respondents with a small effect (Omega = 0.15)

These answers indicate that most people who do not attend user group meetings do not have a user group near them or (as was suggested by free text answers) do not know that there are user groups. Other measures that could help to make people attend user group meetings would be R workshops at conferences in their domain, an online forum or local R workshops of different kinds. When it comes to gender differences, women might be more attracted by user groups explicitly aiming at women, but might also hint in the direction of women being less willing or less able or both to spend their free time with R, as we have seen in an earlier question.

Discussion

Our results draw a complex picture of the sample of attendees who agreed to respond to our survey. When looking at the sample as a whole, the respondents to this survey tended to have programming experience prior to working with R, and usually used R for 2 or more years. Most respondents further wrote their own R code, either to create functions for their own work or to contribute to R packages. Moreover, the average respondent had a positive opinion towards working with R. These results are

not very surprising, given that this sample was collected among the attendants of a useR! conference. However, as no comparable results on the community of R users have been published so far, these findings are nevertheless valuable. Future studies could use the results reported here as a benchmark for the evaluation of long-term developments in the R community.

In accordance with the initial question of the taks force, we did find that women were under-represented at the useR! conference 2016 – but even more strikingly, that non-white users were even more severely under-represented.

When looking at gender differences in more detail, results from three areas of the questionnaire stand out, that may however be confounded as discussed above. First, women respondents tended to be younger and have shorter experience in R usage than men respondents. Second, women used R less in their free time and contributed less to R packages. Third, women agreed less to feeling part of the R community.

The results of the exploratory logistic regression analysis suggest that the survey questions captured some factors associated with gender differences in contributions to R packages, such as the length of R usage, employment in academia and a feeling of belonging to the R community, that were positively associated with contributing to R packages. Yet the results were not conclusive as to whether there may be further gender differences, for example due to personal or structural factors, that may discourage women from writing R packages and would be important to investigate.

An additional hierarchical cluster analysis of the same data found three different groups of people: The first group (around 38% of the sample) are experienced R users, who tend to be men, from academia and people with doctorate. They use R not only in a professional setting, but also for recreational purposes. The second group (around 57% of the sample) are intermediately experienced users who use R for less than 2 years and mainly apply existing packages. They tend to be female, and are either undergraduates or have a master degree. They are using R mainly in professional or educational settings. The last group (around 3% of the sample) are the least experienced users who are using R during their free time. Details on this analysis as well as further multivariate analyses of the data can be found at UseR! 2016 R community: a multivariate analysis and UseR! 2016 participants: a multivariate analysis.

An anonymised form of the survey data, which minimises disclosure risk by excluding some demographic variables and recoding others, is available on CRAN (**forwards**). This data set includes aggregated versions of all demographic variables used in the logistic regression analysis and the majority of demographic variables used to aid interpretation in the multivariate analysis. Apart from the suppression of a small number of data values and a few free text variables that contained sensitive/identifying information, the responses to the programming and community questions are provided as recorded.

As stated already in the beginning of this text, due to the limitations of the study design, the results from the conference sample might not generalize to other subgroups of the R community, in particular not to those individuals who could not attend the conference due to factors associated with their being part of an under-represented group. Further studies are necessary to try to obtain a better picture of the R community as a whole.

Moreover, several topics which could be of further interest for the support of R users had not yet been included in the survey in order to keep it as short as possible. Further potentially interesting questions include, besides the ones already mentioned above, for example, what programming languages R users had already used before they started working with R, whether being in contact with other users that contribute to R packages makes it more likely to contribute to R packages oneself, etc.

Still the answers from this questionnaire give some indication to measures that could be taken to advance the participation of women, for example that user groups and other means of exchange explicitly targeting women might make them more accessible. Even though this would be methodologically challenging both due to the unspecific nature of the R community and the confounding of any interventions with different developments over time, it would be important to evaluate the development of under-represented groups over time.

The fact that women useR! attendees are on average younger and have lower R usage lengths might stimulate the hope that, once they reach the age and experience of their men counterparts, any gender differences might disappear automatically. However, this may not be the case, since multivariate analyses (see for instance Josse and Turner (2017)) tend to suggest that after adjusting for the age, women are still less involved in the R community. We still expect that if the R community and the opportunities to contribute are not equally attractive for women, they will not have the motivation to develop the skills to become contributors.

Although our initial framing looked specifically at gender as an axis of exclusion, the results show that race, not gender, is the area where there is the greatest disparity, and it is vital that inclusivity efforts factor this in. While the survey results give some ideas for improving inclusion in general,

specific efforts should be made to reach out to people from under-represented races, for example through diversity scholarships, invited conference contributions, or invitations to serve in community roles. Further information on who is under-represented, and for what reason, would support such efforts.

Bibliography

K. B. Coffman, L. C. Coffman, and K. M. M. Ericson. The size of the LGBT population and the magnitude of anti-Gay sentiment are substantially underestimated. Working Paper 19508, National Bureau of Economic Research, October 2013. URL http://www.nber.org/papers/w19508. [p543]

J. Cohen. *Statistical power analysis for the behavioral sciences Lawrence Earlbaum Associates*. Lawrence Erlbaum Associates, Hillsdale, NJ, 2 edition, 1988. [p543]

J. Fox. Aspects of the social organization and trajectory of the R project. *The R Journal*, 1(2):5–13, 2009. [p542]

G. J. Gates. How many people are lesbian, gay, bisexual, and transgender?, Apr. 2011. URL http://williamsinstitute.law.ucla.edu/wp-content/uploads/Gates-How-Many-People-LGBT-Apr-2011.pdf. [p543]

J. Josse and H. Turner. useR! 2016 participants and the R community: a multivariate analysis, June 2017. URL https://forwards.github.io/docs/MCA/community/useR2016/survey/. [p550]

P. Mair, E. Hofmann, K. Gruber, R. Hatzinger, A. Zeileis, and K. Hornik. Motivation, values, and work design as drivers of participation in the R open source project for statistical computing. *Proceedings of the National Academy of Sciences of the United States of America*, 112(48):14788–14792, 2015. URL https://doi.org/10.1073/pnas.1506047112. [p542]

Stella Bollmann
Department of Psychology
University of Zurich
Zurich
Switzerland
s.bollmann@psychologie.uzh.ch

Dianne Cook
Department of Econometrics and Business Statistics
Monash University
Melbourne
Australia
dicook@monash.edu

Jasmine Dumas
Simple Finance
Hartford
United States of America
jasmine.dumas@gmail.com

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

Julie Josse
Department of Applied Mathematics
Ecole Polytechnique, University of Paris-Saclay
Paris
France
julie.josse@polytechnique.edu

Oliver Keyes
Department of Human Centred Design & Engineering
University of Washington
Seattle
United States of America
okeyes@uw.edu

Carolin Strobl
Department of Psychology
University of Zurich
Zurich
Switzerland
carolin.strobl@psychologie.uzh.ch

Heather Turner
Department of Statistics
University of Warwick
Coventry
United Kingdom
ORCiD: 0000-0002-1256-3375
ht@heatherturner.net

Rudolf Debelak
Department of Psychology
University of Zurich
Zurich
Switzerland
rudolf.debelak@psychologie.uzh.ch

R Teaching Column

by Matthias Gehrke, Reed Davis, Norman Matloff, Paul Thompson, Tiffany Chen, Emily Watkins and Laurel Beckett

Invitation to collaborate

With this issue of the R Journal, there will be a section for teaching R and teaching using R as well as for empirical research on teaching R.

Teaching material

A GitHub repository will be setup where the teaching material can be accessed. All material should be under some licence of Creative Commons to allow re-use. Please prepare a short text describing your material. For a sample, see the description by Norman Matloff et al. (below) of class room material he is using.

Research on teaching R

Some research is considering the relative success of different approaches to teaching R and teaching statistics with R. For example, see some contributions on the recent useR!2017 in Brussels (Gert Janssenwill et al., The analysis of R learning styles with R, Matthias Gehrke and Karsten Luebke, A quasi-experiment for the influence of the user interface on the acceptance of R). The R Journal is interested in receiving such submissions.

Technical details

This column will appear before this GitHub repository is established, so only links to an existing cooperating repository and mailing list can be provided at this time. Subsequent columns will provide full technical details or links to such details.

Matthias Gehrke
FOM University of Applied Sciences
Franklinstr. 52, 60486 Frankfurt a. M.
Germany
matthias.gehrke@fom.de

The 'revisit' Package As a Teaching Tool

Abstract The **revisit** package, developed as a collaborative tool for scientists, also serves as a tool for teaching statistics, in a manner that can be highly motivating for students. Using either the included case studies or datasets/code provided by the instructor, students can explore several alternate paths of analysis, such as the effects of including/excluding certain variables, employing different types of statistical methodology and so on. The package includes features that help students follow modern statistical standards and avoid various statistical errors, such as "p-hacking" and lack of attention to outlier data.

Overview

The R package **revisit** was developed in response to the recent concern over *reproducibility* in research, especially problems related to statistical analysis (Baker, 2016). The motivation and methods are detailed in Matloff et al. (2017), but in this paper we turn to teaching.

The package has both text and GUI versions, the latter being based on the RStudio integrated development environment for R. It is currently at an early stage of development, with more features and refinements being added continuously. It may be downloaded from `https://github.com/matloff/revisit`.

One can obtain a quick introduction to the package by starting it and running one of the case studies. To start the package in GUI form, click Addins in RStudio, and choose "Revisit". The text version is not colorful, but is more flexible. To run it, simply run `library(revisit)`. Use of the case studies will be introduced later in this paper.

An excellent motivator: the Reinhart/Rogoff study controversy

The package includes a number of examples for student examination and participation. One of the case studies involves the controversial paper by Harvard economists C. Reinhart and K. Rogoff (Cassidy, 2016). They had found that nations with high budget deficits average a -0.1% growth rate in GDP. This finding had major impact on policymakers, with the paper attracting particular interest from the "deficit hawks" in the U.S. Congress. The *Washington Post* took to describing the finding as "consensus among economists."

But when later researchers tried alternate analysis, the picture changed entirely. Some data had been excluded from the initial published analysis, for what arguably were weak reasons. The original analysis also had the flaw of giving equal weights to all nations, regardless of size, as well as other aspects that some researchers considered flaws. When the alternate analysis was run, the figure -0.1% changed to +2.2%. Thus the original findings on deficit spending now seem questionable. This leads to the theme in the present paper, use of **revisit** for teaching.

At a talk given by one of the authors of the present paper (Matloff, 2017), the author was stunned at student reaction to the Reinhart/Rogoff example. The students, all doing graduate work in science and engineering, were captivated by the fact that the study, which had had such influence on policy, may have been seriously flawed, with alternate analyses of the same data yielding starkly different results. The potential of **revisit** as a teaching tool had been suggested earlier by one of the other authors of the present paper, but the strong student reaction here dramatized the point.

General structure of the package

The package is structured as follows:

- A set of case studies for students to explore and modify (and to which instructors can add their own examples).
- "Statistical audit" warnings/advice given to students regarding statistical best practices as they proceed in their analyses.
- A code management infrastructure to facilitate exploration of alternative analyses.

The case study approach

The package includes various case studies, consisting of data and code. The latter comprises a complete, though possibly brief, analysis of the data from start to finish — the code may include data configuration, data cleaning, preliminary graphical analysis, predictor variable/model selection,

and so on. Students can then try their own alternative analyses. Most of the case studies are not as dramatic as Reinhart and Rogoff, but each illustrates important concepts in data analysis.

There are just a few case studies included in the package currently, but more are being added, and instructors will often prefer to use their own data anyway. In many cases, the code will be minimal, affording the students even more opportunity for nonpassive thought and exploration.

Assignments centered on the use of **revisit** can be very specific or more open-ended, according to the instructor's preference. Here are possible samples:

- "This assignment involves the Reinhart and Rogoff data, included in the **revisit** package. Revisit the analysis with different weightings for the nations based on various factors."
- "This assignment involves the famous Pima diabetes study. Some later inspection suggested that the data includes a number of erroneous values, such as impossible 0 values. Investigate this, say using graphical means. Run a logistic regression model, with and without the suspect data points, and compare the results. Also, the authors of the study used a neural network approach. Try using random forests, and compare to the authors' level of predictive ability."
- "This assignment analyzes the famous Forest Cover dataset, in which one of seven cover types is predicted from variables such as Hillside Shade at Noon. The given code tries prediction using, of course, the random forests method. The first 10 predictors are used. In your alternate analysis, try using all 54 predictors. That is quite a bit, and even with over 500,000 observations, one must always worry about overfitting; investigate that here."
- "This assignment analyzes the Fong-Ouliaria currency data. The supplied code fits a linear regression model, with a respectable R^2 value. However, one can do better. First, explore this with some of the graphical methods in the **regtools** package, and then try adding some quadratic and interaction terms to the model, and/or try a nonparametric regression approach."
- "Here you will work with data from the 2000 Census, involving salaries of programmers and engineers in Silicon Valley. One aspect of interest is the gender issue — are women paid less than comparable men? A preliminary regression analysis indicates a difference of over $10,000, for fixed age and educational level. But much more needs to be done. For instance, what happens when occupation is factored in, and when the non-monotonic relation of wage and age is accounted for? Investigate such questions."

We are continuing to add case studies. At this early stage, the list includes:

- UCI Pima diabetes data (Lichman, 2017)
- Zavodny guestworker data (Zavodny, 2011)
- Reinhart and Rogoff (Cassidy, 2016)
- MovieLens (Harper and Konstan, 2015)
- Fong-Ouliaria currency data (Fong and Ouliaris, 1995)
- UCI forest cover data (Lichman, 2017)
- Salary data on programmers and engineers, Silicon Valley, 2000 Census

We anticipate that instructors will typically develop their own case studies. We encourage them to contribute these to **revisit**.

To load a case study in the GUI, just select the desired one from the Case Studies window. The code will then be loaded, ready to run. For the text version, at present this is done as in this example, for the Currency data:

```
fname <- system.file('CaseStudies/Currency/currency.R',package='revisit')
loadb(fname)
```

A "statistical audit"

One of the most important features of **revisit** is that it plays the role of a "statistical audit," in much the same way that tax preparation software might warn of questionable claims in a tax return.

Much of this is accomplished by wrapper functions provided by **revisit**. For instance, if the student wishes to form a confidence interval for a mean in R, she might use `t.test()`. But in **revisit** she can instead use `t.test.rv()`. Instead of **lm()** she can call `lm.rv()`, a wrapper for `lm()` that adds "statistical audit" functionality.

The wrapper lm.rv()

As noted, this function calls `lm()` and returns an object of class `"lm"`, but with additional functionality. At present this takes on two forms:

- It checks whether the response variable takes on only two values, in which case the function suggests a logistic model using **glm()**:

```
> y <- c(1,0,1)
> x <- 1:3
> d <- data.frame(y,x)
> lm.rv(y ~ x,d)
...
Coefficients:
(Intercept)            x
     0.6667       0.0000

Warning message:
In lm.rv(y ~ x, user.data = d) :
  only 2 distinct Y values; consider a logistic model
```

- It runs a parallel analysis, i.e. with the same formula and data as the `lm()` call, but with median (Minimum Absolute Deviation) regression, implemented with `rq()` from the **quantreg** package. For instance, with the salary case study, one might run:

```
> lm.rv(wageinc ~ age+sex+ms+phd,user.data=pe)
max. prop. difference, linear median regression: 0.3221592
larger values, may indicate outlier or model fit issues

Call:
lm(formula = formula, data = user.data)

Coefficients:
(Intercept)          age          sex           ms          phd
    53286.4        441.4     -12343.6      18363.6      27770.3
```

  There is more than a 32% difference in the two model fits, which turns out to be in the age coefficient. (The `rq()` coefficients are available as a component `$rqc` of the `"lm"` object returned by `lm.rv()`.) As noted, this may indicate issues with outliers or model fit.

"Audits" of signficance testing

The package aims to reduce p-hacking by monitoring the number of inference actions — p-values, confidence intervals — the student has accumulated in his/her analysis, and may issue a warning that the student should consider employing multiple-inference methods. For the latter, at present the package offers just Bonferroni's Method, but more will be added (Hsu, 1996).

Moreover, the package responds to the dramatic 2016 announcement by the American Statistical Assocation (Wasserstein and Lazar, 2016), which warned on the overuse of p-values. Though this problem had been common knowledge for many years (Ziliak and McCloskey, 2008; Freedman et al., 1978; Jones and Matloff, 1986), the ASA announcement gave new urgency to the issue. The **revisit** package takes an active role in encouraging students not to rely much on p-values in the first place. Confidence interval-based analysis is preferred.

Here is the code for `t.test.rv()`:

```
> t.test.rv
function (x, y, alpha = 0.05, bonf = 1)
{
    alpha <- alpha/bonf
    tout <- t.test(x, y, conf.level = 1 - alpha)
    muhat1 <- tout$estimate[1]
    muhat2 <- tout$estimate[2]
    tout$p.value <- tout$p.value * bonf
    rvenv$pcount <<- rvenv$pcount + 1
    if (tout$p.value < alpha && muhat1 != 0) {
        if (abs(muhat1 - muhat2)/abs(muhat1) < rvenv$smalleffect)
```

```
            warning(paste("small p-value but effect size",
            "could be of little practical interest"))
    }
    tout
}
```

Here the argument `bonf` is a multiplicative factor used to expand confidence interval widths for Bonferroni corrections, as seen in the above code.

Note the incrementing of `rvenv$pcount`. This is the global count (which includes potential confidence intervals) alluded to earlier, to be used in the warning that the user should consider multiple inference methods.[1] Note too that the code may warn the student that there was a "small p-value but effect size could be of little practical interest."

Other wrappers

Steering students (and their instructors) to confidence intervals instead of p-values can be difficult not only in terms of breaking habits, but also in technical terms. Consider the log-linear model. for instance. Most packages perform log-lin solely from a hypothesis testing point of view. R's `stats::loglin()` function, for instance, will not provide standard errors, and only provides point estimates on request. Our package will go further, offering point estimates and standard errors for estimated cell probabilities. As the user steps through the model hierarchy, at a certain point it will become clear that the estimates are not changing in important ways, and one can stop the model selection process. This will be accomplished by the "Poisson trick"(Christensen, 2013), in conjunction with R's `glm()` function.

Generating and managing alternative analyses

After a student selects an example from the Case Study menu, the package loads the desired code and data, and enters the code into the package's visual text editor. The student can now edit and run the revised code, *including just a partial run, up to a given line*.

The latter capability is especially useful. Say the code consists of 32 lines, and line 11 is of interest to the student. The student can direct **revisit** to run lines 1-10 of the code, then pause. At that point the student can try executing an alternative to line 11, by executing the alternative line in the Console box, which provides direct access to the RStudio R console in which execution takes place. The student can then resume execution of the code starting from line 12. This allows the student to quickly try alternative code without actually changing the contents of the text editor.

In generating various alternative analyses, the student can save the interesting ones, each version in a different file, but all managed conveniently by **revisit**. Borrowing from software engineering terminology, each of these files is called a *branch*.

Example: Pima diabetes study

Let's start with a simple and quite well-known example, the Pima diabetes data. The authors (Baker, 2016) used a form of neural networks to predict diabetes from variables such as BMI and insulin. To keep things simple, though, we will not engage in prediction analysis here.

Upon launching **revisit** in the GUI, a new window associated with the RStudio session then pops up, and the screen then looks like Figure 1. Pima is the first case study listed.

The included code here consists of just forming confidence intervals comparing diabetic and nondiabetic groups, on each of the eight predictor variables. The student can choose to run the entire code or just a part; say she chooses the former. The confidence intervals can then be seen in the R console portion of the RStudio window (Figure 2).

At this point, the student may ask, "What if we make the Bonferroni correction here?" Although she could edit line 15 in the **revisit** text editor window to

```
tmp <- t.test.rv(diab[ ,i], nondiab[ ,i])$conf.int
```

[1] Technically switching to multiple inference methods midstream like this does not fully solve the problem, as the probabilistic behavior of the analysis now is conditional on having made the switch. However, this is a general problem in statistical analysis, far beyond the scope of **revisit**. For instance, typically an analyst will perform graphical analysis of the data before embarking on formal inference analysis, again resulting in a conditional setting.

and then re-run, if this is just a tentative change, she could avoid changing the code, by entering the above into the Console box, as seen in Figure 3. As expected, the confidence intervals become wider (not shown). If the student wishes to make further changes, she may now wish to make the above change in the text editor, and possibly save the new code into a new branch.

After the Pima dataset was curated, there were reports of erroneous values in some data points. To investigate this, one might run the `discparcoord()` function from the package **cdparcoord**, included in **revisit**:

```
discparcoord(pima,k=769)
```

Here the second argument specifies forming the graph on all 769 records in the data.[2]

The resulting graph will be displayed in the Viewer pane in the RStudio window. In the text version, the student would run the above directly, in which case the graph appears in the student's Web browser.

The result is shown in Figure 4. This is a *parallel coordinates* plot (Inselberg, 2009). Each data point is displayed as a segmented line connecting dots at heights given by the values of the variables in that data point.[3]

The results are rather striking. We see that there are data points having the impossible value of 0 for variables such as glucose and blood pressure. Indeed, there are some data points with multiple 0s. Clearly the student will need to remove some of the data points, and re-run the analyses.

The student may wish to create multiple versions of the code, e.g. in this case, code with and without the erroneous points. The **revisit** package facilitates this, creating branches 0, 1, 2 and so on of the code.

Future development

As mentioned, a number of wrapper functions are planned, as well as expanding the "statistical audit" features of existing wrappers. Much more graphical analyis is slated.

The number of case studies will continue to grow.

Conclusions

The **revisit** package facilitates nonpassive, hands-on exploration of statistical and data analytic methodology on real data. The students learn that even published data is not sacrosanct, and that alternate analysis can yield additional insight into the phenomena under study.

Statistical methodology pervades almost every conceivable aspect of our world today. In addition to the students' possible future usage of statistics in professional roles, educators should prepare them to act as informed, critical thinkers in their roles as citizens and consumers. We hope that **revisit** can play a part in achieving such goals.

It is also hoped that instructors and students will contribute suggestions for improvement, including pull requests on GitHub. These, and of course new datasets, would be highly appreciated.

Acknowledgements

We are grateful to Bohdan Khomtchouk for inviting one of the authors to speak on **revisit** at the inaugural meeting of the Stanford R Group, and to Karen Wu for a careful reading of the manuscript.

Bibliography

M. Baker. 1,500 scientists lift the lid on reproducibility. *Nature*, 533(7604):452–454, May 2016. URL https://doi.org/10.1038/533452a. [p554, 557]

J. Cassidy. The Reinhart and Rogoff Controversy: a Summing Up. *The New Yorker*, 533, Apr. 2016. URL https://doi.org/10.1038/533452a. [p554, 555]

[2]The **cdparcoord** package generally displays only the most frequently-appearing tuples, rather than the full dataset of all tuples.

[3]Various functions for parallel coordinates plotting are available in both base R and CRAN. As noted, what sets these apart from **cdparcoord** and its "cousin" **freqparcoord** is that the latter two display only frequently-appearing points. This is aimed at avoiding the "black screen problem," which occurs when we have so many data points that their representations fill the screen, rendering the graph useless.

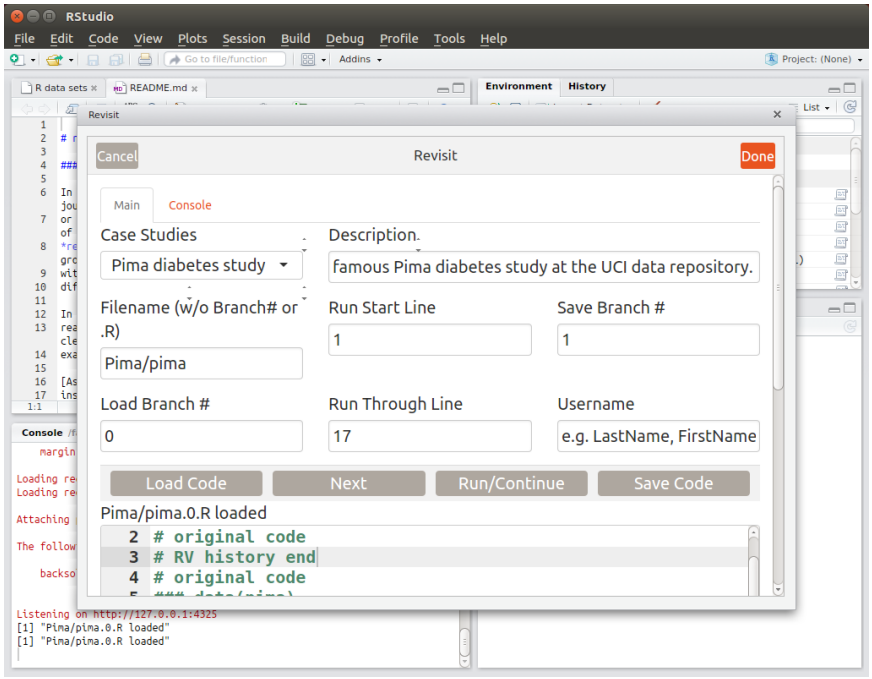


Figure 1: Opening screen.

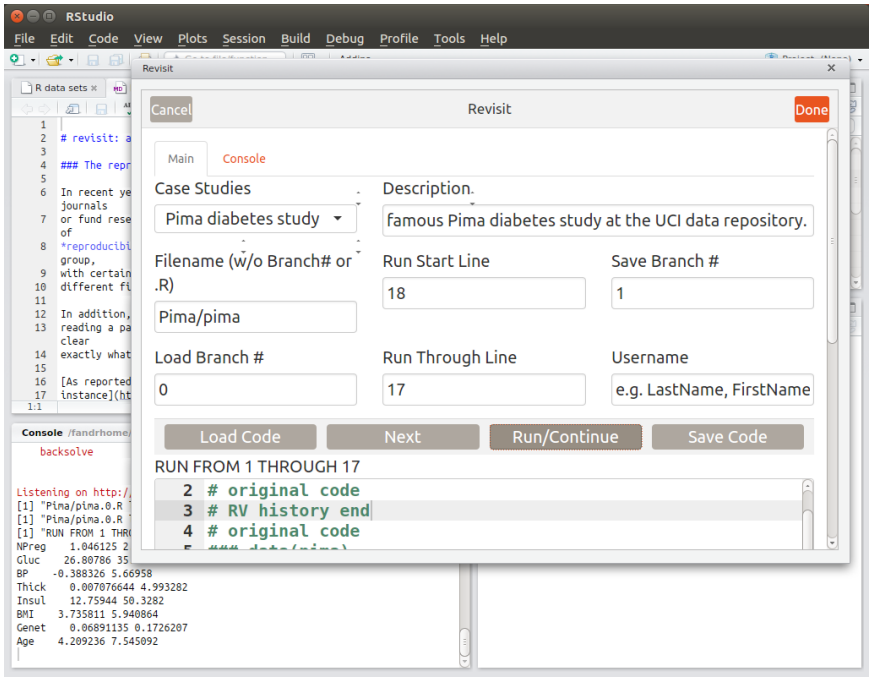


Figure 2: Ordinary CIs.

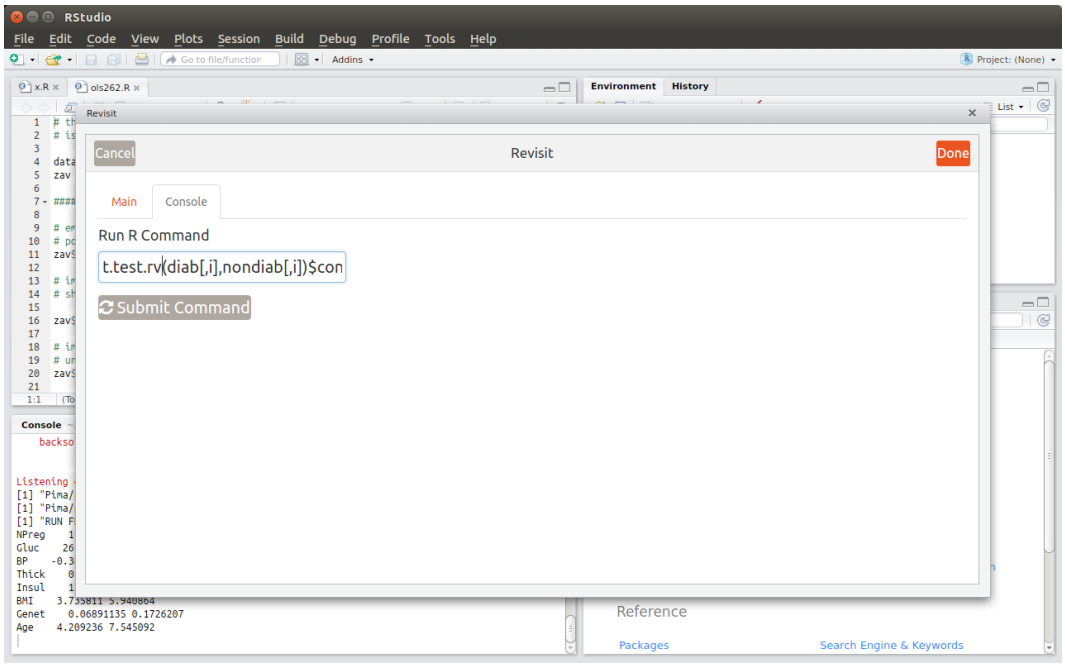


Figure 3: Console box.

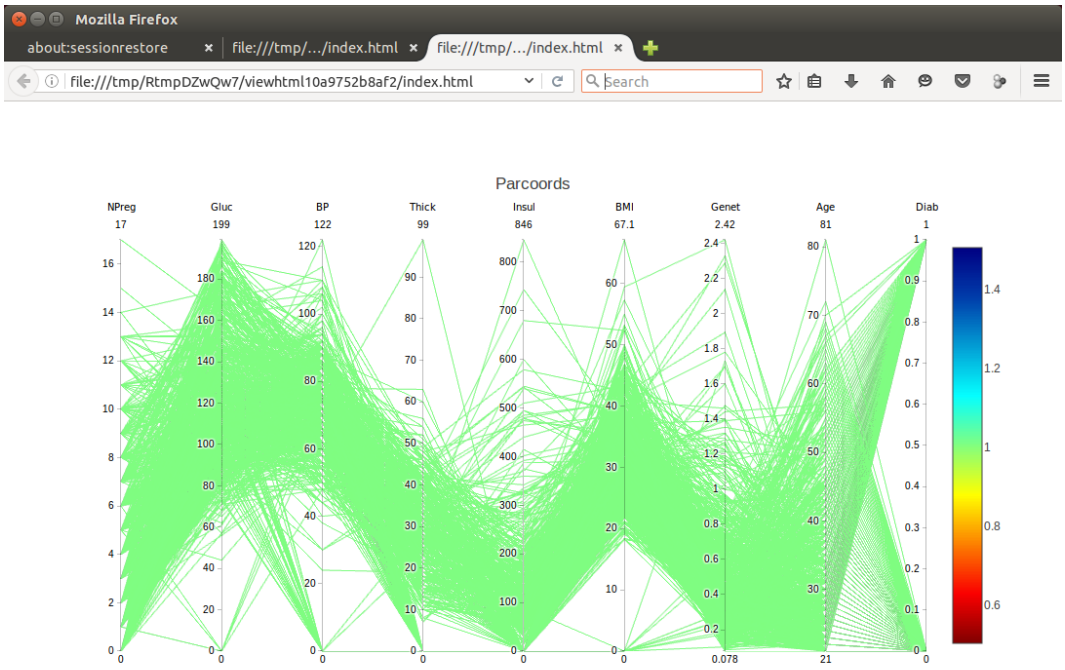


Figure 4: Parallel coordinates view.

R. Christensen. *Log-Linear Models*. Springer Texts in Statistics. Springer New York, 2013. [p557]

W. M. Fong and S. Ouliaris. Spectral tests of the martingale hypothesis for exchange rates. *Journal of Applied Econometrics*, 10(3):255–71, 1995. URL https://doi.org/10.1002/jae.3950100304. [p555]

D. Freedman, R. Pisani, and R. Purves. *Statistics*. Norton, 1978. [p556]

F. M. Harper and J. A. Konstan. The MovieLens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4):19:1–19:19, Dec. 2015. URL https://doi.org/10.1145/2827872. [p555]

J. Hsu. *Multiple Comparisons: Theory and Methods*. Taylor & Francis, 1996. [p556]

A. Inselberg. *Parallel Coordinates: Visual Multidimensional Geometry and Its Applications*. Advanced series in agricultural sciences. Springer New York, 2009. [p558]

D. Jones and N. Matloff. Statistical hypothesis testing in biology: a contradiction in terms. *J. of Economic Entomology*, 79(5):1156–1160, Oct. 1986. [p556]

M. Lichman. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml. [p555]

N. Matloff. 'revisit': an R Package for Taming the Reproducibility Problem (Stanford University R Users Group). public lecture, 2017. URL http://heather.cs.ucdavis.edu/StanfordR.pdf. [p554]

N. Matloff, R. Davis, L. Beckett, and P. Thompson. revisit: a workflow tool for data science, 2017. URL https://arxiv.org/abs/1708.04789. [p554]

R. L. Wasserstein and N. A. Lazar. The asa's statement on p-values: Context, process, and purpose. *The American Statistician*, 70(2):129–133, 2016. URL https://doi.org/10.1080/00031305.2016.1154108. [p556]

M. Zavodny. Immigration and american jobs, 2011. URL http://www.aei.org/wp-content/uploads/2011/12/-immigration-and-american-jobs_144002688962.pdf. [p555]

S. Ziliak and D. McCloskey. *The Cult of Statistical Significance: How the Standard Error Costs Us Jobs, Justice, and Lives*. University of Michigan Press, 2008. [p556]

Reed Davis
Sunnyvale, CA
USA
rdavis2468@gmail.com

Norman Matloff
University of California, Davis
Dept. of Computer Science
USA
matloff@cs.ucdavis.edu

Paul Thompson
Sanford Research
Sioux Falls, SD
USA
Paul.Thompson@sanfordhealth.org

Tiffany Chen
University of California, Davis
Dept. of Statistics
USA
techen@ucdavis.edu

Emily Watkins
University of California, Davis
Dept. of Statistics
USA
emwatkins@ucdavis.edu

Laurel Beckett
University of California, Davis
School of Medicine
USA
labeckett@ucdavis.edu

R Foundation News

by Torsten Hothorn

Donations and members

Membership fees and donations received between 2017-06-16 and 2017-12-01.

Donations

Yahaya Nazoumou (Niger) Jørgen Raffnsøe (Denmark) Romuald Riem (France) Renan Silverio (Brazil) Appstam Consulting GmbH, Berlin (Germany) Spängler IQAM Invest GmbH, Salzburg (Austria)

Supporting benefactors

Thomas Dangl (Austria) Transmitting Science, Barcelona (Spain)

Supporting members

Ayala S. Allon (Israel) Michael Blanks (United States) Gilberto Camara (Brazil) Jorge de la Vega (Mexico) Elliott Deal (United States) Kristina Dietz (United Kingdom) Charles Geyer (United States) Michael Griffiths (United States) Ken Ikeda (Japan) Artem Ilievskiy (Russia) JUNE KEE KIM (South Korea) Felix Kluxen (Germany) Sebastian Kreutzer (France) Pawel R. Kulawiak (Germany) Chel Hee Lee (Canada) Detlef Lehmann (Germany) Gorka Navarrete (Chile) Matt Parker (United States) Gerard Pennefather (Singapore) Alfonso Reyes (United States) Carlos Erwin Rodríguez Hernández Vela (Mexico) Joshua Rosenstein (United States) Fabian Scheipl (Germany) Surendra Singh (United Kingdom) Earo Wang (Australia) Dave Williams (United Kingdom) Kisung You (United States)

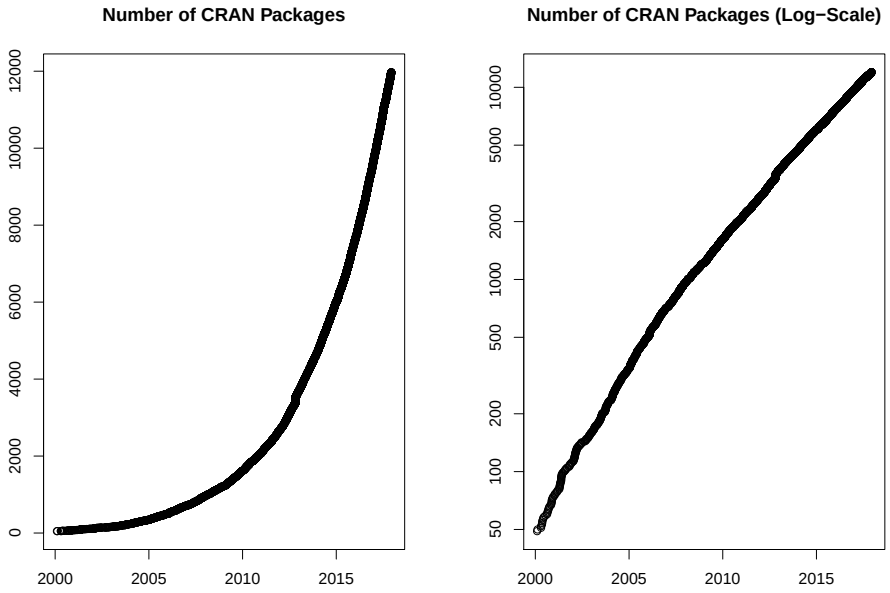
Torsten Hothorn
Universität Zürich, Switzerland Torsten.Hothorn@R-project.org

Changes on CRAN

2017-06-01 to 2017-11-30

by Kurt Hornik, Uwe Ligges and Achim Zeileis

In the past 6 months, 1244 new packages were added to the CRAN package repository. 19 packages were unarchived, 55 archived and 3 removed. The following shows the growth of the number of active packages in the CRAN package repository:



On 2017-11-30, the number of active packages was around 11875.

Changes in the CRAN checks

The package check pages now also show issues found by checks of corruption of constants (provided by Tomáš Kalibera).

Changes in the CRAN submission pipeline

Package maintainers who submitted packages this year found the automated submission system accepted or rejected some packages automatically while other packages went into a manual inspection queue. The number of false positives that led to wrong rejections has been reduced. Given the system is pretty stable now, we will go a step further and also auto-accept packages with reverse dependencies where the check status of all reverse dependencies checked is not worse than before. So far incoming checks in CRAN have been performed on a single platform (Linux or Windows) only. While (incoming) checks are improved all the time, we will shortly have both Linux and Windows systems analyzing packages before publishing automatically.

CRAN received 2087 package submissions in November 2017, i.e., around 70 submissions a day. Hence the CRAN team is no longer able to respond to individual help requests or be involved in lengthy discussions for exceptions. Please really use the corresponding mailing lists such as R-package-devel (see https://www.r-project.org/mail.html).

Changes in the CRAN Repository Policy

The Policy now says the following:

- CRAN packages should use only the public API. Hence they should not use entry points not declared as API in installed headers nor `.Internal()` nor `.Call()` etc. calls to base packages. Also, `:::` should not be used to access undocumented/internal objects in base packages (nor should other means of access be employed).
- Packages should not attempt to disable compiler diagnostics.
- All correspondence with CRAN must be sent to `CRAN-submissions@R-project.org` (not members of the team) and be in plain text ASCII (and not HTML).

In addition, the Policy now also points to a new Checklist for CRAN submissions.

CRAN mirror security

Currently, there are 95 official CRAN mirrors, 58 of which (about 61%) provide both secure downloads via '`https`' *and* use secure mirroring from the CRAN master (via rsync through ssh tunnels). Since the R 3.4.0 release, `chooseCRANmirror()` offers these mirrors in preference to the others which are not fully secured (yet).

Hyperlinks in package DESCRIPTION files on CRAN

For package authors specified via an '`Authors@R`' field in the DESCRIPTION file, ORCID identifiers (see `https://orcid.org/` for more information) can be provided via elements named '`ORCID`' in the `comment` argument of the `person()` calls, e.g., `person("Achim","Zeileis",comment = c(ORCID = "0000-0003-0918-3766"))`. These identifiers will then be hyperlinked in the CRAN package web pages to the corresponding ORCID pages. See, e.g., the page for package **ctv**.

Windows binaries

Starting with R 3.4.3, Jeroen Ooms maintains the Windows base R binaries and the toolchain for building both R and contributed packages on Windows.

New packages in CRAN task views

Bayesian **openEBGM**, **tRophicPosition**.

ClinicalTrials **InformativeCensoring**, **Mediana**, **ThreeArmedTrials**, **clusterPower**, **crmPack**, **dfped**, **dfpk**, **ewoc**, **gsbDesign**.

DifferentialEquations **QPot**, **cOde**, **dMod**, **phaseR**, **rODE**, **rodeo**, **rpgm**.

Distributions **MittagLeffleR**, **coga**, **hyper2**.

Econometrics **OrthoPanels**, **dlsem**, **pder**, **wooldridge**, **zTree**.

ExperimentalDesign **DoE.MIParray**, **FMC**, **MBHdesign**, **PBIBD**, **bioOED**, **edesign**, **idefix**, **minimalRSD**, **odr**, **optbdmaeAT**, **optrcdmaeAT**, **rsurface**, **sFFLHD**, **skpr***, **soptdmaeA**, **unrepx**.

ExtremeValue **POT**.

FunctionalData **covsep**, **denseFLMM**, **freqdom.fda**, **ftsspec**.

HighPerformanceComputing **Sim.DiffProc**, **drake**, **parSim**.

MachineLearning **ICEbox**, **effects**, **ggRandomForests**, **pdp**, **plotmo**, **tensorflow**.

MetaAnalysis **CIAAWconsensus**, **ConfoundedMeta**, **MetaSubtract**, **RandMeta**, **TFisher**, **clubSandwich**, **effsize**, **forestmodel**, **getmstatistic**, **metaBMA**, **metacart**, **metaforest**, **nmaINLA**, **psychmeta**, **ratesci**, **rma.exact**.

NaturalLanguageProcessing **alineR**, **ore**, **rel**, **stm**, **stringdist**.

NumericalMathematics **PythonInR**, **SnakeCharmR**, **XR**, **XRJulia**, **XRPython**, **expint**, **feather**, **findpython**, **fourierin**, **interp**, **logOfGamma**, **reticulate**, **tripack**.

Optimization **ABCoptim**, **CVXR**, **ManifoldOptim**, **Rtnmin**, **SACOBRA**, **colf**, **coneproj**, **ecr**, **flacco**, **metaheuristicOpt**, **mize**, **n1qn1**, **ompr**, **optimr**, **optimsimplex**, **quadprogXT**, **sdpt3r**.

Pharmacokinetics **RxODE**.

Phylogenetics **treeplyr**.

Psychometrics **CTTShiny**, **EFAutilities**, **MIIVsem**, **PLmixed**, **dexter**, **umx**.

Spatial **spm**, **spsann**.

SpatioTemporal **FLightR**, **sf**, **sigloc**.

TimeSeries **dLagM**, **fpp2**, **freqdom**, **freqdom.fda**, **ftsa**, **funtimes**, **influxdbr**, **odpc**, **sweep**, **timetk**, **tscount**, **wktmo**.

WebTechnologies **gtrendsR**.

Kurt Hornik
WU Wirtschaftsuniversität Wien, Austria
Kurt.Hornik@R-project.org

Uwe Ligges
TU Dortmund, Germany
Uwe.Ligges@R-project.org

Achim Zeileis
Universität Innsbruck, Austria
Achim.Zeileis@R-project.org

News from the Bioconductor Project

by Bioconductor Core Team

The Bioconductor project provides tools for the analysis and comprehension of high-throughput genomic data. Bioconductor 3.6 was released on 31 October, 2017. It is compatible with R 3.4.3 and consists of 1473 software packages, 326 experiment data packages, and 911 up-to-date annotation packages. The release announcement includes descriptions of 100 new software packages, and updated NEWS files for many additional packages. Start using Bioconductor by installing the most recent version of R and evaluating the commands

```
source("https://bioconductor.org/biocLite.R")
biocLite()
```

Install additional packages and dependencies, e.g., **BiocFileCache**, with

```
BiocInstaller::biocLite("BiocFileCache")
```

Docker and Amazon images provide a very effective on-ramp for power users to rapidly obtain access to standardized and scalable computing environments. Key resources include:

- The bioconductor.org web site to install, learn, use, and develop Bioconductor packages.
- A listing of available software, linking to pages describing each package.
- A question-and-answer style user support site and developer-oriented mailing list.
- The F1000Research Bioconductor channel for peer-reviewed Bioconductor work flows.
- Our package submission repository for open technical review of new packages.

Our annual conference, still in the planning stages, will be on July 25 ('Developer Day'), 26, and 27, in Toronto, Canada.

Bioconductor Core Team
Biostatistics and Bioinformatics
Roswell Park Cancer Institute, Buffalo, NY
USA maintainer@bioconductor.org

Changes in R

From version 3.4.2 to version 3.4.3

by R Core Team

CHANGES IN R 3.4.3

INSTALLATION on a UNIX-ALIKE

- A workaround has been added for the changes in location of time-zone files in macOS 10.13 'High Sierra' and again in 10.13.1, so the default time zone is deduced correctly from the system setting when R is configured with '`--with-internal-tzcode`' (the default on macOS).
- `R CMD javareconf` has been updated to recognize the use of a Java 9 SDK on macOS.

BUG FIXES

- `raw(0) & raw(0)` and `raw(0) | raw(0)` again return `raw(0)` (rather than `logical(0)`).
- `intToUtf8()` converts integers corresponding to surrogate code points to `NA` rather than invalid UTF-8, as well as values larger than the current Unicode maximum of `0x10FFFF`. (This aligns with the current RFC3629.)
- Fix calling of methods on S4 generics that dispatch on `...` when the call contains `....`
- Following Unicode 'Corrigendum 9', the UTF-8 representations of U+FFFE and U+FFFF are now regarded as valid by `utf8ToInt()`.
- `range(c(TRUE,NA),finite = TRUE)` and similar no longer return `NA`. (Reported by Lukas Stadler.)
- The self starting function `attr(SSlogis,"initial")` now also works when the y values have exact minimum zero and is slightly changed in general, behaving symmetrically in the y range.
- The printing of named raw vectors is now formatted nicely as for other such atomic vectors, thanks to Lukas Stadler.

CHANGES IN R 3.4.2

NEW FEATURES

- Setting the `LC_ALL` category in `Sys.setlocale()` invalidates any cached locale-specific day/month names and the AM/PM indicator for `strptime()` (as setting `LC_TIME` has since R 3.1.0).
- The version of LAPACK included in the sources has been updated to 3.7.1, a bug-fix release.
- The default for `tools::write_PACKAGES(rds_compress=)` has been changed to `"xz"` to match the compression used by CRAN.
- `c()` and `unlist()` are now more efficient in constructing the `names(.)` of their return value, thanks to a proposal by Suharto Anggono. (PR#17284)

UTILITIES

- `R CMD check` checks for and `R CMD build` corrects CRLF line endings in shell scripts `configure` and `cleanup` (even on Windows).

INSTALLATION on a UNIX-ALIKE

- The order of selection of OpenMP flags has been changed: Oracle Developer Studio 12.5 accepts '`-fopenmp`' and '`-xopenmp`' but only the latter enables OpenMP so it is now tried first.

BUG FIXES

- `within(List,rm(x1,x2))` works correctly again, including when `List[["x2"]]` is `NULL`.
- `regexec(pattern,text,*)` now applies `as.character(.)` to its first two arguments, as documented.
- `write.table()` and related functions, `writeLines()`, and perhaps other functions writing text to connections did not signal errors when the writes failed, e.g. due to a disk being full. Errors will now be signalled if detected during the write, warnings if detected when the connection is closed. (PR#17243)
- `rt()` assumed the `ncp` parameter was a scalar. (PR#17306)
- `menu(choices)` with more than 10 choices which easily fit into one `getOption("width")`-line no longer erroneously repeats choices. (PR#17312)
- `length()<-` on a pairlist succeeds. (`https://stat.ethz.ch/pipermail/r-devel/2017-July/074680.html`)
- Language objects such as `quote(("\n"))` or R functions are correctly printed again, where R 3.4.1 accidentally duplicated the backslashes.
- Construction of `names()` for very large objects in `c()` and `unlist()` now works, thanks to Suharto Anggono's patch proposals in PR#17292.
- Resource leaks (and similar) reported by Steve Grubb fixed. (PR#17314, PR#17316, PR#17317, PR#17318, PR#17319, PR#17320)
- `model.matrix(~1,mf)` now gets the row names from `mf` also when they differ from `1:nrow(mf)`, fixing PR#14992 thanks to the suggestion by Sebastian Meyer.
- `sigma(fm)` now takes the correct denominator degrees of freedom for a fitted model with `NA` coefficients. (PR#17313)
- `hist(x,"FD")` no longer "dies" with a somewhat cryptic error message when `x` has extreme outliers or `IQR()` zero: `nclass.FD(x)` tries harder to find a robust bin width h in the latter case, and `hist.default(*,breaks)` now checks and corrects a too large `breaks` number. (PR#17274)
- `callNextMethod()` works for `...` methods.
- `qr.coef(qd,y)` now has correct names also when `qd` is a complex QR or stems from `qr(*,LAPACK=TRUE)`.
- Setting `options(device = *)` to an invalid function no longer segfaults when plotting is initiated. (PR#15883)
- `encodeString(<very large string>)` no longer segfaults. (PR#15885)

- It is again possible to use `configure --enable-maintainer-mode` without having installed `notangle` (it was required in R 3.4.[01]).
- S4 method dispatch on `...` calls the method by name instead of `.Method` (for consistency with default dispatch), and only attempts to pass non-missing arguments from the generic.
- `readRDS(textConnection(.))` works again. (PR#17325)
- `(1:n)[-n]` no longer segfaults for `n <-2.2e9` (on a platform with enough RAM).
- `x <-1:2; tapply(x,list(x,x),function(x) "")[1,2]` now correctly returns `NA`. (PR#17333)
- Running of finalizers after explicit GC request moved from the R interface `do_gc` to the `C` interface `R_gc`. This helps with reclaiming inaccessible connections.
- `help.search(topic)` and `??topic` matching topics in vignettes with multiple file name extensions (e.g., '*.md.rsp' but not '*.Rmd') failed with an error when using `options(help_type = "html")`.
- The X11 device no longer uses the Xlib backing store (PR#16497).
- `array(character(),1)` now gives (a 1D array with) `NA` as has been documented for a long time as in the other cases of zero-length array initialization and also compatibly with `matrix(character(),*)`. As mentioned there, this also fixes PR#17333.
- `splineDesign(..,derivs = 4)` no longer segfaults.
- `fisher.test(*,hybrid=TRUE)` now (again) will use the hybrid method when Cochran's conditions are met, fixing PR#16654.