

The R Journal

Volume 12/1, June 2020

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

Contents

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

Contributed Research Articles

gk: An R Package for the g-and-k and Generalised g-and-h Distributions	5
NlinTS: An R Package For Causality Detection in Time Series	19
mudfold: An R Package for Nonparametric IRT Modelling of Unfolding Processes . .	30
Mapping Smoothed Spatial Effect Estimates from Individual-Level Data: MapGAM .	56
tsmp: An R Package for Time Series with Matrix Profile	74
ari: The Automated R Instructor	85
CopulaCenR: Copula based Regression Models for Bivariate Censored Data in R . .	93
mistr: A Computational Framework for Mixture and Composite Distributions	110
difNLR: Generalized Logistic Regression Models for DIF and DDF Detection . . .	127
BayesMallows: An R Package for the Bayesian Mallows Model	151
Variable Importance Plots—An Introduction to the vip Package	171
Individual-Level Modelling of Infectious Disease Data: EpiILM	195
SurvBoost: An R Package for High-Dimensional Variable Selection in the Stratified Proportional Hazards Model via Gradient Boosting	214
CoxPhLb: An R Package for Analyzing Length Biased Data under Cox Model	227
npordtests: An R Package of Nonparametric Tests for Equality of Location Against Ordered Alternatives	240
rcosmo: R Package for Analysis of Spherical, HEALPix and Cosmological Data . .	265
Tools for Analyzing R Code the Tidy Way	285
spinifex: An R Package for Creating a Manual Tour of Low-dimensional Projections of Multivariate Data	302
SimilaR: R Code Clone and Plagiarism Detection	317
Linear Fractional Stable Motion with the rlfsm R Package	336
The R package NonProbEst for estimation in non-probability surveys	356
ProjectManagement: an R Package for Managing Projects	369
The Rockerverse: Packages and Applications for Containerisation with R	387

Special Articles

S, R, and Data Science	412
Provenance of R's Gradient Optimizers	426

News and Notes

R Foundation News.	434
----------------------------	-----

The R Journal is a peer-reviewed publication of the R Foundation for Statistical Computing. Communications regarding this publication should be addressed to the editors. All articles are licensed under the Creative Commons Attribution 4.0 International license (CC BY 4.0, <http://creativecommons.org/licenses/by/4.0/>).

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

Editor-in-Chief:
Michael Kane

Executive editors:
Dianne Cook, Monash University, Australia
Catherine Hurley, Maynooth University, Ireland
Simon Urbanek, University of Auckland, New Zealand

R Journal Homepage:
<http://journal.r-project.org/>

Email of editors and editorial board:
r-journal@R-project.org

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

Editorial

by Michael J. Kane

On behalf of the editorial board, I am pleased to present Volume 12, Issue 1 of the R Journal and my second issue as the Editor in Chief. Since the last issue Simon Urbanek has joined the editorial board and we have made a few structural changes. First, the R Foundation has approved the R Journal having Associate Editors. This change will allow us to address the increase in submission volume. The addition of the new AE positions should help alleviate some of the workload the editors have been dealing with and will result in shorter turn-around times for submissions. Second, complete issues of the R Journal will no longer be published in a single pdf. The build process for the document was complex and time consuming and we were not seeing the volume of download that would justify the effort. Individual articles are still available and the issue layout is still shown in the “Current Issue” section of the web page.

In this issue

News from the R Foundation is included in this issue along with an update from the The R Foundation’s histoRicalg project, which documents historic and historical numerical algorithms and provides reference implementations in R. In addition, a reprint documenting the history of R, which was initially published in the History of Programming Languages. Finally, this issue features 23 contributed research articles that have been categorized below.

Papers focusing on reproducibility, managing code and projects, and instruction:

- **ari:** The Automated R Instructor
- **ProjectManagement:** an R Package for Managing Projects
- The Rockerverse: Packages and Applications for Containerisation with R
- **SimilaR:** R Code Clone and Plagiarism Detection
- Tools for Analyzing R Code the Tidy Way

Data exploration and visualization:

- **spinifex:** An R Package for Creating a Manual Tour of Low-dimensional Projections of Multivariate Data
- Variable Importance Plots—An Introduction to the **vip** Package

Medicine and epidemiology

- Individual-Level Modelling of Infectious Disease Data: **EpiILM**

Probability distributions and processes

- **BayesMallows:** An R Package for the Bayesian Mallows Model
- **gk:** An R Package for the g-and-k and Generalised g-and-h Distributions
- Linear Fractional Stable Motion with the **r fsm** R Package
- **mistr:** A Computational Framework for Mixture and Composite Distributions
- **mudfold:** An R Package for Nonparametric IRT Modelling of Unfolding Processes

-
- **NlinTS**: An R Package For Causality Detection in Time Series
 - **npordtests**: An R Package of Nonparametric Tests for Equality of Location Against Ordered Alternatives
 - **tsmp**: An R Package for Time Series with Matrix Profile

Supervised learning

- **CopulaCenR**: Copula based Regression Models for Bivariate Censored Data in R
- **CoxPhLb**: An R Package for Analyzing Length Biased Data under Cox Model
- **difNLR**: Generalized Logistic Regression Models for DIF and DDF Detection
- Mapping Smoothed Spatial Effect Estimates from Individual-Level Data: **MapGAM**
- **lspartition**: Partitioning-Based Least Squares Regression
- **SortedEffects**: Sorted Causal Effects in R
- **SurvBoost**: An R Package for High-Dimensional Variable Selection in the Stratified Proportional Hazards Model via Gradient Boosting

Michael J. Kane
michael.kane@r-project.org

gk: An R Package for the g-and-k and Generalised g-and-h Distributions

by Dennis Prangle

Abstract The g-and-k and (generalised) g-and-h distributions are flexible univariate distributions which can model highly skewed or heavy tailed data through only four parameters: location and scale, and two shape parameters influencing the skewness and kurtosis. These distributions have the unusual property that they are defined through their quantile function (inverse cumulative distribution function) and their density is unavailable in closed form, which makes parameter inference complicated. This paper presents the **gk** R package to work with these distributions. It provides the usual distribution functions and several algorithms for inference of independent identically distributed data, including the finite difference stochastic approximation method, which has not been used before for this problem.

Introduction

Statisticians have long sought for a simple extension to the normal distribution which can model data subject to skew, heavy tails or both. One approach is to transform a standard normal random variable $Z \sim \mathcal{N}(0, 1)$ to

$$X = A + BG(Z)H(Z), \quad (1)$$

where A and B are location and scale parameters, $G(\cdot)$ introduces asymmetry, and $H(\cdot)$ elongates the tails of the distribution while having little effect near the mode. This paper considers two such distributions, the *g-and-k* and generalised *g-and-h* distributions. These distributions can model many types of behaviour through just a small number of parameters.

Defining random variables as transformations of Z is equivalent to specifying the distribution's quantile function (defined in the next section), and distributions of this type are known as *quantile distributions*. Work on quantile distributions goes back at least to [Hastings et al. \(1947\)](#). See [Gilchrist \(2000\)](#) for a book length treatment of their history and use in statistics. [Tukey \(1977\)](#) proposed the form (1) and a distribution using it: the original *g-and-h* distribution. [Haynes et al. \(1997\)](#) were the first to use the two distributions considered in this paper: the *g-and-k* distribution and a generalised form of the *g-and-h* distribution. For brevity henceforth “*g-and-h distribution*” will refer to their generalised form. See [Peters et al. \(2016\)](#) for a thorough review of these and other distributions based on (1).

Applications of the *g-and-k* and *g-and-h* distributions have included environmental data ([Rayner and MacGillivray, 2002](#)), financial returns ([Drovandi and Pettitt, 2011](#)) and insurance risk ([Peters et al., 2016](#)). There has also been considerable methodological work on inference for these distributions (e.g. [Rayner and MacGillivray, 2002](#); [Haynes and Mengersen, 2005](#); [Allingham et al., 2009](#); [Drovandi and Pettitt, 2011](#); [Fearnhead and Prangle, 2012](#)). This is because it is not possible to express the densities of quantile distributions in closed form beyond some special cases, which makes it difficult to apply standard likelihood-based inference methods.

This paper presents the **gk** R package to work with the *g-and-k* and *g-and-h* distributions. The remaining sections covering the following:

- A mathematical definition of the distributions.
- A description of the package's functions to perform standard distributional tasks and how they are implemented.
- An exploration of the range of valid parameters for these distributions, as this has a complicated form. We propose a novel rule giving “safe” parameter values for the *g-and-k* distribution.
- A description of several methods for parameter inference and corresponding functions supplied by the package.
- An illustrative analysis of a real dataset.
- A summary.

Definitions

The cumulative distribution function (cdf) of a univariate random variable X , $F_X : \mathbb{R} \rightarrow [0, 1]$, is defined as $\Pr(X \leq x)$. (Later we will often drop subscripts where they are clear from the context.) The cdf suffices to completely specify the probability distribution of X . It is often the case that the cdf is not available in closed form but is implicitly defined through its derivative, the probability density function (pdf). An example of this is the normal distribution.

For quantile distributions, the cdf is implicitly defined through its inverse, the *quantile function* $F_X^{-1}(u)$ where $F_X^{-1} : [0, 1] \rightarrow \mathbb{R}$. The g -and- k and g -and- h distributions use a quantile function of the form $F_X^{-1}(u; \theta) = Q(z(u); \theta)$ where $z(\cdot)$ is the $\mathcal{N}(0, 1)$ quantile function and θ is a vector of parameters. The Q functions are:

$$Q_{gk}(z; A, B, g, k, c) = A + B (1 + c \tanh[gz/2]) z (1 + z^2)^k \quad (2)$$

$$Q_{gh}(z; A, B, g, h, c) = A + B (1 + c \tanh[gz/2]) z \exp(hz^2/2). \quad (3)$$

It is possible to sample from the distributions using the *inversion method*, that is, by simulating $U \sim \mathcal{U}(0, 1)$ and substituting it into the quantile function. Equivalently one can sample $Z \sim \mathcal{N}(0, 1)$ and substitute it into Q_{gk} or Q_{gh} i.e. the process described in the introduction based on Equation (1). In terms of (1), $G(z) = 1 + c \tanh(gz/2)$ produces asymmetry and $H(z) = z(1 + z^2)^k$ or $z \exp(hz^2/2)$ elongates tails.

Each distribution has four main parameters: A (location), B (scale), g (shape parameter mainly affecting skewness), and k or h (shape parameter mainly affecting kurtosis). The remaining parameter c is discussed below. When both shape parameters are zero the distribution is simply $\mathcal{N}(0, 1)$. An illustration of the flexible shapes that the g -and- k density can take is given in Figure 1. The g -and- h can produce similar shapes, with the following exception. The g -and- k distribution allows negative values of k which can produce lighter tails than a normal distribution, but also bimodal distributions of potentially limited usefulness.

Well-defined continuous distributions result from parameter values producing strictly increasing quantile functions. Determining when this is true is complicated so discussion is postponed to a later section. For now note that it is standard to take $B > 0$ and fix $c = 0.8$ (which will be assumed throughout unless mentioned otherwise), and in this case $k \geq 0$ or $h \geq 0$ guarantees a valid distribution.

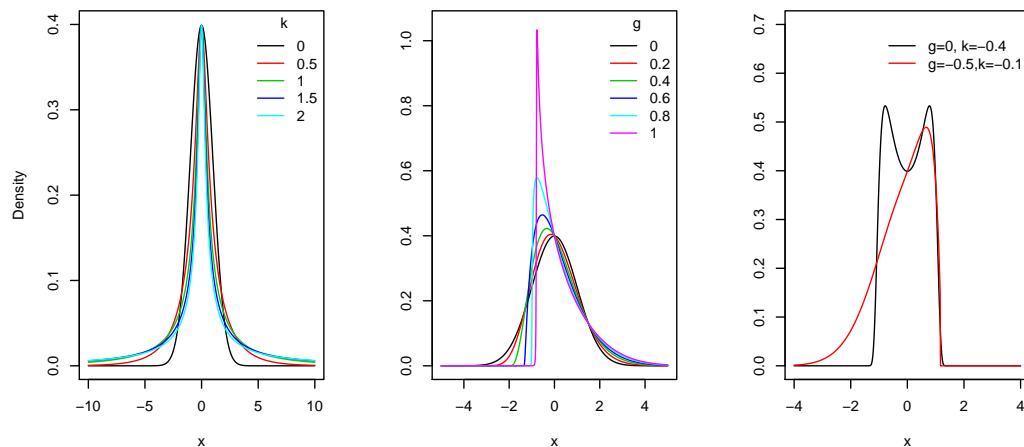


Figure 1: Example g -and- k densities. The first panel fixes $g = 0$ and varies k , mainly altering kurtosis. The second fixes $k = 0$ and varies g , mainly altering skewness. The third shows two examples with $k < 0$.

Distribution functions

The **gk** package provides the standard suite of R functions for the g -and- k and g -and- h distributions i.e. random sampling and calculation of the pdf, cdf and quantile functions. This section describes

how these functions are implemented. It is assumed that parameters have been chosen such that the quantile function is strictly increasing. No warning is given when this is not the case as checking validity is time consuming (see next section).

Quantile function The `qgk` and `qgh` functions calculate the quantile function $F^{-1}(u)$. Their implementation is straightforward. First $z(u)$ is calculated using `qnorm`, then this is passed to an internal function, `z2gk` or `z2gh`, which computes Q_{gk} or Q_{gh} .

Random sampling The `rgk` and `rgh` functions perform random sampling. This is done by the method described earlier of sampling $\mathcal{N}(0, 1)$ draws and substituting them into Q_{gk} or Q_{gh} , via the function `z2gk` or `z2gh`.

Cumulative distribution function The `pgk` and `pgh` functions calculate the cdf $F(x)$ given input x . They numerically solve $Q(z) - x = 0$, which is guaranteed to have a unique root for z . The required final output is then $u = \Phi^{-1}(z)$ where Φ is the $\mathcal{N}(0, 1)$ cdf. An alternative approach would be to directly solve $Q(z(u)) - x = 0$ for u . However we found this was less numerically stable for u close to 0 or 1.

Our code finds the root for z using R's `uniroot` command and `z2gk` or `z2gh` for $Q(z)$ evaluations. The need to run a root finding algorithm means this function is slow relative to cdf calculations of standard distributions - see Table 1.

The functions include an argument `zscale`. Setting this to `TRUE` outputs the z value which is found rather than u . This is used in the density functions below, and more generally is also useful to retain numerical precision when z has large magnitude.

Probability density function The `dgk` and `dgh` functions calculate the pdf $f(x)$, or the log pdf if the argument `log=TRUE` is supplied. The method is based on the standard probability result that if A has density $f_A(a)$ and $t(a)$ is a differentiable 1-1 transformation then the density of $B = t(A)$ is

$$f_B(b) = f_A(a)/t'(a) \quad \text{where } a = t^{-1}(b),$$

and t' denotes the first derivative of t .

For quantile distributions we have $Z \sim \mathcal{N}(0, 1)$ and $X = Q(Z)$ for some Q function. So the pdf of X is

$$f(x) = \phi(z)/Q'(z) \quad \text{where } z = Q^{-1}(x),$$

where $\phi(z)$ is the $\mathcal{N}(0, 1)$ pdf.

Our code to calculate the pdf first finds $z = Q^{-1}(x)$ using `pgk` or `pgh` with `zscale=TRUE`. Then the pdf or its log is calculated using formulae (4) and (5) (see Appendix A) for $Q'(u)$. The reliance on performing root finding within `pgk` and `pgh` means that `dgk` and `dgh` are slow relative to pdf calculations for standard distributions - see Table 1.

Note that an alternative representation of $f(x)$ is $1/q'(u)$ where $q(u)$ represents $F^{-1}(u)$ and $u = F(x)$. Density calculations based on this approach are described in Rayner and MacGillivray (2002). However we found that calculating the u values required for this approach was occasionally numerically unstable, as mentioned above.

Cost Table 1 compares the time to execute `gk`'s distributional functions to those for the normal distribution. It illustrates that random sampling and quantile function calculation are reasonably efficient, but calculating the cdf and pdf are expensive.

Range of valid parameters

Recall that a valid continuous distribution requires the quantile function to be strictly increasing. Clearly this property is unaffected by the choice of A and $B > 0$. This section discusses the effects of g, h, k and c .

Several theoretical results on valid parameters can be derived. It's convenient to concentrate on $c \geq 0$. In this case $h < 0$ or $k < -1/2$ is invalid. Taking $k \geq 0$ or $h \geq 0$ produces valid distributions when $0 \leq c < c^* \approx 0.83$. This is the reason for taking $c = 0.8$ as standard: it maintains this property while allowing the skewness factor in (2) and (3) to have a large effect. For justification of all these results, see Appendix B.

	Time (microseconds)			Ratio vs normal	
	Normal	<i>g-and-k</i>	<i>g-and-h</i>	<i>g-and-k</i>	<i>g-and-h</i>
Quantile function	175	972	445	5.56	2.55
Random sampling	150	921	436	6.15	2.91
cdf	313	143151	116928	457	374
pdf	369	138381	111279	375	302

Table 1: Mean times to perform various distributional operations, evaluated by the `microbenchmark` package (Mersmann, 2015). For example the random sampling row compares `rnorm(N)`, `rgk(N, 1, 2, 3, 4)` and `rgh(N, 1, 2, 3, 4)` for $N = 100$. We also tried $N = 1$, which gave qualitatively similar results but slightly better relative efficiency of the `gk` functions.

When $c = 0.8$, the above results completely characterise the range of valid parameters for the *g-and-h* distribution. For the *g-and-k* distribution, there is still some uncertainty for $-0.5 \leq k < 0$, which, as mentioned earlier, corresponds to light tails. For both distributions, the case where $c > c^*$ is less clear: even positive values of k or h do not guarantee validity. Therefore we provide the function `isValid` to test parameter validity numerically.

Validity can be checked by testing whether the minimum derivative of (2) or (3) is positive. Appendix A shows that it is equivalent to test whether the functions (6) or (7) are positive. `isValid` uses numerical optimisation to minimise these and returns whether the minimum value is positive. To reduce the possibility of finding local minima, multiple optimisation starting points can be supplied as a vector to the argument `initial_z`. However it is still not guaranteed that the global minimum is found, so there remains a possibility that the function may produce false positives.

The function can be used as follows to illustrate the region of valid *g-and-k* parameter values for $c = 0.8$. The results are plotted as Figure 2.

```
gk_grid = expand.grid(g = seq(-10, 10, 0.1), k = seq(-0.6, 0.1, 0.01))
v = isValid(gk_grid$g, gk_grid$k)
```

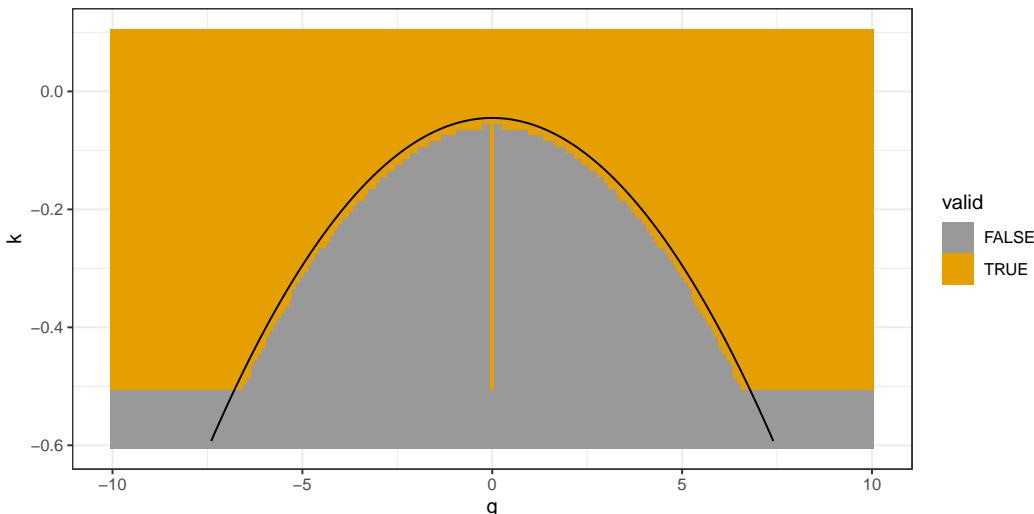


Figure 2: Validity of parameter values for the *g-and-k* distribution when $c = 0.8$, calculated using `isValid`. Also shown is a quadratic function $\tilde{k}(g)$ near the curved part of the boundary between the regions.

We do not test validity automatically within the package's other functions. This is because `isValid` is relatively computationally expensive and not guaranteed to be correct. Therefore particular care should be taken for $k < 0$ or $c > c^*$, as the distribution functions will not provide warnings when invalid parameters are used. A reasonable region of g and k values to use in practice with $c = 0.8$ can be derived from Figure 2. It shows that for $|g| < 7$ some $-0.5 \leq k < 0$ values are invalid. Apart from a narrow strip near $g = 0$, this invalid region's boundary is roughly quadratic, as illustrated by the curve $\tilde{k}(g) = -0.045 - 0.01g^2$ on the figure. Based on this analysis, $k \geq \max(-0.5, \tilde{k}(g))$ seems a reasonable sufficient condition for parameter validity to use in practice.

Inference functions

The package provides three inference methods for data x_1, x_2, \dots, x_n which are assumed to be independent and identically distributed (IID) draws from a g -and- k or g -and- h distribution with unknown parameters. This section describes these methods. An illustration of their use is provided in the next section. See the `gk` help files for a full description of all the arguments available.

MCMC inference The `mcmc` function implements inference using *Markov chain Monte Carlo* (MCMC). This samples from a Markov chain whose stationary distribution is the Bayesian posterior of interest for the parameters θ . We use a Metropolis-Hastings algorithm, in which a proposed new state of the chain θ' is sampled by adding a $\mathcal{N}(0, \Sigma)$ increment to the current state θ_t . A decision to accept or reject θ' is made based on the prior and likelihood values at θ_t and θ' and a random variable (see steps 3-4 of Algorithm 1.).

Tuning Σ can be difficult. Haynes and Mengerson (2005), who first used MCMC for the g -and- k distribution, did this manually. Instead we use the adaptive Metropolis (AM) algorithm of Haario et al. (2001) which tunes Σ automatically during its operation. The resulting θ_t s no longer form a Markov chain, but it has been proved (Saksman and Vihola, 2010) that, under suitable conditions, calculations using them still converge to posterior quantities as the length of the chain increases. The AM algorithm is presented as Algorithm 1. Step 1 states the proposal matrix used in terms of the empirical variance of the past MCMC states. To calculate this empirical variance efficiently, the code updates it each time a new state is observed. As a default we specify tuning choices $\epsilon = 10^{-6}$ and $t_0 = 100$.

Like other Bayesian methods, MCMC requires a prior density for the parameters, $\pi(\theta)$, to be specified. This must be supplied by the user. For computational convenience this should be supplied in the form of a function `get_log_prior` which takes a vector of parameters as input and returns the log prior density. We allow the user to reparameterise θ , using $\log B$ rather than B , via the `logB` argument. This can improve MCMC efficiency when the posterior for B is concentrated on values close to zero.

For IID data the likelihood is $L(\theta) = \prod_{i=1}^n f(x_i; \theta)$, the product each observation's pdf. Evaluating this for the g -and- k or g -and- h distributions using the `pgk` or `pgh` command requires n calls to numerical optimisation. Therefore MCMC becomes computationally expensive for even moderately large datasets.

Algorithm 1 The Adaptive Metropolis MCMC algorithm

Input: observations x , prior density $\pi(\theta)$, number of iterations to perform N , initial state θ_0 , initial variance matrix Σ_0 , pre-tuning period t_0 , tuning parameter $\epsilon > 0$.

Loop over $1 \leq t \leq N$:

1. If $t \leq t_0$ let $\Sigma_t = \Sigma_0$. Otherwise let $\Sigma_t = \frac{1}{4}(2.4)^2 (\hat{\Sigma}_{t-1} + \epsilon I)$, where $\hat{\Sigma}_{t-1}$ is the variance of $\theta_1, \theta_2, \dots, \theta_{t-1}$.
2. Sample $\theta' \sim \mathcal{N}(\theta_{t-1}, \Sigma_{t-1})$
3. Sample $u \sim \mathcal{U}(0, 1)$ and let $r = \frac{\pi(\theta')L(\theta')}{\pi(\theta_{t-1})L(\theta_{t-1})}$.
4. If $u < r$ let $\theta_t = \theta'$. Otherwise let $\theta_t = \theta_{t-1}$.

Output: sample $\theta_0, \theta_1, \dots, \theta_N$.

ABC inference The `abc` function implements inference by *approximate Bayesian computation* (ABC). This is a method for approximate Bayesian inference which avoids evaluating the likelihood function. It is especially useful when the likelihood function is unavailable or, as for quantile distributions, is expensive to compute. ABC is based instead on finding parameter values which produce simulated data similar to the observations. The `abc` function implements a simple version of ABC, Algorithm 2. Here a simulation is accepted if it has one of the M smallest distances to the observations. Distance refers to a weighted version of Euclidean distance between vectors of simulated and observed summary statistics. Details of the weighting are given in the algorithm's description. (For n large, `abc` avoids high memory requirements by running several batches of

Algorithm 2. Each batch uses $N = 10^4$ and returns the M best simulations. The overall best M best simulations are then found and returned. The v_j weights calculated in the first batch are reused in the others.)

Like `mcmc`, `abc` is a Bayesian method and requires a prior distribution for θ to be provided. It is convenient for this to be provided in a different form to the `mcmc` case. A function `rprior` should be supplied which a single numeric input and returns that many samples from the prior distribution as rows of a matrix.

Algorithm 2 Approximate Bayesian computation (ABC)

Input: observations x , prior distribution $\pi(\theta)$, summary statistic function $s(x)$, number of simulations to perform N , number of samples to output M .

1. Calculate observed summaries $s_0 = s(x)$.
 2. For $1 \leq i \leq N$ sample parameters θ_i from the prior.
 3. For $1 \leq i \leq N$ simulate summary statistics $s(x_i)$ given parameters θ_i . Let s_{ij} denote the j th component of $s(x_i)$.
 4. For $1 \leq j \leq q$ (where $q = \dim(s_0)$) calculate the empirical variance v_j of the $(s_{ij})_{1 \leq i \leq n}$ values.
 5. For $1 \leq i \leq N$ let $d_i = \sum_{j=1}^q (s_{ij} - s_{0j})^2 / v_j$.
 6. Find the M smallest d_i values and return the corresponding θ_i s.
-

ABC produces samples from an approximation to the Bayesian posterior distribution. The quality of the approximation depends in a complex way on the choice of summary statistics and the tuning parameters N and M . For more background on ABC see the review paper by Marin et al. (2012) and the handbook of Sisson et al. (2017). Two general R packages for ABC which implement more advanced methods are `abc` (Csilléry et al., 2012) and `EasyABC` (Jabot et al., 2013).

Using ABC for the g -and- k and g -and- h distributions was proposed by Allingham et al. (2009) and has been investigated in many subsequent papers. Following Drovandi and Pettitt (2011) we offer three choices of summary statistics which can be selected through the `sumstats` argument: (1) the full order statistics; (2) octiles of the observations, E_1, E_2, \dots, E_7 ; (3) robust estimates of the moments based on the octiles:

$$S_A = E_4, \quad S_B = E_6 - E_2, \quad S_g = (E_6 + E_2 - 2E_4) / S_b, \quad S_k = (E_7 - E_5 + E_3 - E_1) / S_b.$$

Many more sophisticated approaches to choosing ABC summary statistics have been proposed (Blum et al., 2013), but these are a simple starting point.

For summaries (2) or (3) we follow Fearnhead and Prangle (2012) and speed up step 3 of Algorithm 2 by using the fact that the octiles (or close approximations) can be simulated quickly without the need to simulate a full dataset. Suppose X_1, X_2, \dots, X_N are g -and- k or g -and- h variables, and let $X_{(1)} < X_{(2)} \dots < X_{(N)}$ denote the order statistics. We replace E_i with $E'_i = X_{(r(iN/8))}$ where $r(\cdot)$ rounds to the nearest integer. Now we need to simulate 7 order statistics from the g -and- k or g -and- h distribution. To do so we simulate corresponding order statistics of the $\mathcal{U}(0, 1)$ distribution using the exponential spacings method (Ripley, 1987). This is implemented by the `orderstats` function. The uniform order statistics are then substituted into $F^{-1}(u)$.

FDSA inference The `fdsa` function performs inference using *finite difference stochastic approximation* (FDSA). FDSA, originally due to Kiefer and Wolfowitz (1952), attempts to find θ^* minimising a loss function $\mathcal{L}(\theta)$ by iteratively calculating estimates $\theta_1, \theta_2, \dots$. Each iteration moves the estimate in the opposite direction to an estimate of the loss gradient, based on finite difference calculations.

We use FDSA for maximum likelihood estimation of IID observations. In this setting $\mathcal{L}(\theta)$ can be taken to be the negative log likelihood,

$$\mathcal{L}(\theta) = -\log L(\theta) = -\sum_{i=1}^n \log f(x_i; \theta).$$

The gradient of $\mathcal{L}(\theta)$ can be estimated using only a small subset of the data, so FDSA has the potential to scale up to large datasets better than MCMC, while avoiding the approximation error of ABC. Unlike ABC and MCMC, we are not aware of FDSA having previously been used for the g -and- k and g -and- h distributions.

The g -and- k and g -and- h distributions have some parameter constraints (e.g. $B > 0$, $h \geq 0$). Also we found setting further constraints from preliminary analyses sometimes helps FDSA behave well. Therefore we use a version of FDSA for bounded minimisation from L'Ecuyer and Glynn (1994), presented as Algorithm 3.

Algorithm 3 Finite difference stochastic approximation (FDSA)

Input: initial state θ_0 , choice of a_t and c_t sequences, function $\hat{\mathcal{L}}(\cdot)$ which calculates an unbiased estimate of $\mathcal{L}(\cdot)$, number of iterations to perform N , vectors of (possibly infinite) upper and lower parameter bounds θ^+, θ^- .

Loop over $0 \leq t \leq N - 1$.

1. Calculate \hat{g}_t by performing the following steps for $i \leq 1 \leq 4$.
 - (a) Let Δ_i be a 4-dimensional vector whose i th component is 1 and others are zero.
 - (b) Let $\phi^+ = P(\theta_t + c_t \Delta_i)$ and $\phi^- = P(\theta_t - c_t \Delta_i)$.
(Here $P(\phi)$ is a projection operator. Its output is ϕ' such that ϕ'_i is the closest value to ϕ_i in $[\theta_i^-, \theta_i^+]$. The i subscripts represent i th components.)
 - (c) Let $\hat{g}_{it} = \frac{1}{|\phi_i^+ - \phi_i^-|} [\hat{\mathcal{L}}(\phi^+) - \hat{\mathcal{L}}(\phi^-)]$.
2. Let $\theta_{t+1} = P(\theta_t - a_t \hat{g}_t)$.

Output: Final estimate θ_N .

The unbiased estimate of $\mathcal{L}(\theta)$ required by Algorithm 3, $\hat{\mathcal{L}}(\theta)$, can be taken to be the sum of a random sample of m negative log likelihood terms multiplied by n/m . Hence for a vector y containing a random subsample of m observations (sometimes referred to as a *batch*), $\hat{\mathcal{L}}(\theta)$ can be calculated using `-sum(dgk(y,A,B,g,k,log=TRUE))*n/m` (or similar for the g -and- h distribution). Variance reduction in step 1c of Algorithm 3 is possible by coupling the two estimates (Kushner and Yin, 2003). Hence we use the same random subsample of data for all $\hat{\mathcal{L}}$ calculations in an iteration of step 1.

FDSA convergence requires that the *gain* sequences a_t and c_t must satisfy certain conditions. Following Spall (1998) we take $a_t = a_0 (A + t + 1)^{-\alpha}$ and $c_t = c_0 (t + 1)^{-\gamma}$. This leaves several tuning choices, which can be selected by the user, or left at default values which we provide. Following Kleinman et al. (1999) we use default values $\alpha = 1$ and $\gamma = 0.49$. Following Spall (1998) our default for c_0 is an estimate of the standard deviation of $\hat{\mathcal{L}}(\theta_0)$ using some preliminary simulations. We provide defaults $a_0 = 1$ and $A = 100$ but it is recommended to manually tune these to produce rapid convergence. This may require several short pilot runs of the algorithm. The `fdsa` function allows a_0 and c_0 to be vectors, in which case operations in Algorithm 3 are interpreted as elementwise where necessary. This allows the user to tune gain sequences differently for each parameter. As for `mcmc`, we allow the user to reparameterise θ , using $\log B$ rather than B , via the `logB` argument, which can improve FDSA efficiency when the MLE value of B is close to zero.

Under weak assumptions, FDSA converges to a local minimum of $\mathcal{L}(\theta)$ (Kushner and Yin, 2003). In our experience the likelihood for the g -and- k and g -and- h distributions is usually unimodal, so there is little danger of converging to an incorrect mode. Nonetheless it may be a useful check on the results to rerun the algorithm from various starting points or compare with the output of another algorithm.

An alternative to FDSA is *simultaneous perturbation stochastic approximation* (SPSA) (Spall, 1998). Here each iteration makes a finite difference estimate of the derivative of the loss function when moving in a random direction from θ_t . An update moves θ_t a distance (negatively) proportional to this estimated derivative in the selected direction. Each SPSA iteration requires fewer likelihood estimates than FDSA, and it is asymptotically more efficient (Kushner and Yin, 2003). However we found in exploratory work that for our application the SPSA updates were dominated by improving A and B estimates, and the remaining parameters were learned very slowly.

Illustration

We illustrate **gk**'s inference methods on the **Garch** exchange rate dataset from the **Ecdat** package (Croissant, 2016). This consists of 1967 daily US dollar exchange rates against other currencies from 1980 to 1987. We concentrate on the exchange rate with Canadian Dollars. Let x_t denote the exchange rate on day t . The *log return* is defined as $\log(x_{t+1}/x_t)$. Figure 3 is a time series plot of the log returns. Figure 7 shows a histogram and a quantile-quantile plot indicating that the tails are heavier than those of a normal density.

We focus on using the *g-and-k* distribution to model the log returns under an IID assumption. For models also including time series structure see for example Drovandi and Pettitt (2011). The full code for the analysis below can be run via the **fx** function.

The ABC and MCMC analyses which follow are Bayesian and require specification of a prior. We use a uniform prior for ease of comparison to the maximum likelihood results from FDSA. For MCMC we are able to use an improper uniform prior. For ABC a proper prior is required so we bound the parameters as follows $-1 < A < 1$, $0 < B < 1$, $-5 < g < 5$, $0 < k < 10$. We restrict A and B to magnitude 1 at most, as we believe log returns of this magnitude are highly unlikely. The g and k parameters are given wider support which can capture a broad range of distributional shapes.

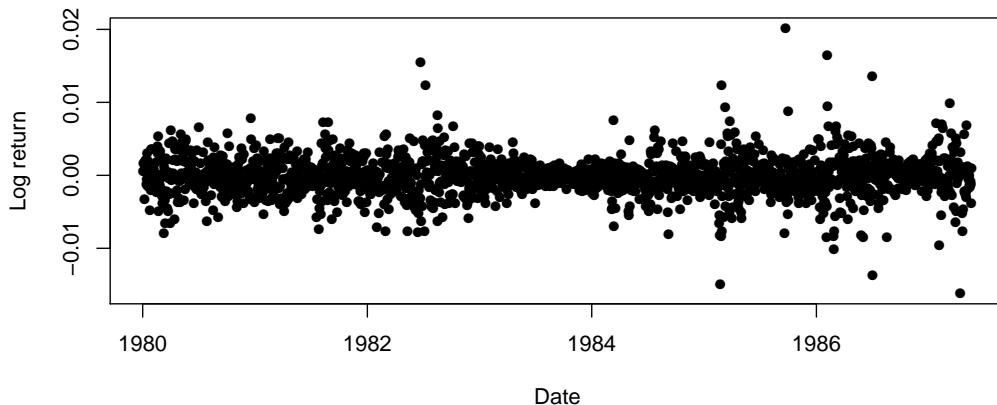


Figure 3: Log returns for US dollar / Canadian Dollar exchange rates.

ABC We ran ABC as follows:

```
rprior = function(i) {
  cbind(runif(i, -1, 1), runif(i, 0, 1), runif(i, -5, 5), runif(i, 0, 10))
}
abc_out = abc(log_return, N = 1E7, rprior = rprior, M=200,
  sumstats = 'moment estimates')
```

This simulated 10^7 parameter vectors and accepting the best 200. We used moment estimator summary statistics, described earlier, which can be simulated quickly without the need to simulate an entire dataset. As a result this analysis took only 6 minutes.

The resulting approximate posterior samples are shown in Figure 6. Figure 7 shows density and quantile-quantile plots under the mean parameter values. These reveal a very poor fit to the data. However this short ABC analysis does provide reasonable tuning choices for the other methods.

FDSA We ran FDSA as follows:

```
abc_out_tf = abc_out[, 1:4]
abc_out_tf[, 2] = log(abc_out_tf[, 2])
abc_est_tf = colMeans(abc_out_tf)
fdsa_out_pilot = fdsa(log_return, N = 1E4, logB = TRUE, theta0 = abc_est_tf,
  batch_size = 100, a0 = 2E-4)
a0 = c(1E-6, 1E-2, 1E-2, 1E-2)
```

```
fdsa_out = fdса(log_return, N = 1E4, logB = TRUE, theta0 = abc_est_tf,
batch_size = 100, a0 = a0)
```

We found that using the original parameterisation caused high variance in our gradient estimates. This is because the log-likelihood surface becomes extremely steep for B close to 0. Therefore we reparameterised B to $\log B$. The initial FDSA state was set to equal the ABC means. The FDSA steps sizes a_0 were tuned by trial-and-error.

Figure 4 shows a trace plot of the FDSA algorithm output. A pilot run with $a_0 = 2 \times 10^{-4}$ is shown in black. Parameters $\log B$, g and k do not converge over 10,000 iterations. However they have smooth curves, indicating that there is relatively little noise in their gradient estimates and so larger steps could be taken. In contrast A converges quickly and then oscillates noisily. This indicates that a smaller step size could be used to average out this noise more effectively without endangering convergence. Therefore for the final run we used $a_0 = (10^{-6}, 10^{-2}, 10^{-2}, 10^{-2})$.

The final FDSA analysis took 17 minutes. The final states were $A = 9.1 \times 10^{-5}$, $B = 1.7 \times 10^{-3}$, $g = 2.0 \times 10^{-2}$ and $k = 0.35$. Figure 7 shows density and quantile-quantile plots under these parameter values. These are a much better fit to the data than the ABC results.

Next we use the FDSA results to help tune an MCMC algorithm, which quantifies the uncertainty in the parameter values.

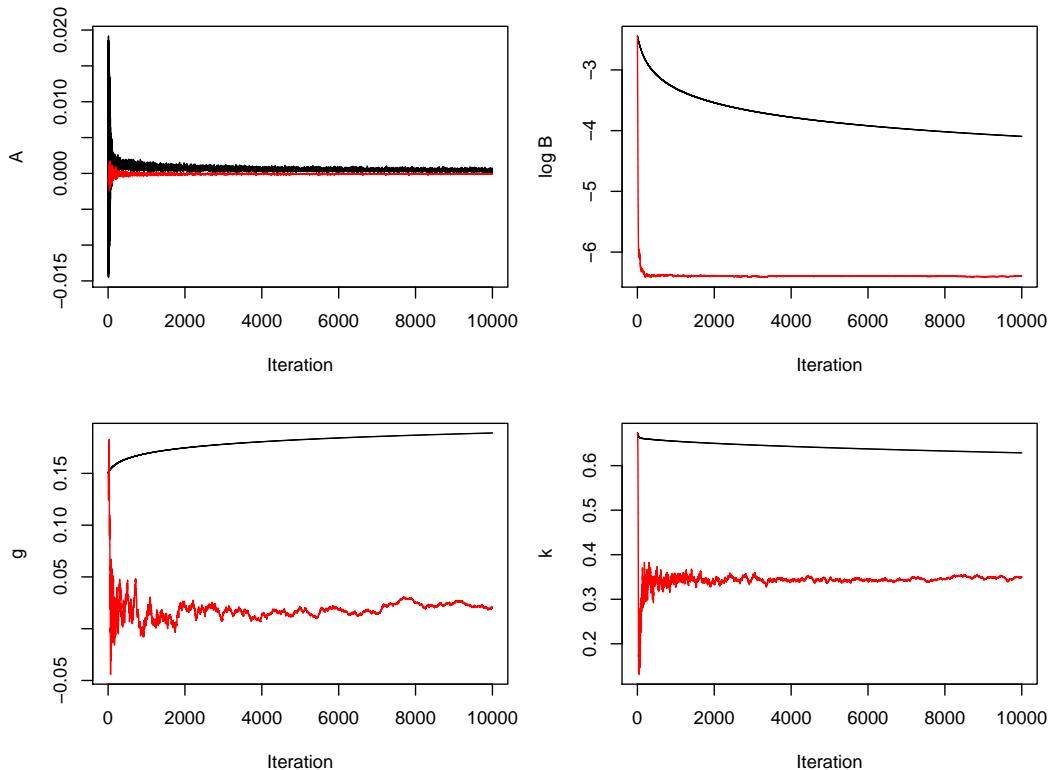


Figure 4: Output from the FDSA algorithm to infer g -and- k parameters for exchange rate log returns. Black shows output from a pilot run with $a_0 = 2 \times 10^{-4}$. Red shows output from the final run with $a_0 = (10^{-6}, 10^{-2}, 10^{-2}, 10^{-2})$.

MCMC We ran MCMC as follows:

```
fdса_est_tf = fdса_out[1E5, 1:4]
Sigma0 = var(fdса_out[1E5 + (-1000:0), 1:4])
log_prior = function(theta) {
  if (theta[4] < 0) return(-Inf)
  return(theta[2])
}
mcmc_out_tf = mcmc(log_return, N = 1E4, logB = TRUE, get_log_prior = log_prior,
theta0 = fdса_est, Sigma0 = Sigma0)
```

Again we used a log reparameterisation for B . To achieve an improper uniform prior on the original

parameterisation, we used a prior density proportional of $B\mathbf{1}(k > 0)$ on $(A, \log B, g, k)$ (where $\mathbf{1}$ represents an indicator function). Our initial parameter vector was the final FDSA state. We use the variance matrix of the last 1000 FDSA states to select the initial MCMC proposal variance.

Figure 5 shows a trace plot of the MCMC algorithm output. For the first few hundred iterations small proposals are made, at least for $\log B$, g and k , but the proposal variance quickly adapts and the remainder of the output appears to have converged. Exploratory work showed that taking a poor initial state meant MCMC is very slow to converge, because the variance matrix adapts to the transient state of the algorithm. Hence tuning based on FDSA output is very useful.

The MCMC analysis took 39 minutes. Figure 6 parameter histograms and figure 7 shows density and quantile-quantile plots. These are similar to the FDSA fit. Note that the density plot is based on mean parameter values from the MCMC output (after discarding the first half of the output as burn-in and transforming $\log B$ values back to the original parameterisation).

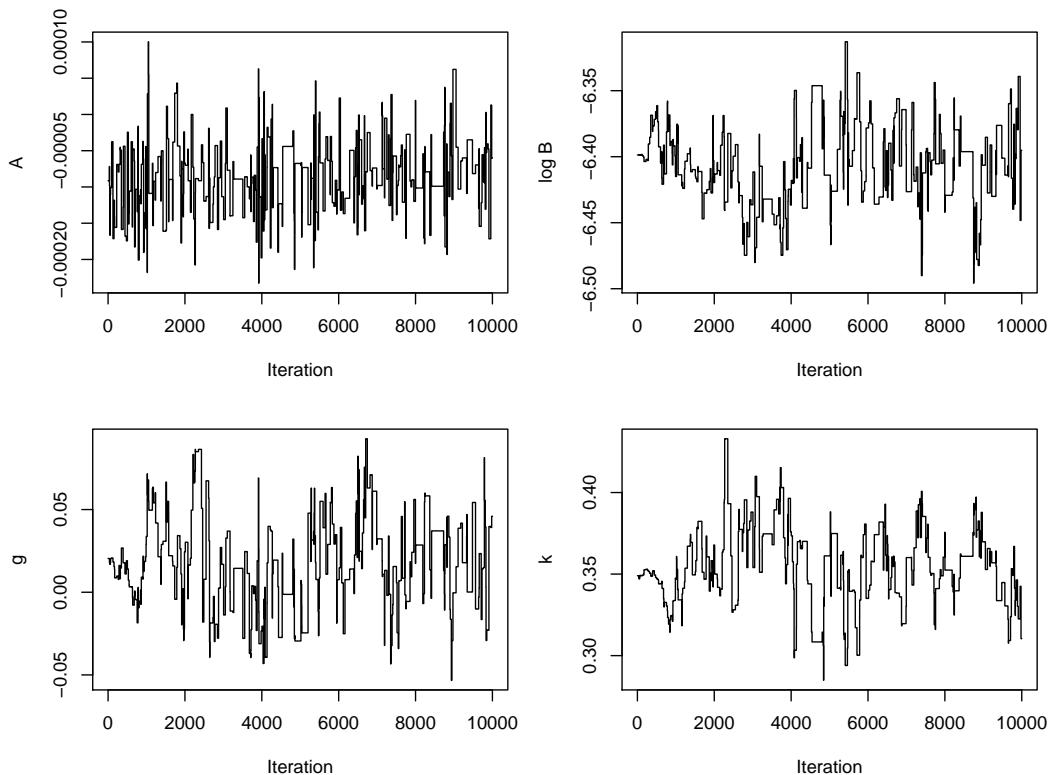


Figure 5: States of an MCMC algorithm to infer g -and- k parameters for exchange rate log returns.

Summary The ABC analysis is quick but produces a poor fit. However it helps tune the FDSA method which finds a good estimate of the MLE in a reasonable time. Further computational effort using MCMC provides a Bayesian fit. Figure 7 shows that the g -and- k distribution fits the data better than a normal distribution, but still does not fit the most extreme observations. Further improvements might be possible by using more flexible distributions, for example allowing different k parameters for the upper and lower tails (Peters et al., 2016).

Discussion

This paper has reviewed the g -and- k and g -and- h distributions, and introduced the **gk** package to work with them. The package includes the usual distributional functions, although the pdf and cdf functions are slow due to relying on numerical root-finding. Another function tests the validity of different parameter combinations, and this was used to produce a novel result on which parameters are valid for the g -and- k distribution (i.e. it appears to be sufficient that $k \geq \max(-0.5, -0.045 - 0.01g^2)$.) The package also provides several methods for inference of IID data under these distributions, and their use has been illustrated above. The methods include a

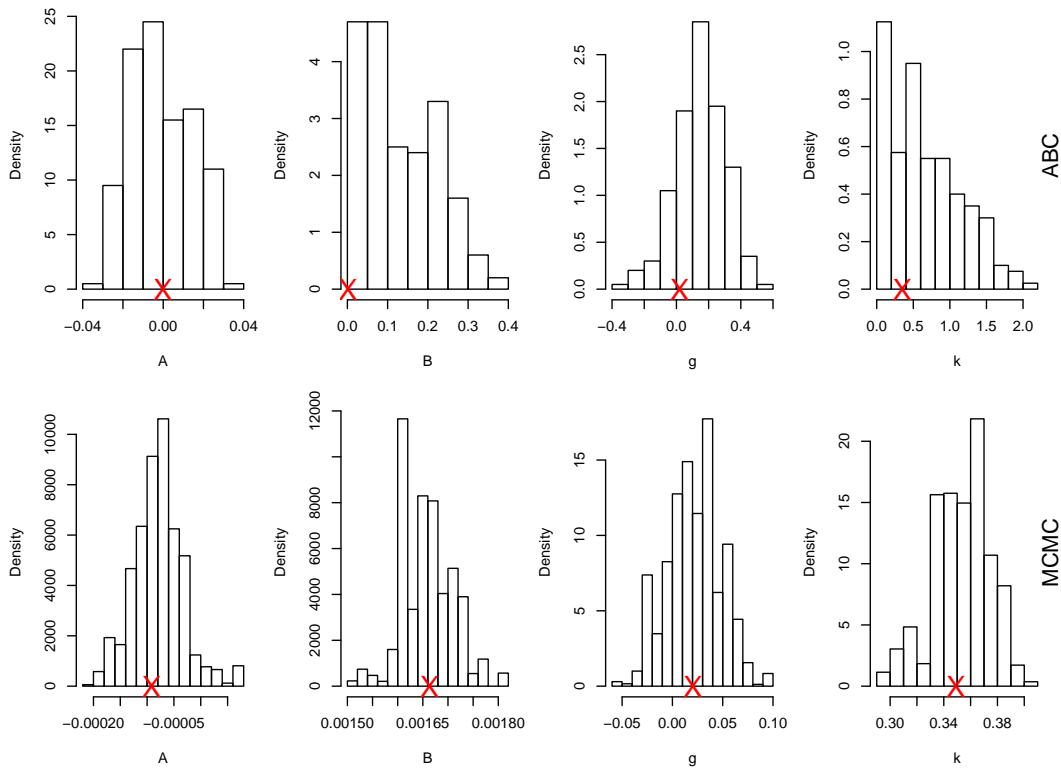


Figure 6: Parameter inference for fitting the g -and- k distribution to exchange rate log returns. The top row shows the ABC posterior sample and the bottom row the MCMC posterior sample, which requires much more concentrated parameter scales. FDSA estimates of the MLEs are shown by crosses on the x -axis.

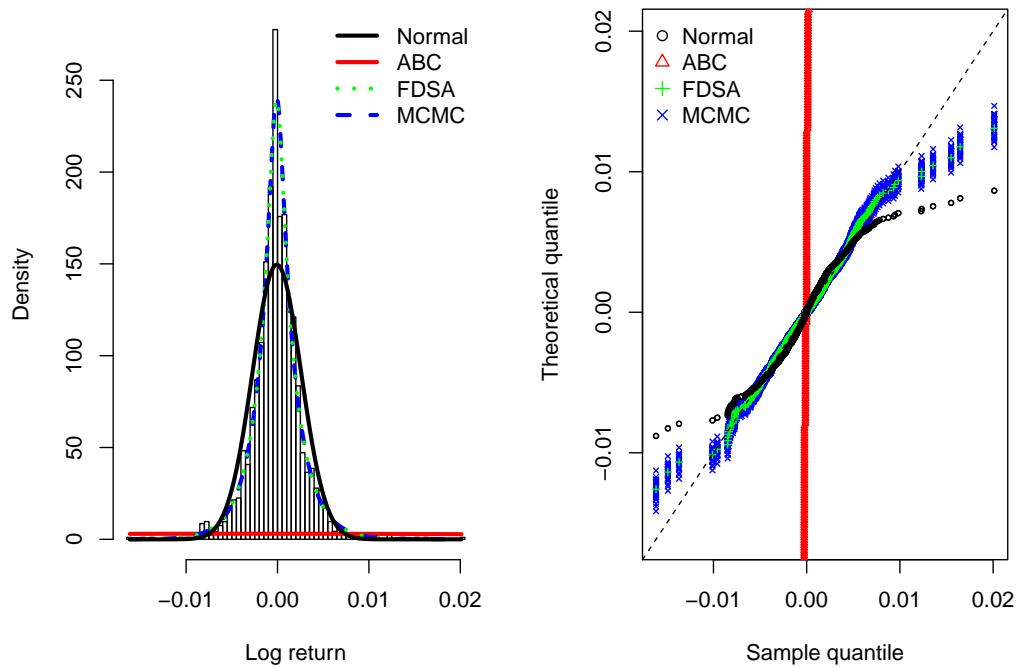


Figure 7: (Left) Histogram of exchange rate log returns, and fitted g -and- k densities. (Right) Quantile-quantile (QQ) plots of fitted g -and- k densities. QQ plots are shown for 30 vectors of parameters sampled from the second half of the MCMC output.

FPSA algorithm which can find MLEs for large datasets in a reasonable time and has not been applied to this problem before.

Appendix A: Formulae

The derivatives of the Q functions are as follows:

$$Q'_{gk}(z; A, B, g, k, c) = B(1+z^2)^k R_{gk}(z; g, k), \quad (4)$$

$$Q'_{gh}(z; A, B, g, h, c) = B \exp(hz^2/2) R_{gh}(z; g, h), \quad (5)$$

where

$$R_{gk}(z; g, k, c) = [1 + c \tanh(gz/2)] \frac{1 + (2k+1)z^2}{1+z^2} + \frac{cgz}{2 \cosh^2(gz/2)}, \quad (6)$$

$$R_{gh}(z; g, h, c) = [1 + c \tanh(gz/2)] (1 + hz^2) + \frac{cgz}{2 \cosh^2(gz/2)}. \quad (7)$$

Observe that each Q' function has the same sign and roots as the corresponding R function.

Appendix B: Range of valid parameters - theory

This appendix proves theoretical results quoted earlier about which parameter values produce valid g -and- k and g -and- h distributions.

First note that the defining functions in (2) and (3) both have the property that $Q(z; A, B, -g, k, c) = Q(z; A, B, g, k, -c)$. Therefore any behaviour produced by $c < 0$ can be replicated with $c > 0$ and a different choice of g . So for simplicity it suffices to concentrate on $c \geq 0$.

For the remainder of this appendix, distributional validity will correspond to a strictly increasing quantile function. This property is generally violated if $c > 1$, as there are two solutions to $Q(z) = A$: $z = 0$ and a solution to $1 + c \tanh(gz/2) = 0$ (The only exception is the special case of $g = 0$.) Also taking $h < 0$ or $k < -1/2$ is invalid, as in either case Q , which is continuous, has a positive gradient at $z = 0$ but limits of zero.

Finally it is shown that non-negative values of k or h produce valid distributions provided that $0 \leq c < c^* \approx 0.83$ (Rayner and MacGillivray, 2002). From Appendix A it suffices to derive the values of c such that $R(z)$ – representing either $R_{gk}(z; g, k, c)$ or $R_{gh}(z; g, h, c)$ – is guaranteed to be positive for $k \geq 0$ or $h \geq 0$. Note that $R(z)$ is a continuous function of z , and $R(0) > 0$. So a sufficient condition for validity is that no solution to $R(z) = 0$ exists. Rearranging $R(z) = 0$ using (6) and (7) gives

$$1/c = uv \operatorname{sech}^2 u + \tanh u, \quad (8)$$

where $u = -gz/2$,

$$\text{and } v = \begin{cases} \frac{1+z^2}{1+(2k+1)z^2} & (\text{g-and-}k) \\ \frac{1}{1+hz^2} & (\text{g-and-}h) \end{cases}$$

For $k \geq 0$ or $h \geq 0$, v can only take values in $(0, 1]$ with 1 attained by $z = 0$. Hence (8) gives $c > 0$ if and only if $u > 0$, and we concentrate on this case from now on. We wish to find the minimum positive solution for c . Since $1/c$ is increasing in v it suffices to concentrate on its largest value, $v = 1$. The problem reduces to minimising $(u \operatorname{sech} u + \tanh u)^{-1}$ for $u > 0$. Numerically this gives $c^* \approx 0.83$, as shown in Figure 8.

Acknowledgements

Thanks to Kieran Peel who wrote a helpful undergraduate dissertation on this topic.

Bibliography

- D. Allingham, R. A. R. King, and K. L. Mengersen. Bayesian estimation of quantile distributions. *Statistics and Computing*, 19(2):189–201, 2009. URL <https://doi.org/10.1007/s11222-008-9122-1>

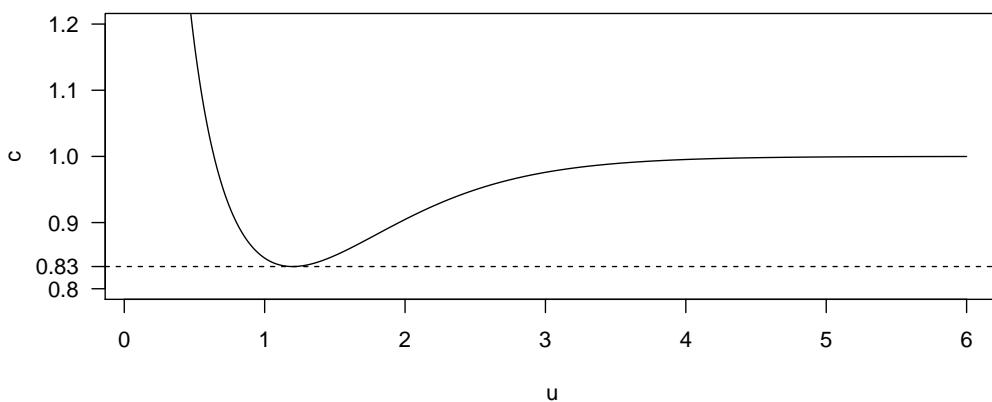


Figure 8: Solutions to (8) for $v = 1$ and $u > 0$.

9083-x. [p]

- M. G. B. Blum, M. A. Nunes, D. Prangle, and S. A. Sisson. A comparative review of dimension reduction methods in approximate bayesian computation. *Statistical Science*, 28(2):189–208, 2013. URL <https://doi.org/10.1214/12-sts406>. [p]
- Y. Croissant. *Ecdat: Data Sets for Econometrics*, 2016. URL <https://CRAN.R-project.org/package=Ecdat>. R package version 0.3-1. [p]
- K. Csilléry, O. François, and M. G. B. Blum. abc: an R package for approximate Bayesian computation (ABC). *Methods in ecology and evolution*, 3(3):475–479, 2012. URL <https://doi.org/10.1111/j.2041-210X.2011.00179.x>. [p]
- C. C. Drovandi and A. N. Pettitt. Likelihood-free Bayesian estimation of multivariate quantile distributions. *Computational Statistics & Data Analysis*, 55(9):2541–2556, 2011. URL <https://doi.org/10.1016/j.csda.2011.03.019>. [p]
- P. Fearnhead and D. Prangle. Constructing summary statistics for approximate Bayesian computation: Semi-automatic ABC. *Journal of the Royal Statistical Society, Series B*, 74:419–474, 2012. URL <https://doi.org/10.1111/j.1467-9868.2011.01010.x>. [p]
- W. Gilchrist. *Statistical modelling with quantile functions*. CRC Press, 2000. URL <https://doi.org/10.1201/9781420035919>. [p]
- H. Haario, E. Saksman, and J. Tamminen. An adaptive Metropolis algorithm. *Bernoulli*, pages 223–242, 2001. URL <https://doi.org/10.2307/3318737>. [p]
- C. Hastings, Jr., F. Mosteller, J. W. Tukey, and C. P. Winsor. Low moments for small samples: a comparative study of order statistics. *The Annals of Mathematical Statistics*, pages 413–426, 1947. URL <https://doi.org/10.1214/aoms/1177730388>. [p]
- M. Haynes and K. Mengersen. Bayesian estimation of g-and-k distributions using MCMC. *Computational Statistics*, 20(1):7–30, 2005. URL <https://doi.org/10.1007/BF02736120>. [p]
- M. A. Haynes, H. L. MacGillivray, and K. L. Mengersen. Robustness of ranking and selection rules using generalised g-and-k distributions. *Journal of Statistical Planning and Inference*, 65(1):45–66, 1997. URL [https://doi.org/10.1016/s0378-3758\(97\)00050-5](https://doi.org/10.1016/s0378-3758(97)00050-5). [p]
- F. Jabot, T. Faure, and N. Dumoulin. EasyABC: performing efficient approximate Bayesian computation sampling schemes using R. *Methods in Ecology and Evolution*, 4(7):684–687, 2013. URL <https://doi.org/10.1111/2041-210x.12050>. [p]
- J. Kiefer and J. Wolfowitz. Stochastic estimation of the maximum of a regression function. *The Annals of Mathematical Statistics*, 23(3):462–466, 1952. URL https://doi.org/10.1007/978-1-4613-8505-9_4. [p]

- N. L. Kleinman, J. C. Spall, and D. Q. Naiman. Simulation-based optimization with stochastic approximation using common random numbers. *Management Science*, 45(11):1570–1578, 1999. URL <https://doi.org/10.1287/mnsc.45.11.1570>. [p]
- H. J. Kushner and G. G. Yin. *Stochastic approximation and recursive algorithms and applications*. Springer, 2003. URL <https://doi.org/10.1007/b97441>. [p]
- P. L'Ecuyer and P. W. Glynn. Stochastic optimization by simulation: Convergence proofs for the GI/G/1 queue in steady-state. *Management Science*, 40(11):1562–1578, 1994. URL <https://doi.org/10.1287/mnsc.40.11.1562>. [p]
- J.-M. Marin, P. Pudlo, C. P. Robert, and R. J. Ryder. Approximate Bayesian computational methods. *Statistics and Computing*, 22(6):1167–1180, 2012. URL <https://doi.org/10.1007/s11222-011-9288-2>. [p]
- O. Mersmann. *microbenchmark: Accurate Timing Functions*, 2015. URL <https://CRAN.R-project.org/package=microbenchmark>. R package version 1.4-2.1. [p]
- G. W. Peters, W. Y. Chen, and R. H. Gerlach. Estimating quantile families of loss distributions for non-life insurance modelling via L-moments. *Risks*, 4(2):14, 2016. URL <https://doi.org/10.2139/ssrn.2739417>. [p]
- G. D. Rayner and H. L. MacGillivray. Numerical maximum likelihood estimation for the g-and-k and generalized g-and-h distributions. *Statistics and Computing*, 12(1):57–75, 2002. URL <https://doi.org/10.1023/A:1013120305780>. [p]
- B. Ripley. *Stochastic Simulation*. Wiley, 1987. URL <https://doi.org/10.1002/9780470316726>. [p]
- E. Saksman and M. Vihola. On the ergodicity of the adaptive Metropolis algorithm on unbounded domains. *The Annals of Applied Probability*, 20(6):2178–2203, 2010. URL <https://doi.org/10.1214/10-aap682>. [p]
- S. A. Sisson, Y. Fan, and M. Beaumont, editors. *Handbook of Approximate Bayesian Computation*. Chapman & Hall/CRC, 2017. URL <https://doi.org/10.1201/9781315117195>. [p]
- J. C. Spall. Implementation of the simultaneous perturbation algorithm for stochastic optimization. *IEEE Transactions on aerospace and electronic systems*, 34(3):817–823, 1998. URL <https://doi.org/10.1109/7.705889>. [p]
- J. W. Tukey. Modern techniques in data analysis. In *Proceedings of the NSF-Sponsored Regional Research Conference*. Southern Massachusetts University, 1977. [p]

Dennis Prangle
Department of Mathematics and Statistics
Newcastle University
NE1 7RU
UK
dennis.prangle@newcastle.ac.uk

NlinTS: An R Package For Causality Detection in Time Series

by Youssef Hmamouche

Abstract The causality is an important concept that is widely studied in the literature, and has several applications, especially when modelling dependencies within complex data, such as multivariate time series. In this article, we present a theoretical description of methods from the **NlinTS** package, and we focus on causality measures. The package contains the classical Granger causality test. To handle non-linear time series, we propose an extension of this test using an artificial neural network. The package includes an implementation of the Transfer entropy, which is also considered as a non-linear causality measure based on information theory. For discrete variables, we use the classical Shannon Transfer entropy, while for continuous variables, we adopt the k-nearest neighbors approach to estimate it.

Introduction

The study of dependencies between variables is an important step in the analysis of multi-variate time series. Not surprisingly, it can be exploited in causal discovery for financial and neuroscience datasets, in feature selection to determine the most important variables as inputs of prediction models, *etc*. Standard measures like correlation and mutual information are very used for analyzing relationships between time series. Because these measures are symmetrical, they do not provide enough information concerning the transfer of information over time from one variable to another one. Therefore, in cases where we are interested in approximating non-symmetrical dependencies between variables, causality is more adequate than correlation measures.

In the literature, two main causality measures have been well investigated in the field of time series analysis; the Granger causality test (Granger, 1980), and the Transfer entropy (Schreiber, 2000). The Granger causality is based on the principle that a variable causes another variable if it contains useful information in terms of prediction. Consequently, it is mainly linked to the idea of using of a prediction model to test the causality. The Transfer entropy in the other hand is based on information theory and has gained an increasing attention during recent years. It measures the flow of information between variables using the conditional Shannon entropy. Although these two measures seem radically different, an interesting finding has been presented in Barnett et al. (2009) showing that they are equivalent for variables that follow a normal distribution. In addition, Transfer entropy is considered as a non-linear alternative for the Granger causality, since it does not model the relationships between variables using a statistical model, instead, it is based on information theory.

This article covers a theoretical description of methods implemented in the **NlinTS** package (Hmamouche, 2020). Particularly, we focus on methods and models that are related to causality measures. This package includes the Granger causality test. To deal with non-linear dependencies between time series, we propose an non-linear extension of the Granger causality test using feed-forward neural networks. The package includes also an implementation of Transfer entropy. Two versions are provided, one for discrete variables, and the second is an estimate for continuous variables based on the k-nearest neighbors approach (Kraskov et al., 2004). Therefore, We detail the Granger causality test, the proposed non-linear Granger causality test, the VARNN (Vector Auto-Regressive Neural Network) model, since it is used in the later. Then, we represent the Transfer entropy, including the original formulation and the continuous estimation, starting by the estimate of the entropy and the mutual information, because they will be useful to understand the Transfer entropy estimator.

It is worth to mention that there are several R packages that contain an implementation of the Granger causality test, such as **vars** (Pfaff, 2008), **lmtest** (Zeileis and Hothorn, 2002). However, for Transfer entropy, especially for the continuous estimation, we found only the **RTransferEntropy** package (Simon et al., 2019). The approach used for estimating the Transfer entropy for continuous variables is based on discretization methods, by transforming continuous variables to discrete, then, applying Shannon Transfert entropy. In this paper, our approach is based on the same principle proposed in Kraskov et al. (2004) to estimate the mutual information, which inherits from the Kozachenko-Leonenko estimator of the Shannon entropy.

The organization of the paper is as follows, the two first sections are for the theoretical formulation of the causality tests and the Transfer entropy measures. The third section provides R code examples of the presented measures, illustrating the usage of the implemented methods. Finally,

the last section summarizes this paper.

The Granger causality test

The Granger causality test (Granger, 1980) is the classical method to test the causality between time series. To test if a variable X causes another variable Y , the principle of this test is to predict Y using its own history, and to predict it using its history plus the history of the variable X , and finally to evaluate the difference between these two situations to see if the added variable has some effect on the predictions of the target variable.

Formally, two VAR (p) (Vector Auto-Regressive) models are considered. The first one uses the precedent values of Y , and the second uses both passed values of X and Y in order to predict Y :

$$\text{Model}_1 \quad Y_t = \alpha_0 + \sum_{i=1}^p \alpha_i Y_{t-i} + U_t, \quad (1)$$

$$\text{Model}_2 \quad Y_t = \alpha_0 + \sum_{i=1}^p \alpha_i Y_{t-i} + \sum_{i=1}^p \beta_i X_{t-i} + U_t, \quad (2)$$

where p is the lag parameter, $[\alpha_0, \dots, \alpha_p]$ and $[\beta_0, \dots, \beta_p]$ are the parameters of the models, and U is a white noise error term.

To quantify the causality, we have to evaluate the variances of the errors of Model₁ and Model₂. In this case, the Granger causality index (GCI) can be used, and it is expressed as follows:

$$\text{GCI} = \log \left(\frac{\sigma_1^2}{\sigma_2^2} \right), \quad (3)$$

where σ_1^2 and σ_2^2 are the variances of the errors of Model₁ and Model₂ resp. In order to evaluate the statistical significance of the difference between these variances, the Fisher test can be used, where the statistic is as follows:

$$F = \frac{(RSS_1 - RSS_2) / p}{RSS_2 / (n - 2p - 1)}.$$

RSS₁ and RSS₂ are the residual sum of squares related to Model₁ and Model₂ resp., and n is the size of the lagged variables. Two hypotheses have to be considered:

- $H_0: \forall i \in \{1, \dots, p\}, \beta_i = 0,$
- $H_1: \exists i \in \{1, \dots, p\}, \beta_i \neq 0.$

H_0 is the hypothesis that X does not cause Y . Under H_0 , F follows the Fisher distribution with $(p, n - 2p - 1)$ as degrees of freedom.

A non-linear Granger causality test

Using artificial neural networks (ANNs) may be very important when computing causalities, especially for time series that change non-linearly over time. We take advantage from the characteristics of ANNs and propose an implementation of an extended version of the Granger causality test using the VARNN model. Before describing the proposed causality test, let us first present briefly the VARNN model which is also available in the package as a prediction model.

The VARNN model: Consider a training dataset that consists of a multivariate time series containing one target variable Y , and k predictor variables $\{Y_1, \dots, Y_k\}$. The VARNN (p) model is a multi-layer perceptron neural network model that takes into account the p previous values of the predictor variables and the target variable (Y) in order to predict future values of Y . We made this choice to allow for the possibility of predicting each target variable with a specific set of predictors, since target variables do not necessarily have the same predictors. First, the model reorganizes the data in a form of a supervised learning form with respect to the lag parameter. The optimization algorithm used to update the weights of the network is based on the Stochastic Gradient Descent (SGD) algorithm. The Adam algorithm can also be used to update the learning rate while using SGD (Kingma and Ba, 2015). The global function of the VARNN (p) can be written as follows:

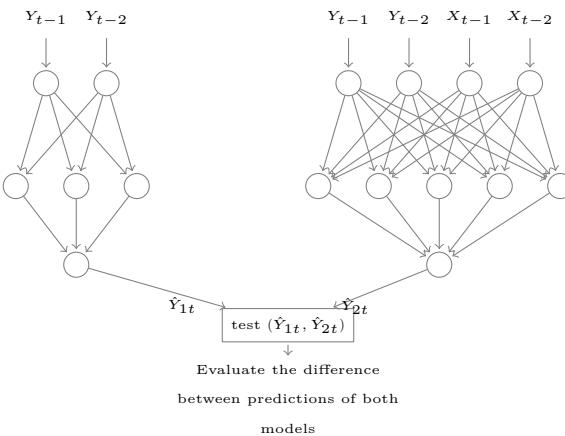


Figure 1: Illustration of the ANN model for the Granger causality test.

$$Y_t = \Psi_{nn} (Y_{t-1}, \dots, Y_{t-p}, \dots, Y_{k(t-1)}, \dots, Y_{k(t-p)}) + U_t, \quad (4)$$

where Ψ_{nn} is the network function, and U_t represents the error terms.

A causality test using the VARNN model: Consider two variables X and Y . Similarly to the Granger causality, to test the causality from X to Y , two prediction models are considered, the first takes into account the passed values of the target time series, and the second takes the passed values of the target and the predictor time series,

$$\text{Model}_1 : \quad Y_t = \Psi_{1nn} (Y_{t-1}, \dots, Y_{t-p}) + U_t, \quad (5)$$

$$\text{Model}_2 : \quad Y_t = \Psi_{2nn} (Y_{t-1}, \dots, Y_{t-p}, X_{t-1}, \dots, X_{t-p}) + U_t, \quad (6)$$

where Ψ_{1nn} and Ψ_{2nn} are the network functions of Model₁ and Model₂ resp., using the VARNN model. Then, we evaluate the difference between these two models by comparing the residual sum of squares of their errors, and the evaluation is carried out using the Fisher test to examine the null hypothesis (the hypothesis that X does not cause Y). Figure 1 shows an illustration of the used structure of the causality model.

The difference compared to the classical test, is that instead of using 2 VAR models (univariate and bivariate), two VARNN models are used. Therefore, we have to change the statistic of the Fisher test because there are more parameters in the VARNN models than in the VAR model. In this case, the statistic of test is as follows:

$$F = \frac{(RSS_1 - RSS_2) / (d_2 - d_1)}{RSS_2 / (n - d_2)},$$

where d_1 and d_2 are the number of parameters of the univariate and the bivariate model resp. They depend on the chosen structure (number of layers and of neurons).

Let us emphasize an important point about this causality. It is evident that computing causalities using ANNs may have the classical drawback of increasing the computational time. This is not exactly precise in some cases, because suppose that we have a large number of time series and we have to compute causalities between all variables. Also, suppose that relationships between variables change over time. Therefore, this implies that we need to recalculate the causalities periodically or after each change. In addition, the basic formulations of the classical causality measures (Granger causality test and Transfer entropy) are not adaptive, which means they do not make it possible to update the new values by using the old ones. In the other hand, with ANNs, the first computation of causalities may be slow compared to the Granger test or the Entropy Transfer, but if we have new observations in the time series, the model adapts more quickly thanks to the learning properties of ANNs.

Transfer entropy

Transfer entropy (Schreiber, 2000) between two time series X and Y , measures the information flow from X to Y . It was developed to overcome the main drawback of mutual information, which provides the common information between two variables (symmetric measure), but does not consider the transfer of information from one variable to the other. To avoid this problem, time delay parameters are included in the equation of the mutual information to specify the direction of information:

$$\begin{aligned} T_{X \rightarrow Y} &= \sum_{Y_t, Y_t^q, X_t^p} P(Y_t, Y_t^q, X_t^p) \log \left(\frac{P(Y_t | Y_t^q, X_t^p)}{P(Y_t | Y_t^q)} \right) \\ &= I(Y_t; X_t^p | Y_t^q), \end{aligned}$$

where $Z_t^l = (Z_{t-1}, \dots, Z_{t-l})$ for $Z = X, Y$, p, q are the time delay parameters for X and Y resp., P represents the probability, and I represents the mutual information symbol. The transfer entropy can also be seen as the difference between two conditional entropies, where in the first one, only past values of Y are used, and in the second, both X and Y are considered:

$$\begin{aligned} TE_{X \rightarrow Y} &= H(Y_t | (Y_{t-1}, \dots, Y_{t-q})) \\ &\quad - H(Y_t | (Y_{t-1}, \dots, Y_{t-q}), (X_{t-1}, \dots, X_{t-p})), \end{aligned}$$

where H represents the conditional entropy. Note that this expression resembles, in some sense, the principle of the Granger causality test which compares two prediction models.

A continuous estimation of Shannon Transfer entropy

In this section, we describe the estimation of Transfer entropy based on the k-nearest neighbors. First, we show the entropy estimator represented in Kraskov et al. (2004). Then, we show the mutual information estimator that is based on an extended formulation based on the same principal. Then, we use this approach to estimate the Transfer entropy.

Entropy estimation The basic approach for estimating the entropy of continuous variables is based on binning the data, in order to get back to the classical definition of Shannon entropy. However, more efficient approaches are proposed by estimating directly the continuous entropy:

$$H(X) = - \int p(x) \log(p(x)) dx,$$

where p represents the density function of X . One estimation of the continuous entropy of a random variable X with n realizations is the expected value of $\log(p(X))$:

$$\hat{H}(X) = -\frac{1}{n} \sum_{i=1}^n \log(\hat{p}(x_i))$$

The main point of the Kozachenko-Leonenko estimator to approximate $\log(p(x_i))$ by considering $p(x_i)$ constant in the sphere centered at x_i , with radius the distance from x_i to the k-nearest neighbors of each point. We do not show the details of the mathematical proof, but just the obtained formula:

$$\hat{H}(X) = \Gamma(n) - \Gamma(k) + \log(c) + \frac{m}{n} \sum_{i=1}^n d_i, \quad (7)$$

where Γ is the gamma function, m is the dimension of X , i.e., the number of variables, d_i is twice the distance from x_i to its k^{th} neighbor, and c is the volume of the unit ball of dimension m . To compute the distances between two points x_i and x_j , we use the max norm, $|x_i - x_j|$, therefore, $c = 1$, and $\log(c) = 0$. In the rest of the equations, for simplicity, we neglect this term.

Mutual Information estimation The mutual information between two variables X and Y having n observations can be expressed as follows:

$$I(X;Y) = H(X) + H(Y) - H(X,Y). \quad (8)$$

It is possible to adopt the Kozachenko-Leonenko approach to estimate the mutual information. In this case, we need to estimate the individual entropy of each variable and the joint entropy. For the joint entropy, it can be computed using the same way by considering the joint space spanned by X and Y . Let $z_i = (x_i, y_i)$ for $i \in [1, n]$, and d_i be the distance for z_i to its k^{th} neighbor. The estimate of the joint entropy can be expressed as follows:

$$\hat{H}(X, Y) = \Gamma(n) - \Gamma(k) + \frac{m_x + m_y}{n} \sum_{i=1}^n d_i, \quad (9)$$

where m_x and m_y are the dimensions of X and Y .

In Kraskov et al. (2004), two new methods have been proposed to improve the Kozachenko-Leonenko estimator for mutual information. The first method is based on the idea that when estimating $H(X)$ and $H(Y)$, we do not have to use the same k as used in the joint entropy, but instead, it is more precise to use the number of neighbors of each variable separately. Thus, the estimate of the individual entropy, of X for example, is the following:

$$\hat{H}(X) = \Gamma(n) - \frac{1}{n} \sum_{i=1}^n \Gamma(nx(i) + 1) + \frac{m}{n} \sum_{i=1}^n d_i, \quad (10)$$

where $nx(i)$ is the number of points where the distance from X_i is strictly less than $d_i/2$. As for Y , $\hat{H}(Y)$ is computed with the same way. Finally, based on Equations 8, 9 and 10, the mutual information estimator is as follows:

$$\hat{I}(X;Y) = \Gamma(k) + \Gamma(n) - \frac{1}{n} \sum_{i=1}^n (\Gamma(nx(i) + 1) + \Gamma(ny(i) + 1)) \quad (11)$$

Following the same method and generalizing the previous formulation to l variables $\{X_1, \dots, X_l\}$, the multivariate mutual information estimator is as follows:

$$\hat{I}(X_1, \dots, X_l) = \Gamma(k) + (l-1)\Gamma(n) - \frac{1}{n} \sum_{i=1}^n (\Gamma(n_1(i) + 1) + \dots + \Gamma(n_l(i) + 1)), \quad (12)$$

where $n_j(i)$, for $(j, i) \in [1, l] \times [1, n]$, is the number of points where the distance from the point X_{ji} is strictly less than $d_i/2$.

The motivation behind the second estimator of mutual information presented in Kraskov et al. (2004) is that the Kozachenko-Leonenko estimation of the joint entropy ($H(X, Y)$ in the bi-variate case) may be more precise than the first estimator if we consider that the density is constant in hyper-rectangles instead of hyper-cubes. Based on this remark, the second estimate of the mutual information of l variables $\{X_1, \dots, X_l\}$, with n observations, can be expressed as follows:

$$\hat{I}(X_1, \dots, X_l) = \Gamma(k) + \frac{l-1}{k} + (l-1)\Gamma(n) - \frac{1}{n} \sum_{i=1}^n (\Gamma(n_1(i) + \dots + \Gamma(n_l(i)))), \quad (13)$$

where $n_j(i)$, for $(j, i) \in [1, l] \times [1, n]$, is the number of points where the distance from the X_{ji} is less (not strictly) than $d_{ji}/2$, and d_{ji} is the distance from X_{ji} to its k^{th} neighbor.

Transfert entropy estimation Let us use the first strategy used by Kraskov for mutual information estimation to estimate the Transfer entropy. Let X and Y be two time series. The goal is to estimate the Transfer entropy from X to Y , with time delay parameters p and q resp.

$$\hat{TE}_{X \rightarrow Y} = \hat{H}(Y_t | Y_{t-1}, \dots, Y_{t-q}) - \hat{H}(Y_t | (Y_{t-1}, \dots, Y_{t-p}), (X_{t-1}, \dots, X_{t-p})). \quad (14)$$

Consider the following notations :

- $Y_t^m = \{Y_{t-1}, \dots, Y_{t-q}\}$
- $X_t^m = \{X_{t-1}, \dots, X_{t-p}\}$

- $Y_t^f = \{Y_t, Y_t^m\}$
- $X_t^f = \{X_t, X_t^m\}$
- $Z_t^m = \{Y_t^m, X_t^m\}$
- $Z_t^f = \{Y_t^f, Y_t^m, X_t^m\}$

We can rewrite then Equation 14 as follows:

$$\begin{aligned}\hat{T}E_{X \rightarrow Y} &= \hat{H}(Y_t, Y_p) - \hat{H}(Y_t|X_p, Y_p), \\ &= \hat{H}(Y_t, Y_p) - \hat{H}(Y_p) - \hat{H}(Y_t, X_p, Y_p) + \hat{H}(X_p, Y_p) \\ &= \hat{H}\left(Y_t^f\right) - \hat{H}(Y_t^m) - \hat{H}\left(Z_t^f\right) + \hat{H}(Z_t^m).\end{aligned}\quad (15)$$

The maximum joint space is defined by $Z_t^f = \{Y_t, Y_t^m, X_t^m\}$. Consider that Z_t^f contains n observations. The first step is to computes the distances d_i , i.e., the distance from the point z_i to its k^{th} neighbor, for $i \in [1, n]$. In the same way as estimating mutual information, we compute the maximal joint entropy $\hat{H}\left(Z_t^f\right)$ using the Kozachenko-Leonenko estimator, and the other terms by projecting the number of neighbors in each marginal space using the Kraskov approach:

$$\begin{aligned}\hat{H}\left(Z_t^f\right) &= -\Gamma(k) + \Gamma(n) + \frac{p+q+1}{n} \sum_{i=1}^n d_i, \\ \hat{H}\left(Y_t^f\right) &= -\frac{1}{n} \sum_{i=1}^n \Gamma(n_{y_f}(i)+1) + \Gamma(n) + \frac{p+1}{n} \sum_{i=1}^n d_i, \\ \hat{H}(Y_t^m) &= -\frac{1}{n} \sum_{i=1}^n \Gamma(n_{y_m}(i)+1) + \Gamma(n) + \frac{p}{n} \sum_{i=1}^n d_i, \\ \hat{H}(Z_t^m) &= -\frac{1}{n} \sum_{i=1}^n \Gamma(n_{z_m}(i)+1) + \Gamma(n) + \frac{p+q}{n} \sum_{i=1}^n d_i,\end{aligned}$$

where $n_{y_f}(i)$, $n_{y_m}(i)$ and $n_{z_m}(i)$ are the numbers of points where the distance from the point Y_i^f , Y_i^m , and Z_i^m resp., is strictly less than $d_i/2$, for $i \in [1, n]$. By replacing each oh these terms in Equation 15, we obtain:

$$\begin{aligned}\hat{T}E_{X \rightarrow Y} &= \Gamma(k) - \Gamma(n) - \frac{(p+1)-p-(p+q+1)+(p+q)}{n} \sum_{i=1}^n d_i \\ &\quad + \frac{1}{n} \sum_{i=1}^n (-\Gamma(n_{y_f}(i)+1) + \Gamma(n_{y_m}(i)+1) - \Gamma(n_{z_m}(i)+1)),\end{aligned}\quad (16)$$

By simplifying this expression, the Transfer entropy estimator can be expressed as follows:

$$\hat{T}E_{X \rightarrow Y} = \Gamma(k) - \Gamma(n) + \frac{1}{n} \sum_{i=1}^n (\Gamma(n_{y_m}(i)+1) - \Gamma(n_{y_f}(i)+1) - \Gamma(n_{z_m}(i)+1)). \quad (17)$$

And this is the classical Transfer entropy estimator investigated and discussed in Vicente et al. (2011); Lizier (2014); Zhu et al. (2015).

Normalizing the Transfer entropy

The values obtained by the Transfer entropy (TE) are not normalized, and practically, it is hard to quantify the causality in this case. Normalizing the values of TE between 0 and 1 simplifies the interpretation of the amounts of transferred information. For discrete data, The Transfer entropy from a variable X to a variable Y has a maximum value $H(Y_t|Y_t^m)$. Thus, the normalized TE (NTE) can be obtained by dividing TE by its maximum value:

$$NTE = \frac{\hat{H}(Y_t|Y_t^m) - \hat{H}(Y_t|Y_t^m, X_t^m)}{\hat{H}(Y_t|Y_t^m)} \quad (18)$$

In Gourévitch and Eggermont (2007), a preparation step is added to compute NTE to consider data that contain noise. It consists of subtracting first the average of TE by shuffling the variable X several times (rearranged it randomly):

$$NTE = \frac{TE_{X \rightarrow Y} - \sum_{i=1}^n TE_{X_{shuffled} \rightarrow Y}}{\hat{H}(Y_t|Y_t^m)}$$

In the package, we implemented just the first normalization (*cf.* Equation 18), because the second one depends on the way of shuffling the variable X . But it can be obtained easily by computing the NTE with the original variables, and the average of NTE with several shuffled variables of X .

Concerning continuous Transfer entropy, the term $\hat{H}(Y_t|Y_t^m)$ may be negative, which means that if we apply the same method to normalize the discrete TE, we will not obtain values in $[0, 1]$. To avoid this problem, we adopt another approach presented in Duan et al. (2013):

$$NTE = \frac{TE_{X \rightarrow Y} - \sum_{i=1}^n TE_{X_{shuffled} \rightarrow Y}}{H_0 - \hat{H}(Y_t|Y_t^m)},$$

where H_0 is the maximum entropy of Y by considering the uniform distribution, *i.e.*, $H_0 = \log(Y_{max} - Y_{min})$, and Y_{max} and Y_{min} are the maximum and the minimum values of Y .

R code examples

In this section, we demonstrate worked examples about the usage of the methods implemented in the package and discussed theoretically in the two previous sections. We use financial time series from the package `timeSeries` (Wuertz et al., 2017). We will present the classical Granger causality test, the VARNN prediction model, and the proposed non-linear Granger causality test. These functionalities are provided via `Rcpp` modules. We present also the functions associated to Transfer entropy measures, including the discrete and continuous estimate. Since other entropy measures are implemented, we will present them as well, such as the entropy and the mutual information.

The Granger causality test

The `causality.test` module is based on an `Rcpp` module. The two first arguments of the constructor of this module are two numerical vectors, (the goal is to test if the second vector causes the first one). The third argument is the lag parameter, which is an integer value. The last argument is logical (false by default) for the option of making data stationary using the Augmented Dickey-Fuller test, before performing the causality test.

```
library (timeSeries)
library (NlinTS)
data = LPP2005REC
# Construct the causality model from the second column to the first one,
# with a lag equal to 2, and without taking into account stationarity
model = causality.test (data[,1], data[,2], 2, FALSE)
```

The `causality.test` module has a `summary` method to show all the results of the test, and 3 properties: the Granger causality index; `gci` (*cf.* 2), the statistic of the test (`Ftest`), and the p-value (the probability of non causality) of the test (`pvalue`).

```
# Compute the causality index, the Ftest, and the pvalue of the test
model$summary ()
model$gci
model$Ftest
model$pvalue
```

The VARNN model

The `varmlp` module represents the implementation of the VARNN model. It is an `Rcpp` module, where the constructor takes as arguments a numerical Dataframe. Each column represents a variable, and the first column is the target variable. Note that the Dataframe may contain one column.

In this case, the model will be univariate (ARNN model). The second argument is the lag parameter, then, a numerical vector representing the size of the hidden layers of the network, then, an integer argument for the number of iterations to train the model. Other arguments with default values are available about using the bias neuron, the activation functions to use in each layer, the learning rate, and the optimization algorithm. More details about these arguments can be found in the manual of the package ([Hmamouche, 2020](#)).

```
library (timeSeries)
library (NlinTS)

# Load the data
data = LPP2005REC

# The lag parameter
lag = 1

# The training set
train_data = data[1:(nrow (data) - 1), ]

# Build and train the model
model = varmlp (train_data, 1, c(10,5), 100)
```

The `varmlp` module has 3 methods. The method named `forecast` compute predictions from an input `dataframe`, in other words, to test the model. And a method `train` update the parameters of the model from new data.

```
# Predict the last row of the data
predictions = model$forecast (train_data)

# Show the predictions
print (predictions[nrow (predictions),])

# Update the model (two observations are required at least since lag = 1)
model$train (data[nrow (data) - lag: nrow (data)])
```

The non-linear Granger causality test

Similarly to the previous test, the `nlin_causality.test` is an `Rcpp` module. The two first arguments of the constructor of this module are two numerical vectors, (the goal is to test if the second causes the first). The third argument is the lag parameter. The next two arguments are two numerical vectors representing the size of the hidden layers used in models 1 and 2, *resp*. The next argument is an integer for the number of the iterations to train the networks. Similarly to the `varmlp` model, other arguments with default values are available about the bias neuron, the activation functions, the learning rate, and the optimization algorithm. The manual of the package contain more details concerning these arguments ([Hmamouche, 2020](#)). The following is an example of using the non-linear causality test:

```
library (timeSeries)
library (NlinTS)
data = LPP2005REC
# Build and train the model
model = nlin_causality.test (data[,1], data[,2], 2, c(2), c(4))
```

The `nlin_causality.test` module returns the same values as the `causality.test`; a `summary` method to show all the results of the test, and 3 properties; the Granger causality index (`gci`), the statistic of the test (`Ftest`), and the p-value of the test (`pvalue`).

```
# Compute the causality index, the Ftest, and the pvalue of the test
model$summary ()
model$gci
model$Ftest
model$pvalue
```

The discrete entropy

The function `entropy_disc` permits to compute the Shannon entropy, where the first argument is a discrete vector, and the second argument is the logarithm function to use (\log_2 by default):

```
library (NlinTS)
# The entropy of an integer vector
print (entropy_disc (c(3,2,4,4,3)))
```

The continuous estimation of the entropy

The function `entropy_disc` permits to compute the continuous estimation of Shannon entropy, where the first argument is a numerical vector, and the second argument is the number of neighbors (see 4.1):

```
library (timeSeries)
library (NlinTS)
# Load data
data = LPP2005REC
# The entropy of the first column with k = 3
print (entropy_cont (data[,1], 3))
```

The discrete mutual information

The function `mi_disc` permits to compute the Shannon multivariate mutual information, where the first argument is an integer dataframe, and the second argument is the logarithm function to use (\log_2 by default):

```
library (NlinTS)
# Construct an integer dataframe with 2 columns
df = data.frame (c(3,2,4,4,3), c(1,4,4,3,3))
# The mutual information between columns of df
mi = mi_disc (df)
print (mi)
```

The continuous estimation of the mutual information

The function `mi_cont` permits to compute the continuous estimate of the mutual information between two variables. The two first arguments are two vectors, and the third argument is the number of neighbors (see 4.1):

```
library (timeSeries)
library (NlinTS)
# Load data
data = LPP2005REC
# The mutual information between of the two first columns of the data with k = 3
print (mi_cont (data[,1], data[,2], 3))
```

The discrete Transfer entropy

The function associated to the discrete TE is named `te_disc`. The two first arguments are two integer vectors. Here we allow the two time series to have different lag parameters. Therefore, the second two arguments are the lag parameters associated to the first and the second arguments *resp.* The next argument indicates the logarithm function to use (\log_2 by default). The last argument is logical for the option of normalizing the value of TE, with a false value by default. The `te_disc` function returns the value of Transfer entropy from the second variable to the first variable:

```
library (NlinTS)
# The transfer entropy between two integer vectors with lag = 1 to 1
te = te_disc (c(3,2,4,4,3), c(1,4,4,3,3), 1, 1)
print (te)
```

The continuous estimation of the Transfer entropy

The associated function is named `te_cont`. The two first arguments are two vectors. Then, the second two arguments are the associated lag parameters for the first and the second arguments `resp`. The fifth argument is the number of neighbors. The last argument is logical for the option of normalizing the value of TE, with a false value by default. The `te_cont` function returns the value of Transfer entropy from the second variable to the first one:

```
library (timeSeries)
library (NlinTS)
# Load data
data = LPP2005REC
# The transfer entropy between two columns with lag = 1 and k = 3
te = te_cont (data[,1], data[,2], 1, 1, 3)
print (te)
```

Conclusion

In this paper, we have presented methods of our `NlinTS` package for computing causalities in time series. We have considered two main measures well studied in the literature, the Granger causality test and the Transfer entropy. The Transfer entropy is originally formulated for discrete variables. For continuous variables, we adopted a k-nearest neighbors estimation based on the same strategy used to estimate the Mutual Information in Kraskov et al. (2004). To deal with non-linear time series, we have proposed another causality measure as an extension of the Granger causality test using an artificial neural network. Finally, we showed examples for the usage of these methods.

Bibliography

- L. Barnett, A. B. Barrett, and A. K. Seth. Granger causality and transfer entropy are equivalent for gaussian variables. *Phys. Rev. Lett.*, 103:238701, Dec 2009. URL <https://doi.org/10.1103/PhysRevLett.103.238701>. [p]
- P. Duan, F. Yang, T. Chen, and S. L. Shah. Direct causality detection via the transfer entropy approach. *IEEE Transactions on Control Systems Technology*, 21(6):2052–2066, Nov 2013. ISSN 1063-6536. URL <https://doi.org/10.1109/TCST.2012.2233476>. [p]
- B. Gourévitch and J. J. Eggermont. Evaluating Information Transfer Between Auditory Cortical Neurons. *Journal of Neurophysiology*, 97(3):2533–2543, Mar. 2007. ISSN 0022-3077. URL <https://doi.org/10.1152/jn.01106.2006>. [p]
- C. W. J. Granger. Testing for causality. *Journal of Economic Dynamics and Control*, 2:329–352, Jan. 1980. ISSN 0165-1889. URL [https://doi.org/10.1016/0165-1889\(80\)90069-X](https://doi.org/10.1016/0165-1889(80)90069-X). [p]
- Y. Hmamouche. *NlinTS: Models for Non Linear Causality Detection in Time Series*, 2020. URL <https://CRAN.R-project.org/package=NlinTS>. R package version 1.4.2. [p]
- D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In Y. Bengio and Y. LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL <http://arxiv.org/abs/1412.6980>. [p]
- A. Kraskov, H. Stögbauer, and P. Grassberger. Estimating mutual information. *Phys. Rev. E*, 69: 066138, Jun 2004. URL <https://doi.org/10.1103/PhysRevE.69.066138>. [p]
- J. T. Lizier. Jidt: An information-theoretic toolkit for studying the dynamics of complex systems. *Frontiers in Robotics and AI*, 1:11, 2014. ISSN 2296-9144. URL <https://doi.org/10.3389/frobt.2014.00011>. [p]
- B. Pfaff. Var, svar and svec models: Implementation within r package vars. *Journal of Statistical Software, Articles*, 27(4):1–32, 2008. ISSN 1548-7660. URL <https://doi.org/10.18637/jss.v027.i04>. [p]
- T. Schreiber. Measuring Information Transfer. *Physical Review Letters*, 85(2):461–464, July 2000. URL <https://doi.org/10.1103/PhysRevLett.85.461>. [p]

- B. Simon, D. Thomas, P. Franziska J., and Z. David J. Rtransferentropy —quantifying information flow between different time series using effective transfer entropy. *SoftwareX*, 10(100265):1–9, 2019. URL <https://doi.org/10.1016/j.softx.2019.100265>. [p]
- R. Vicente, M. Wibral, M. Lindner, and G. Pipa. Transfer entropy—a model-free measure of effective connectivity for the neurosciences. *Journal of Computational Neuroscience*, 30(1):45–67, Feb. 2011. ISSN 1573-6873. URL <https://doi.org/10.1007/s10827-010-0262-3>. [p]
- D. Wuertz, T. Setz, and Y. Chalabi. *timeSeries: Rmetrics - Financial Time Series Objects*, 2017. URL <https://CRAN.R-project.org/package=timeSeries>. R package version 3042.102. [p]
- A. Zeileis and T. Hothorn. Diagnostic checking in regression relationships. *R News*, 2(3):7–10, 2002. URL <https://CRAN.R-project.org/doc/Rnews/>. [p]
- J. Zhu, J.-J. Bellanger, H. Shu, and R. Le Bouquin Jeannès. Contribution to Transfer Entropy Estimation via the k-Nearest-Neighbors Approach. *Entropy*, 17(6):4173–4201, June 2015. URL <https://doi.org/10.3390/e17064173>. [p]

Youssef Hmamouche

Aix Marseille Université, Université de Toulon, CNRS, LIS, UMR7020, Marseille, France

Aix Marseille Université, CNRS, LPL, UMR7309, Aix-en-Provence, France

youssef.hmamouche@lis-lab.fr

mudfold: An R Package for Nonparametric IRT Modelling of Unfolding Processes

by Spyros E. Balafas, Wim P. Krijnen, Wendy J. Post and Ernst C. Wit

Abstract Item response theory (IRT) models for unfolding processes use the responses of individuals to attitudinal tests or questionnaires in order to infer item and person parameters located on a latent continuum. Parametric models in this class use parametric functions to model the response process, which in practice can be restrictive. MUDFOLD (Multiple UniDimensional unFOLDing) can be used to obtain estimates of person and item ranks without imposing strict parametric assumptions on the item response functions (IRFs). This paper describes the implementation of the MUDFOLD method for binary preferential-choice data in the R package **mudfold**. The latter incorporates estimation, visualization, and simulation methods in order to provide R users with utilities for nonparametric analysis of attitudinal questionnaire data. After a brief introduction in IRT, we provide the methodological framework implemented in the package. A description of the available functions is followed by practical examples and suggestions on how this method can be used even outside the field of psychometrics.

Introduction

In this paper we introduce the R package **mudfold** (Balafas et al., 2019), which implements the non-parametric IRT model for unfolding processes MUDFOLD. The latter, was developed by Van Schuur (1984) and later extended by Post (1992) and Post and Snijders (1993). IRT models have been designed to measure mental properties, also called latent traits. These models have been used in the statistical analysis of categorical data obtained by the direct responses of individuals to tests and questionnaires. Two response processes that result in different classes of IRT models can be distinguished. The cumulative (also called monotone) processes and the unfolding (also called proximity) processes in the IRT framework differ in the way that they model the probability of a positive response to a question from a person as a function of the latent trait, which is termed as item response function (IRF).

Cumulative IRT models also known as Rasch models (Rasch, 1961), assume that the IRF is a monotonically increasing function. That is, the higher the latent trait value for a person, the higher the probability of a positive response to an item (Sijtsma and Junker, 2006). This assumption makes cumulative models suitable for testing purposes where latent traits such as knowledge or abilities need to be measured. The unfolding models consider nonmonotone IRFs. These models originate from the work of Thurstone (1927, 1928) and have been formalized by Coombs (1964) in his deterministic unfolding model. In unfolding IRT the IRF is assumed to be a unimodal (single 'peak') function of the distance between the person and item locations on a hypothesized latent continuum. Unimodal IRFs imply that the closer an individual is located to an item the more likely is that he responds positively to this item (Hoijtink, 2005). Unfolding models can be used when one is interested to measure bipolar latent traits such as preferences, choices, or political ideology, which are generally termed as attitudes (Andrich, 1997). Such type of latent traits when they are analyzed using monotone IRT models usually result in a multidimensional solution. In this sense, unfolding models are more general than the cumulative IRT models (Stark et al., 2006; Chernyshenko et al., 2007) and can be seen as a form of quadratic factor analysis (Maraun and Rossi, 2001).

Parametric IRT (PIRT) models for unfolding processes exist for dichotomous items (Hoijtink, 1991; Andrich and Luo, 1993; Maydeu-Olivares et al., 2006), polytomous items (Roberts and Laughlin, 1996; Luo, 2001) as well as for bounded continuously scored items (Noel, 2014). Typically, estimation in PIRT models exploits maximum likelihood methods like the marginal likelihood (e.g. Roberts et al., 2000) or the joint likelihood (e.g. Luo et al., 1998), which are optimized using the expectation-maximization (EM) or Newton type of algorithms. Unfolding PIRT models that infer model parameters by adopting Bayesian Markov Chain Monte Carlo (MCMC) algorithms (Johnson and Junker, 2003; Roberts and Thompson, 2011; Liu and Wang, 2019; Lee et al., 2019) are also available. PIRT models however, make explicit parametric assumptions for the IRFs, which in practice can restrict measurement by eliminating items with different functional properties.

Nonparametric IRT (NIRT) models do not assume any parametric form for the IRFs but instead introduce order restrictions (Sijtsma, 2005). These models have been used to construct or evaluate scales that measure among others, internet gaming disorder (Finserås et al., 2019), pedal sensory

loss (Rinkel et al., 2019), partisan political preferences (Hänggli, 2020), and relative exposure to soft versus hard news (Boukes and Boomgaarden, 2015). The first NIRT model was proposed by Mokken (1971) for monotone processes. His ideas were used for the unfolding paradigm by Van Schuur (1984) who designed MUDFOLD as the unfolding variant of Mokken's model. MUDFOLD was extended by Van Schuur (1992) for polytomous items and Post (1992) and Post and Snijders (1993) derived testable properties for nonparametric unfolding models that were adopted in MUDFOLD. Usually, NIRT methods employ heuristic item selection algorithms that first rank the items on the latent scale and then use these ranks to estimate individual locations on the latent continuum. Such estimates for individuals' ideal-points in unfolding NIRT have been introduced by Van Schuur (1988) and later by Johnson (2006). NIRT approaches can be used for exploratory purposes, preliminary to PIRT models, or in cases where parametric functions do not fit the data.

IRT models can be fitted by means of psychometric software implemented in R (Choi and Asilkalkan, 2019), which can be downloaded from the Comprehensive R Archive Network (CRAN)¹. An overview of the R packages suitable for IRT modelling can be found at the dedicated task view *Psychometrics*. PIRT models for unfolding where the latent trait is unidimensional, such as the graded unfolding model (GUM) (Roberts and Laughlin, 1996) and the generalized graded unfolding model (GGUM) (Roberts et al., 2000) can be fitted by the R package **GGUM** (Tendeiro and Castro-Alvarez, 2018). Sub-models in the GGUM class are also available into the Windows software **GGUM2004** (Roberts et al., 2006). A large variety of unfolding models for unidimensional and multidimensional latent traits can be defined and fitted to data with the R package **mirt** (Chalmers, 2012). To our knowledge, software that fits nonparametric IRT in the unfolding class of models (analogous to the **mokken** package (Van der Ark, 2007, 2012) in the cumulative class) is not yet available in R.

In order to fill this gap, we have developed the R package **mudfold**. The main function of the package implements item selection algorithm of Van Schuur (1984) for scaling the items on a unidimensional scale. Scale quality is assessed using several diagnostics such as, scalability coefficients similar to the homogeneity coefficients of Loevinger (1948), statistics proposed by Post (1992), and newly developed tests. Uncertainty for the goodness-of-fit measures is quantified using nonparametric bootstrap (Efron et al., 1979) from the R package **boot** (Canty and Ripley, 2017). Missing values can be treated using multiple multivariate imputation by chained equations (MICE, Buuren et al., 2006), which is implemented in the R package **mice** (van Buuren and Groothuis-Oudshoorn, 2011). Estimates for the person locations derived from Van Schuur (1988) and Johnson (2006) are available to the user of the package. Generally, the MUDFOLD algorithm is suitable for studies where there are no restrictions on the number of items that a person can "pick". Besides these *pick-any-out-of-N* study designs, sometimes individuals are restricted to select a prespecified number of items, i.e. *pick-K-out-of-N*. The latter design, due to the violation of independence does not respect the IRT assumptions. However, our package is also able to deal with such situations.

Methodology

Consider a sample of n individuals randomly selected from a population of interest in order to take a behavioral test. Participants indexed by i , $i = 1, 2, \dots, n$ are asked to state if they do agree or do not with each of $j = 1, 2, \dots, N$ statements (i.e. items) towards a unidimensional attitude θ that we intend to measure. Let X_{ij} be random variables associated with the 0, 1 response of subject i on item j . We will denote the response of individual i on item j as X_{ij} and x_{ij} its realization.

Subsequently, we can define the IRF for an item j as a function of θ . That is, the probability of positive endorsement of item j from individual i with latent parameter θ_i we write $P_j(\theta_i) = P(X_{ij} = 1|\theta_i)$. In PIRT models for unfolding, $P_j(\theta_i)$ is a parametric unimodal function of the proximity between the subject parameter θ_i and the item parameter β_j . NIRT unfolding models avoid to impose strict functional assumptions on the IRFs. In the latter case, the focus is on ordering the items on a unidimensional continuum. The item ranks are then used as measurement scale to calculate person specific parameters (ideal-points) on the latent continuum.

Assumptions of the nonparametric unfolding IRT model

In unidimensional IRT models, *unidimensionality* of the latent trait, and *local independence* of the responses are common assumptions. However, the usual assumption of *monotonicity* that we meet in the cumulative IRT models, needs modification in the unfolding IRT where *unimodal* shaped IRFs are considered. For obtaining diagnostic properties for the nonparametric unfolding model,

¹URL: <http://CRAN.R-project.org>

Post and Snijders (1993) proposed two additional assumptions for the IRFs. The assumptions of the nonparametric unfolding model are:

- A1.** *Unidimensionality* (UD): There exists a unidimensional latent variable $\theta \in \mathbb{R}$ on which individuals and items are scaled.
- A2.** *Local Independence* (LI): The responses of individuals on distinct items are independent given the latent parameter θ , i.e. the joint conditional probability of N responses simplifies into the likelihood form,

$$P(\mathbf{X} = \mathbf{x} \mid \theta = \theta_0) = \prod_{j=1}^N P_j(\theta_0)^{x_j} [1 - P_j(\theta_0)]^{1-x_j}.$$

- A3.** *Unimodality* (UM): For every item j , $P_j(\theta)$ is a weakly *unimodal* function of θ .

For the sake of clarity, a function $P_j(\theta) : \mathbb{R} \rightarrow \mathbb{R}$, is weakly unimodal if there exists a $\beta_j \in (-\infty, +\infty)$ such that, $P_j(\theta)$ is non decreasing for all $\theta \leq \beta_j$ and non increasing for all $\theta \geq \beta_j$. The location parameter β_j for the j th item is the value of the latent trait for which the IRF $P_j(\theta)$ reaches its maximum (or the midpoint of the interval where $P_j(\theta)$ is maximum when β_j is not unique).

- A4.** *Stochastic Ordering* (SO): For any probability distribution $G(\theta)$ of latent trait values and any value θ_0 on the latent scale, $P_G(\theta > \theta_0 | X_j = 1)$ is nondecreasing function of j for all j such that $p_j(x) > 0$.

Given the item ordering this assumption is equivalent to two properties for the IRFs. First, given that a single item is chosen, the posterior densities g of θ have a monotone likelihood ratio (MLR) in θ , and second, the IRFs have a monotone traceline ratio (MTR). The next assumption concerns only unfolding models and is not applicable for cumulative IRT.

- A5.** *Manifest unimodality* (MUM): For any probability distribution $G(\theta)$ of latent trait values, and for any values $\theta_1 < \theta_2$, the posterior probability $P_G(\theta_1 < \theta < \theta_2 | X_j = 1)$ is a weakly unimodal function of j .

Assumption **A1** implies that there exist only one latent trait that explains the responses of persons on the items. Assumption **A2** is mathematically convenient since it reduces the likelihood to a simple product and implies that given the latent trait value no other information on the other items is relevant to predict the responses to a particular item. The next assumption concerns the conditional distribution of each item given the latent trait. The unimodality assumption that is described in **A3** restricts the IRFs to have a single-peak shape without imposing any explicit functional form. If **A3** holds for all the IRFs then we can order the items on the unidimensional continuum based on their location parameter β_j such that $\beta_1 \leq \beta_2 \leq \dots \leq \beta_N$. The set of assumptions **A1-A3** is the core in unfolding IRT models.

Additionally, two assumptions are needed about the individuals $\{i \mid i = 1, \dots, n\}$ and the distribution G of their latent trait values $\{\theta_i \mid i = 1, \dots, n\}$ in order to obtain testable properties for the nonparametric unfolding model (Post and Snijders, 1993). Assumption **A4** is analogous to the invariant item ordering (IIO) assumption in the monotone IRT models and implies that the posterior distribution of θ given a positive response to an item located at β_j is stochastically ordered by the location β_j (Johnson, 2006). In simple words, **A4** assumes that an individual who responds positively to an item with higher rank should have a larger latent trait than those individuals who respond positively to a low-rank item. For example, if a person responds positively to an item that is considered politically conservative, then this person is more likely to be a conservative compared to a person who responded positively to a liberal statement. Despite the fact that this assumption seems intuitive, not all parametric unfolding models require this additional assumption. Assumption **A5** suggests that individual i who endorses item j has a latent trait value θ_i that is most likely close to item location β_j and less likely either much lower or much higher on the latent scale than that. Post (1992) shows that the measurement assumptions **A4-A5** are related to the mathematical property of total positivity of order 2 (TP_2) (Karlin, 1968). In addition, if the IRFs $P_j(\theta)$ are positive for all j , then these assumptions hold if and only if the IRFs satisfy the property of TP_3 .

Errors and scalability coefficients

PIRT approaches use well defined IRFs that parametrize explicitly persons and items on some known parameter space. Estimates of the parameters can be obtained using suitable frequentist or Bayesian methods and the fit of the model to the data is assessed using goodness-of-fit indices. Contrarily, in NIRT modelling the functional form of the IRF is unknown and alternative estimation methods are needed (Mokken, 1997).

Models in the NIRT class, typically employ item selection algorithms that construct ordinal measurement scales for persons by iteratively maximizing some scalability measure upon the items. The resulting scales are then used to locate the individuals on the latent continuum based on their responses. Usually, these item selection algorithms are bottom-up methods that are divided into two parts. In the first part the algorithms seek to find the best minimal scale, that is a minimal set of items that meets certain scalability requirements. The best minimal scale is the starting point for the second part of the scaling procedure, where it is extended iteratively by adding in each step the item that best fulfills the prespecified scalability criteria.

As in other NIRT models, MUDFOLD adopts a two step item selection algorithm that identifies the unique rank order for a maximal (sub) set of items. In this algorithm, scalability coefficients analogous to the ones defined by [Mokken \(1971\)](#) are used as tests for the goodness-of-fit. Mokken's coefficients are similar to the H coefficients proposed by [Loevinger \(1948\)](#), which were defined on the basis of violation probabilities of the deterministic cumulative model (see [Guttman, 1944](#)) for ordered item pairs. In the same line, the scalability coefficients in MUDFOLD are defined on the basis of violation probabilities of the deterministic unfolding model of [Coombs \(1964\)](#) for triples of items. MUDFOLD's scalability coefficients in a triple of items compare the number of errors observed (i.e. the number of {1, 0, 1} responses, which falsify the Coombsian model) with the number of errors that we would expect if the items were statistically independent. A triple of items is a permutation (ordering) of three distinct items.

Observed errors (O) in an ordered triple of items (h, l, k) with h, l, k distinct elements of the set $\{1, 2, \dots, N\}$, is the frequency of {1, 0, 1} responses over all individuals. The observed errors can be calculated by $O_{hlk} = \sum_{i=1}^n x_{ih} (1 - x_{il}) x_{ik}$ where x_i is the realization of random variable X_i , and $x_i = 1$ if the i^{th} individual responds positively on item (.) otherwise $x_i = 0$. It can be seen that the number of observed errors for three items stays invariant for the permutations (h, l, k) and (k, l, h) for any $h \neq l \neq k \neq h$ in the integer set $\{1, 2, \dots, N\}$.

Expected errors (EO) in an ordered item triple (h, l, k) under random ordering is the expected frequency of {1, 0, 1} responses if the items h , l , and k were statistically independent multiplied by the sample size, $EO_{hlk} = p(h) (1 - p(l)) p(k) n$. We can estimate $p(j)$ for item j $p(j) = \sum_{i=1}^n \frac{x_{ij}}{n}$ as the relative frequency for item j .

Scalability coefficient (H) for any ordered item triple (h, l, k) , is defined as the value obtained if we subtract from unity the ratio of observed errors over the expected errors for this triple,

$$H_{hlk} = 1 - \frac{O_{hlk}}{EO_{hlk}}, \forall h, l, k \in \{1, 2, \dots, N\}. \quad (1)$$

Using the scalability coefficients for triples, we can extend the notion of scalability for a scale s consisting of m items, where $3 < m \leq N$ and for an item $j \in s$. The H coefficient for an item $j \in s$, $j = 1, 2, \dots, m$ is given by,

$$H_j(s) = 1 - \frac{\sum_{(h,l,k) \in T_j(s)} O_{hlk}}{\sum_{(h,l,k) \in T_j(s)} EO_{hlk}}, \quad (2)$$

where $T_j(s) = \{(s_h, s_l, s_k) \mid s_h < s_l < s_k : j \in \{s_h, s_l, s_k\}\}$ is the set of all item triples (with respect to the item order), that include item j .

Given that the m items constituting the scale are ordered, we are able to calculate the H coefficient for the total scale s by summing the observed errors and the expected errors for all $\frac{m!}{3!(m-3)!}$ triples of items of s and calculate their error ratio. If we subtract the obtained number from the unity results in a total scalability measure,

$$H_{\text{total}}(s) = 1 - \frac{\sum_{(h,l,k) \in T(s)} O_{hlk}}{\sum_{(h,l,k) \in T(s)} EO_{hlk}}, \quad (3)$$

where $T(s) = \{(s_h, s_l, s_k) \mid s_h < s_l < s_k\}$ is the set of all item triples for a given scale s .

Perfect fit of the scale to the data yields a scalability coefficient value of $H_{\text{total}}(s) = 1$. The latter means that no error patterns are observed in this scale. Likewise, $H_{\text{total}}(s) = 0$ implies that the number of observed errors is equal to what you would have expected for a random ordering. Values around 0.5 suggest a moderate unfolding scale. Calculating the triple scalability coefficients for all the items is the first step in the construction of a MUDFOLD scale.

We will demonstrate how the H coefficients for triples are calculated using the dataset **ANDRICH** that comes with the **mudfold** package in R data format. The dataset contains the binary responses of $n = 54$ students on $N = 8$ statements towards capital punishment. This attitudinal test have been constructed by [Andrich \(1988\)](#) in order to measure attitudes towards capital punishment.

Calculating scalability coefficients for the ANDRICH data. We can install and subsequently load the package and the data into the R environment.

```
## Install and load the mudfold package and the ANDRICH data
install.packages("mudfold")
library(mudfold)
data("ANDRICH")
N <- ncol(ANDRICH) # number of items
n <- nrow(ANDRICH) # number of persons
item_names <- colnames(ANDRICH) # item names
```

Functions for calculating the observed errors, expected errors, and H coefficients for each possible item triple are available internally in the **mudfold** package. These functions can be accessed by the `:::` operator. For the ANDRICH data the H coefficients for triples can be calculated as follows.

```
experr <- mudfold:::Err_exp(ANDRICH) # errors expected
obserr <- mudfold:::Err_obs(ANDRICH) # errors observed
hcoefft <- 1 - (obserr / experr) # H coefficients
```

Generally, there exist N^3 item permutations of length three with repetitions that can be obtained from N items. Thus, the corresponding H coefficients of each possible item permutation of length three can be stored into a three way array with dimension $N \times N \times N$. In the ANDRICH data example, the scalability coefficients for the item permutations of length three are stored into three-way array with dimension $8 \times 8 \times 8$. It can be seen that the H coefficients for symmetric permutations stay invariant and we demonstrate this feature below. Consider the ordered triple of items (*HIDEOUS*, *DONTBELIEV*, *DETERRENT*) and its symmetric permutation (*DETERRENT*, *DONTBELIEV*, *HIDEOUS*).

```
triple_HDODE <- matrix(c("HIDEOUS", "DONTBELIEV", "DETERRENT"), ncol = 3)
triple_DEDOH <- matrix(rev(triple_HDODE), ncol = 3)
```

If we compare the H coefficients of these two (symmetric) triples we will see that they coincide.

```
## Compare H coefficients
hcoefft[triple_HDODE] == hcoefft[triple_DEDOH]
```

The H_{hlk} coefficients form the basis in order to calculate the scalability coefficients for items and scales. The item selection algorithm implemented in the package runs in two steps and scalability criteria are used in both steps.

Scale construction

In the first step of the item selection algorithm, a search in order to find the best triple of items is conducted. A lower bound λ_1 that controls the scalability properties of the best triple can be specified by the user (default value is $\lambda_1 = 0.3$). The value of λ_1 is used as a threshold to determine if the triple is good enough to continue the scaling process. Larger values of λ_1 lead to more strict criteria while lower values of λ_1 relax these criteria.

In its second step, the item selection algorithm extends the best elementary scale repeatedly until no more items fulfill its scalability criteria. A second threshold $\lambda_2 = 0$ is explicitly used in the first criterion of this step. This threshold controls the scalability properties of the triples containing a candidate item in the scale extension procedure. As for λ_1 , larger values of λ_2 lead to more strict scalability requirements, while, lower values relax these requirements.

Step 1: search for the best unique triple.

The search for the optimal item triple in the first step requires the calculation of the scalability coefficients for every possible permutation of length 3 that can be obtained from N starting items.

Among the set of all permutations of length three we seek to find those that fulfill certain scalability criteria and we call this set of permutations unique triples. Unique triples is a

finite set containing all (h, l, k) with $h, l, k \in \{1, 2, \dots, N\}$, and $h \neq l \neq k \neq h$ for which only one of their permutations (out of three possible) presents a positive H_{hlk} coefficient i.e.

$$H_{hlk} > 0, \quad H_{hkl} < 0, \quad H_{lkh} < 0.$$

This guarantees that triples in the set of unique triples are “uniquely” represented on the latent dimension, i.e. are scalable together in only one permutation besides the reverse permutation. From the set of unique triples, the triple (h, l, k) that has the maximum H_{hlk} is called the best unique triple and it will be selected as the best starting scale if its scalability coefficient is positive and greater than a specified lower bound λ_1 . If more than one triples fulfill the requirements for being the best unique triple it can be shown that all of them will converge to same solution in the second step.

If the set of unique triples is empty, the algorithm stops automatically without proceeding in the second step. The same holds also in the case in which unique triples exist but their scalability coefficient is lower that the bound specified by the user.

First step: search for best minimal scale in the ANDRICH data. Here we describe how the main function of the **mudfold** package searches for the best minimal unfolding scale in the first step of the implemented algorithm. After we calculated the observed errors, the expected errors, and the scalability coefficients for each triple of items in the ANDRICH dataset, we need to determine the optimal triple for the first step of MUDFOLD’s item selection algorithm. The triples of items in the order (h, l, k) for the ANDRICH data can be obtained with the `combinations()` function from the R package **gtools** (Warnes et al., 2015). These combinations are then permuted twice to yield the orderings (h, k, l) and (l, h, k) respectively.

```
## Install and load the library "gtools"
install.packages(gtools)
library(gtools)

## Obtain item permutations (h,l,k), (h,k,l), and (l,h,k)
perm1 <- combinations(N, 3, item_names, set = FALSE)
perm2 <- perm1[, c(1,3,2)]
perm3 <- perm1[, c(2,1,3)]
```

The set of unique triples can then be obtained.

```
## Find the set of unique triples.
unq <- rbind(perm1[(hcoeft[perm1] > 0 & hcoeft[perm2] < 0 & hcoeft[perm3] < 0), ],
              perm2[(hcoeft[perm1] < 0 & hcoeft[perm2] > 0 & hcoeft[perm3] < 0), ],
              perm3[(hcoeft[perm1] < 0 & hcoeft[perm2] < 0 & hcoeft[perm3] > 0), ])
```

The set of unique triples in the ANDRICH data example contains sixteen item triples. With the command `hcoeft[unq]` we can see that all except one of the triples show H_{hlk} coefficients greater than the lower bound. The ordered triple of items (*INEFFECTIV, DONTBELIEV, DETERRENT*) is selected as the best starting scale with a maximum scalability coefficient of 0.853 which is indeed larger than λ_1 . This triple will be extended repeatedly in the second step of the algorithm. In each iteration one from the remaining ones is added to the scale in a specific position if certain scalability requirements are met.

Step 2: extending the best starting scale

Given the best unique triple obtained in the first step of the algorithm, in the second step of the item selection process the algorithm investigates repeatedly the remaining $N - 3$ items to find the best fourth, fifth, etc to add to the scale. In each iteration of this step, all the possible scales that contain one of the remaining items in every possible position are investigated to choose the most appropriate one.

For a scale consisting of m items, $(3 \leq m \leq N - 1)$ we intend to find one of the remaining $N - m$ items to add in the scale. For the $(m + 1)^{th}$ item there exist $m + 1$ possible scale positions that have to be investigated with respect to their scalability properties. In each iteration of the MUDFOLD scaling algorithm, the number of candidate scales under investigation is $(N - m)(m + 1)$.

In order to determine the $(m + 1)^{th}$ best fitting item we test three criteria. The first criterion uses an explicit value λ_2 (by default $\lambda_2 = 0$) as a lower bound for the scalability coefficients. The scalability criteria in the second step are :

1. All the $\binom{m}{2}$ item triples in the scale (with respect to the item order), containing the candidate item must have H_{hik} coefficient greater than λ_2 .
2. If more than one item fulfills the first criterion, then the item with the minimum number of possible scale positions is chosen.
3. The scalability coefficient $H_j(s)$ of the selected item has to be higher than λ_1 .

It can be the case that more than one scales fulfill these criteria. In such instances, the algorithm continues by choosing the scale that includes the most uniquely represented item and shows the minimum number of expected errors. The scale extension process continues as long as the scalability criteria described above are fulfilled.

Second step: scale extension for the ANDRICH data For the ANDRICH data, after the first step of the item selection process where we obtained the best unique triple, the remaining five items can still be added to the scale.

```
BestUnique <- unq[which.max(hcoeft[unq]), ] # Best unique triple
ALLitems <- colnames(ANDRICH)
Remaining <- ALLitems[!ALLitems %in% BestUnique] # Remaining items
```

Next, an iterative procedure needs to be defined for the second, scale extension step of the MUDFOLD algorithm. Adding one item in each repetition implies that a maximum of $N - 3 = 5$ iterations can take place if all items fit in a MUDFOLD scale. In each iteration we construct the scales to be evaluated where each scale contains one of the remaining items in a specific position.

For example, in the first iteration of the scale extension step for the ANDRICH dataset, all the scales that need to be assessed can be constructed as follows. First we need to consider all the possible positions where a new item can be added. The possible positions depend on the length of the existing scale. At this point, since the scale consists of three items there exist four possible positions where a new item can be added.

```
## Create indices to be used in constructing scales
lb <- length(BestUnique) # length of best unique triple
lr <- length(Remaining) # number of remaining items to add in the scale

## create all possible positions where each new item from Remaining
## can be added in the scale
index_rep <- rep(seq(1, (lb+1)), lr) - 1 # possible positions
index_irp <- rep(Remaining, each = lb+1) # item for each position
```

After we define all the possible positions for new items, each item is added in every position and results in a different scale to be assessed.

```
## Create all possible scales by adding each item in Remaining
## to every possible position of BestUnique
ALLscales <- lapply(1:length(index_rep),
  function(i) append(BestUnique, index_irp[i], after = index_rep[i] ))
```

Each of these scales will be judged in terms of its scalability properties. For instance, let us consider the first scale that is constructed in the first iteration of the scale extension step in the ANDRICH data.

```
ExampleScale <- ALLscales[[1]]
ExampleScale
# "HIDEOUS"      "INEFFECTIV" "DONTBELIEV" "DETERRENT"
```

This scale has been constructed after inserting the item *HIDEOUS* into the first possible position of the minimal scale (*INEFFECTIV*, *DONTBELIEV*, *DETERRENT*). The first scalability criterion for this scale determines if the H_{hik} coefficients of the triples that contain the new item (i.e. *HIDEOUS*) are larger than a user specified λ_2 (default $\lambda_2 = 0$). We can extract all the triples for this specific scale using the `combinations()` function.

```
les <- length(Examplescale)
ExamplescaleTRIPLES <- combinations(n = les, r = 3, v = Examplescale, set = FALSE)
```

From the four triples in total, only the first three are containing the new item *HIDEOUS*. We can obtain the H coefficient for each of these triples with

```
hcoeft[ExamplescaleTRIPLES[1:3, ]]
```

and we can see that the triple (*HIDEOUS, INEFFECTIV, DETERRENT*) has a H coefficient which is lower than λ_2 . Hence, this scale does not fulfill the first criterion and should be excluded from the scale extension process. The first criterion is calculated for every scale possible and the scales that conform to this criterion continue the scale extension process. Lowering the values of λ_2 to a negative number will allow more scales to pass this criterion, while setting λ_2 to a large negative number e.g. -99 will allow all scales to pass this criterion.

The second scale assessment determines which scale or scales contain the item that is the most “uniquely” represented. Let us assume that the number of scales that fulfill the first criterion is six. Moreover, assume that five out of these six scales contain the item *MUSTHAVEIT* and one scale contains the item *CRIMDESERV*. In this scenario the scale that contains the item *CRIMDESERV*, will be the one that continues the scale extension.

The scales that contain the least frequently observed item are checked according to a third criterion. The third and last criterion in the iterative scale extension phase concerns the scalability properties of the new item. The scale that contains the new item with the highest item scalability coefficient will be chosen as the best MUDFOLD scale if and only if $H_j(s) > \lambda_1$ where λ_1 is the lower bound that have been used also in the first step of the item selection algorithm.

In the **ANDRICH** example the algorithm completes five iterations in the second step which means that all the items are included in the MUDFOLD scale. The latter, consists of eight items and shows a scale scalability coefficient equal to 0.64.

After a MUDFOLD scale with a good fit is obtained, one can assess its unfolding quality. This is done by scale diagnostics described by Post (1992) and Post and Snijders (1993). These diagnostics are based on sample proportions from which the unimodality assumption of the scale is evaluated and nonparametric estimates of the item response functions are obtained.

MUDFOLD diagnostics

In this section, we discuss diagnostics implemented in the **mudfold** package, which can be used to assess if a scale s consisting of m items, $j = 1, \dots, m$ conforms with the assumptions **A2** to **A5** of a unidimensional nonmonotone homogeneous MUDFOLD scale.

Diagnostic for assumption A2

Let us denote by \mathbf{X}_{-j} the $n \times (m - 1)$ matrix that contains the responses of n individuals to all the items in the scale except item j . Testing if **A2** (local independence) holds, is equivalent to testing if the positive response on an item depends solely on the latent trait θ , i.e. $P(X_j = 1|\mathbf{X}_{-j}, \theta) = P(X_j = 1|\theta)$. If $p_j = P(X_j = 1)$ denotes the probability of positive response to item j , testing this hypothesis implies fitting the following regularized logistic regression model,

$$\log \frac{p_j}{1 - p_j} = \beta_0 + \sum_{k=1}^{m-1} \beta_k X_{-jk} + \beta_\theta \hat{\theta}, \quad (4)$$

where X_{-jk} denotes the k th column of \mathbf{X}_{-j} and $\hat{\theta} = (\hat{\theta}_1, \dots, \hat{\theta}_n)$ is a nonparametric estimate of the latent attitude with regression parameter β_θ . The response regression parameters β_k are penalized using the least absolute shrinkage and selection operator (LASSO, Tibshirani, 1996). LASSO shrinks the coefficients β_k of the regression in (4) towards zero. If $\beta_k = 0$ for all $k = 1, \dots, m$ then the local independence assumption is fulfilled and the probability

of positive response on the item j depends only on θ . On the other hand if there is any k for which $\beta_k \neq 0$ there is evidence of violations in the local independence assumption. Fitting sparse generalized linear models with simultaneous estimation of the regularization parameter is straightforward in R with the function `cv.glmnet()` that is available with the package `glmnet` (Friedman et al., 2010).

Diagnostic for assumption A3

The condition **A3** required by MUDFOLD is the assumption of unimodality of the IRFs, which are unknown nonlinear functions of the latent trait. In order to obtain estimates of these functions, we use a nonlinear generalized additive model (GAM, Wood, 2011) that is implemented in the R package `mgcv` (Wood, 2017). Specifically, for each item the probability of positive response p_j is modelled as a smooth function of the latent trait θ , that is,

$$\log \frac{p_j}{1 - p_j} = \beta_0 + \beta_\theta f_\theta(\hat{\theta}), \quad (5)$$

where $f_\theta(\hat{\theta})$ is a smooth function of $\hat{\theta}$. Plotting the probability of positive response modelled by (5) against a nonparametric estimate of the latent trait $\hat{\theta}$, should yield a single 'peaked' curve if the unimodality assumption for the IRFs holds.

Diagnostics for assumptions A4 and A5

For the assumptions **A4-A5**, diagnostic statistics that quantify to which extent the scale agrees with these assumptions have been proposed by Post (1992). These statistics are based on conditional IRF probabilities, which are estimated by their corresponding sample proportions and collected into a matrix that is called the conditional adjacency matrix (CAM).

CAM in its (j, k) element contains the conditional frequency that a subject from the sample will choose the row item j given that the column item k is chosen. The probability $P(X_j = 1 | X_k = 1)$ is estimated from the data by dividing the joint frequency of choosing both items j and k by the relative frequency of choosing item k . That is,

$$\text{CAM}_{jk} = \frac{\sum_{i=1}^n x_{ij} x_{ik} / n}{\sum_{i=1}^n x_{ik} / n} = \frac{\sum_{i=1}^n x_{ij} x_{ik}}{\sum_{i=1}^n x_{ik}}, \text{ for } j \neq k. \quad (6)$$

In the package `mudfold`, the CAM can be obtained using the function `CAM()`, which takes as input either a fitted MUDFOLD object or a dataset with the complete responses of n individuals to m items. In the `ANDRICH` dataset example, the CAM of the original data can be calculated using the command `CAM(ANDRICH)`.

Each row of the CAM is regarded as an empirical estimate of the corresponding IRF. Hence, if the ordering of the items is correct, and if assumptions **A1** to **A5** hold, then (i) the observed maxima of the different rows of the CAM are expected to appear around the principal diagonal (moving maxima property), and (ii) the rows of the CAM are expected to show a weakly unimodal pattern. One can potentially evaluate the unfolding model by checking how strongly the observed row patterns of the CAM deviate from the expected patterns described above.

Max statistic (MAX) : The moving maxima property of the CAM corresponds to condition **A4**, which assumes stochastic ordering of the items by their location parameter β_j . In order to formally check this assumption, Post (1992) proposed a statistic that quantifies the violations of the moving maxima property for the rows of the CAM , which is called the max statistic (MAX).

Calculation of the MAX can be done in two ways, namely a top-down and a bottom-up method

$$\text{MAX}_j = \begin{cases} \sum_{k=j+1}^m \max(0, (M_j - M_k)) & (\text{top-down method}) \\ \sum_{k=1}^{j-1} \max(0, (M_k - M_j)) & (\text{bottom-up}), \end{cases} \quad (7)$$

where M_j is the position of the maximum in the j th row of CAM. In order to create a measure of the moving maxima property that is bounded within the interval $[0, 1]$ we divide MAX_j by the number of potential violations of the moving maxima property which are approximately equal to $m^2/12$.

The sum over all rows yields the total MAX statistic of the scale, i.e. $\text{MAX}_{\text{total}} = \sum_{j=1}^m \text{MAX}_j$. The quantity $\text{MAX}_{\text{total}}$ will be the same for both methods in (7), however, the number of items showing positive MAX can be different. In this situation the method that yields the minimum number of items showing positive MAX is chosen. If the number of items with positive MAX is the same for both methods then we choose arbitrarily the top-down method. In the case where M_j is next to a diagonal element then the maximum in the j th row can have two positions and the position that yields the lower MAX value will be chosen.

The MAX statistic can be calculated using the function `MAX()` from the R package **mudfold**, which takes as input either a fitted MUDFOLD object obtained from the main `mudfold()` function, or an object of class "`cam.mdf`" calculated from the function `CAM()`. The argument '`type`' of the `MAX()` function controls if the MAX for the items or the whole scale will be returned to the user. Visual inspection of the observed maxima pattern can also be useful. If the maximum values of the CAM rows are close to the diagonal then the unfolding model holds. The `diagnostics()` will return and plot a matrix with a star at the maximum of each CAM row for visual inspection of their distribution.

Iso statistic (ISO) : In order to quantify if the rows of the CAM show a weakly unimodal pattern, the iso statistic (proposed by I. Molenaar, personal communication) was introduced. Iso statistic (ISO), is a measure for the degree of unimodality violation in the rows of CAM. ISO can be obtained for each item (ISO_j) and their summation results in the total ISO for the scale (ISO_{tot}).

To come up with an ISO value for an item j , one should first locate the maximum in each row of the CAM. If we index m^* the maximum in row j of CAM, the ISO measures deviations from unimodality to the left and right of m^* , i.e.

$$\text{ISO}_j = \sum_{h \leq k \leq m^*} \max(0, \text{CAM}_{jh} - \text{CAM}_{jk}) + \sum_{m^* \leq h \leq k} \max(0, \text{CAM}_{jk} - \text{CAM}_{jh}). \quad (8)$$

The total ISO statistic for a scale consisting of m items is calculated as the sum of the individual ISO statistics, i.e. ISO_j 's, i.e. $\text{ISO}_{\text{total}} = \sum_{j=1}^m \text{ISO}_j$. The ISO statistic, both for an item or for the scale, is zero if the unimodality in row j of the conditional adjacency matrix is not disturbed and positive if disturbances in unimodality occur in row j .

The user can calculate the ISO statistic using the function `ISO()`, which takes as input outputs either from the `mudfold()` function, or from the function `CAM()` and returns a vector with the ISO_j 's for each $j \in \{1, 2, \dots, m\}$ or the sum of this vector if `type = 'scale'`.

All the diagnostic tests discussed in this section are implemented in the function `diagnostics()` of the **mudfold** package. The function `diagnostics()` can be used with fitted objects from the main `mudfold()` function.

Uncertainty estimates for MUDFOLD statistics

Since the sampling distributions of the MUDFOLD's goodness-of-fit and diagnostic statistics are non-standard, calculating their standard errors is not straightforward. Instead, for providing uncertainty estimates of the MUDFOLD statistics both at the item and the scale level, nonparametric bootstrap is used (Efron et al., 1979). Bootstrap is a resampling technique that can be used for assessing uncertainty in instances when statistical inference is based on complex procedures. With bootstrapping we sample R times n samples with replacement from a dataset of size n . The bootstrap samples of the statistic obtained from R iterations are then used to approximate the sampling distribution of the statistic.

Given a MUDFOLD scale s , statistics for items such as the $O_j(s)$, $EO_j(s)$, $H_j(s)$, and the total scale such as the O_{total} , EO_{total} , H_{total} are bootstrapped R times. The bootstrap procedure implemented in **mudfold** depends on the function `boot()` from the R package

boot (Canty and Ripley, 2017). Using the **boot** package allows the user of **mudfold** package to obtain different types of confidence intervals for assessing uncertainty using the function `boot.ci()`.

Additional to the uncertainty estimates, a bootstrap estimate of the unfolding scale can be also calculated. This estimate corresponds to the most frequently obtained MUDFOLD scale in *R* bootstrap iterations. In many instances the bootstrap estimate will coincide with MUDFOLD scale obtained by the item selection algorithm. When the two estimates are different the bootstrap scale estimate can be used to correct the MUDFOLD scale after assessing its properties carefully.

Nonparametric estimation of person ideal points

With MUDFOLD, after obtaining an item ordering (scale) that consists of a (sub) set of m items, $m \leq N$, one can estimate in a nonparametric way subject locations on a latent continuum. Two nonparametric estimators can be used with slightly different properties both based on the Thurstone (1927, 1928) estimator for the measurement of attitudes.

Originally, the Thurstone estimator $\hat{\theta}_i^\beta$ of the i -th respondent location parameter given a vector of known item location parameters $\beta = (\beta_1, \beta_2, \dots, \beta_m)^\top$ was defined as,

$$\hat{\theta}_i^\beta = \frac{\sum_{j=1}^m \beta_j x_{ij}}{\sum_{j=1}^m x_{ij}}, \quad (9)$$

where x_{ij} is the response of person i on item j . The parameter estimate $\hat{\theta}_i^\beta$ for each i takes values within the item parameter range. In MUDFOLD however, the item parameters vector β is unknown, thus we need to estimate it. In order to do so, we make use of two alternative estimates for β 's proposed by Van Schuur (1988) and Johnson (2006), respectively. The former uses item ranks as approximations of the item locations while latter uses item quantiles.

Van Schuur's person parameter estimator uses the item ranks obtained from MUDFOLD's item selection algorithm as estimates for the vector $\beta = (\beta_1, \beta_2, \dots, \beta_m)^\top$. Since MUDFOLD estimates only the rank order of the parameter vector, i.e. $r = (r_1, r_2, \dots, r_m)^\top$ one can define a rank estimate

$$\hat{\beta}_j^r = r_j, \quad (10)$$

where r_j is the rank of the item j on the MUDFOLD scale. By using the estimated ranks as approximations of the parameter vector we can estimate a respondent's location as the mean of the endorsed item ranks. That is,

$$\hat{\theta}_i^r = \begin{cases} \frac{\sum_{j=1}^m r_j x_{ij}}{\sum_{j=1}^m x_{ij}}, & \text{if } \sum_{j=1}^m x_{ij} > 0 \\ \text{undefined}, & \text{if } \sum_{j=1}^m x_{ij} = 0. \end{cases} \quad (11)$$

Alternatively Johnson's quantile estimator bounds both estimates for θ 's and β 's within a unit interval. This estimator uses the item ranks divided by the length of the scale m as approximations for the β vector. In all the estimators described in this section, no estimates can be defined for individuals with total score $X_i^+ = \sum_{j=1}^m x_{ij}$ equal to zero. These individuals are not endorsing any item and therefore provide no information whether they belong to the extreme right of the scale or to extreme left. The user of the package **mudfold** can choose between Van Schuur's and Johnson's estimators for obtaining persons scores on the factors.

Missing values

Missing data occur when intended responses from one or multiple persons are not provided. Handling missing values is critical since it can bias inferences or lead to wrong conclusions. One way to go is to ignore the missing observations by applying list-wise deletion. This,

however, can lead to a great loss of information especially if the number of missing values is large. The other approach, is to replace the missing values with actual values which is called imputation.

In the case of random missing value mechanisms such as missing completely at random (MCAR) and missing at random (MAR) (Rubin, 1976; Little and Rubin, 1987), different approaches can be used in order to impute the missing observations. Imputation within IRT is in general associated with more accurate estimates of item location and discrimination parameters under several missing data generating mechanisms (Sulis and Porcu, 2017). In the package **mudfold** missing values can be imputed using the logistic regression version of multiple multivariate imputation by chained equations (MICE). The latter is available from the R package **mice**. MICE imputation within **mudfold** can be used solely or in combination with bootstrap uncertainty estimates. In the latter case, each bootstrap sample is imputed before fitting a MUDFOLD scale, while in the former the data are imputed M times and the results are averaged across the M datasets.

The mudfold package

The R package **mudfold** contains a collection of functions related to the MUDFOLD item selection algorithm. In the following we describe the functionality of the package and the ANDRICH dataset is used for demonstration purposes.

Description of the functions **mudfold()** and **as.mudfold()**

The main function of this package, called **mudfold()**, fits Van Schuur's item selection algorithm to binary data in order to obtain a unidimensional ordinal scale for the persons. The **mudfold()** function can be called with,

```
mudfold(data, estimation, lambda1, lambda2, start.scale,
nboot, missings, nmice, seed, mincor, ...)
```

The functions has ten main arguments where only the first one is obligatory. These are:

data: The input data, i.e. a $n \times N$ **data.frame** or **matrix**, with persons in the rows and items in the columns. It contains the binary responses of n individuals on N items. . .

estimation: This argument handles the nonparametric estimation of the person parameters. The default, **estimation = "rank"** uses a rank based estimator (Van Schuur, 1988). Alternatively, person parameters are obtained by a quantile estimator (Johnson, 2006), which is accessible by setting **estimation = "quantile"**.

lambda1: The parameter λ_1 , $0 \leq \lambda_1 \leq 1$ is a user specified lower bound for scalability criteria that are used in MUDFOLD's item selection algorithm. In the default setting, $\lambda_1 = 0.3$. Large values of λ_1 lead to more strict criteria in the item selection procedure.

lambda2: Parameter λ_2 , $-\infty < \lambda_2 \leq 1$ is a lower bound explicitly used at the first scalability criterion of the second step (default $\lambda_2 = 0$).

start.scale: The user can pass to this argument a character vector of length greater than or equal to three, containing ordered item names from **colnames(data)** that are used as the best elementary scale for the second step of the item selection algorithm. If **start.scale = NULL** (default), the first step of the item selection algorithm determines the best elementary triple of items that is extended in the second step.

nboot: Argument that controls the number of bootstrap iterations. If **nboot = NULL** (default) no bootstrap is applied.

missings: Argument that controls treatment of missing values. If **missings = "omit"** (default) list-wise deletion is applied to **data**. If **missings = "impute"** then the **mice** function is applied to **data** in order to impute the missings **nmice** times.

nmice: Argument that controls the number of mice imputations (This argument is used only when **missings = "impute"** and **nboot = NULL**).

seed: Argument that is used for reproducibility of bootstrap results.

mincor: This can be scalar, numeric vector (of size `ncol(data)`) or numeric matrix (square, of size `ncol(data)`) specifying the minimum threshold(s) against which the absolute correlation in the data is compared. See `?mice:::quickpred` for more details. To be used when mice becomes problematic due to co-linear terms.

... : Additional arguments to be passed into the `boot()` function (see `?boot` in R).

The function `mudfold()` internally has four main steps. A data checking step, the first step of the item selection process, the second step of the item selection process, and the bootstrap step if the user chooses this option. The output of `mudfold()`, is a `list()` of class "`mdf`" that contains information for each internal step of the function. The first element of the output list contains information on the function call. The second element contains results of the data checking step. The next element of the output contains descriptive statistics obtained from the observed data and the last element of the output has all the information from the the fitting process (triple statistics, first step, second step). If bootstrap is applied to estimate uncertainty , an additional element that contains the bootstrap information is given to the output.

For example, if you want to fit a MUDFOLD scale to the `ANDRICH` data and run a nonparametric bootstrap with $R = 100$ iterations in parallel, you can specify it directly into the `mudfold()` function as follows.

```
fitANDRICH <- mudfold(ANDRICH, nboot = 100, parallel = "multicore", seed = 1)
```

In the example above, the first two arguments are core in the `mudfold()` function. The third argument `parallel` is an argument of the `boot()` function that runs bootstrapping in parallel fashion in order to reduce computational time. The last argument `seed` is used to ensure reproducibility of the bootstrap results.

In some cases the unfolding scale could be known. In these instances, the user is interested in obtaining the MUDFOLD goodness-of-fit and diagnostic statistics for the given scale. The function `as.mudfold()` can be used for treating the given rank order of the items as a MUDFOLD scale. The function uses only the first two arguments of the `mudfold()` function. In principle, this function transforms a given scale into an S3 class "`mdf`" object.

Description of the generic functions

For "`mdf`" objects from the `mudfold()` or `as.mudfold()` functions, generic functions for `print()`, `summary()` and `plot()` and `coef()` are available. The generic function `print.mdf()` can be accessed with,

```
print(x)
```

where `x` is an "`mdf`" class object. This function prints information for `x`, such as time elapsed for fitting, warnings from the data checking step, convergence for each step of the algorithm and statistics with bootstrap confidence intervals if `nboot` is not equal to `NULL`.

In the `ANDRICH` data example, the command `print(fitANDRICH)` is used to print information from the `fitANDRICH` object to the console. The function call together with the elapsed time to fit the model, the number of individuals, and the number of items used in the analysis is the first part of the output. Next, the values of the `mudfold()` arguments are given, which are followed by convergence indicators for each step of the item selection algorithm. Scale statistics such as the scalability coefficient and the ISO statistic are also printed together with their percentile confidence intervals obtained in 1000 bootstrap iterations. The summary of the bootstrap iterations finalize the output when printing the `fitANDRICH` object.

The function `summary` is a generic function that is summarizing information from model fitting functions. In our case the output of `summary.mdf()` is a list object summarizing results from the `mudfold()` function. The function can be called via

```
summary(object, boot, type = "perc", ...)
```

and consists of three arguments:

object: a list of class "mdf", output of the `mudfold()` function.

boot: logical argument that controls if bootstrap confidence intervals and bootstrap summary for each coefficient will be returned. If `boot = FALSE` (default) no information for bootstrap is returned. When `boot=TRUE`, confidence intervals, standard errors, biases, calculated from the bootstrap iterations for each parameter are given with the output.

type: The type of bootstrap confidence intervals to be calculated if the argument `boot = TRUE`. Available options are "norm", "basic", "perc" (default), and "bca". See the argument `type` of the `boot.CI()` for details.

The output of the `summary.mdf()` is a list with two main components. The first component of the list is a `data.frame` with scale statistics and the second component is a list with item statistics.

Typing `summary(fitANDRICH, boot = TRUE)` into the R console will return the summary of the fitted scale to the ANDRICH data. The output consists of six distinct `data.frame` objects. The first `data.frame` contains information on scale statistics with their bootstrapped statistics. The next four `data.frame` objects correspond to the H coefficients, the ISO statistics, the observed errors, and the expected errors for each item in the scale together with their bootstrap summary statistics. The last `data.frame` gives descriptive statistics for the items in the scales.

A generic function for plotting S3 class "mdf" objects is also available to the user. The function `plot.mdf()` returns empirical estimates of the IRFs, the order of the items on the latent continuum or a histogram of the person parameters . You can plot "mdf" class objects with the following R syntax.

```
plot(x, select = NULL, plot.type = "IRF")
```

This function consists of three arguments from which the first is the usual argument `x` which stands for the "mdf" object to be plotted. The argument `plot.type` controls the type of plot that is returned, and three types of plots are available. If `plot.type = "scale"`, a unidimensional continuum with the items in the obtained rank order is returned. In the default settings of this function (i.e. `plot.type = "IRF"`), the corresponding plot has the items on the x-axis indicating their order on the latent continuum and the probability of a positive response on the y-axis. The IRF of each item among the latent scale is plotted with different colours. When `plot.type = "IRF"` will return a plot with the distribution of person parameters on the latent continuum. The argument `select` is optional and provides the possibility for the user to plot a subset of items. The user can provide in this argument a vector of item names to be plotted. If `select = NULL`, the function returns the estimated IRFs for all items in the obtained MUDFOLD scale. For plotting S3 class "mdf" objects, we use the functions `na.approx()`, `melt()` and `ggplot()` from the R packages `zoo` (Zeileis and Grothendieck, 2005), `reshape2` (Wickham, 2007), and `ggplot2` (Wickham, 2009), respectively.

A generic `coef.mdf()` function for S3 class "mdf" objects can also be used. This function is a simple wrapper that uses a single argument named 'type'. The `coef.mdf()` will extract nonparametric estimates of: persons ranks when `type = "persons"`, item ranks when `type = "items"`, or both when `type = "all"` from a fitted MUDFOLD object.

The `diagnostics()` function

After a scale has been obtained, scale diagnostics need to be applied in order to assess its unfolding properties. The MUDFOLD diagnostics described in section 2.4 of this paper are implemented into a function named `diagnostics()` that can calculate all of them simultaneously. The function syntax is,

```
diagnostics(x, boot, nlambda, lambda.crit, type, k, which, plot)
```

and uses eight arguments described below.

x: a list of class "mdf", output of the `mudfold()` function.

boot: logical argument that controls if bootstrap confidence intervals and summary for the H coefficients and the ISO and MAX statistics will be returned. If `boot = FALSE` (default) no information for bootstrap is returned. When `boot = TRUE`, confidence intervals, standard errors, biases, calculated from the bootstrap iterations for each diagnostic are given with the output.

nlambda: The number of regularization parameters to be used in `cv.glmnet()` function when testing local independence.

lambda.crit: String that specifies the criterion to be used by cross-validation for choosing the optimal regularization parameter. Available options are "class" (default), "deviance", "auc", "mse", "mae". See the argument '`type.measure`' in the `cv.glmnet()` function for more details.

type: The type of bootstrap confidence intervals to be calculated if the argument `boot = TRUE`. Available options are "norm", "basic", "perc" (default), and "bca". See the argument `type` of the `boot.CI()` for details.

k: The dimension of the basis in the thin plate spline that is used when testing for IRF unimodality. The default value is `k = 4`.

which: Which diagnostic should be returned by the function. Available options are "H", "LI", "UM", "ISO", "MAX", "STAR", "all" (default).

plot: Logical. Should plots be returned for the diagnostics that can be plotted? Default value is `plot = TRUE`.

For the ANDRICH data example the command `diagnostics(fitANDRICH)` will calculate and plot the scale diagnostics for the `fitANDRICH` object.

Unfolding data simulation and description of the `mudfoldsim()` function

In order to provide the user the flexibility of simulating unfolding data, the function `mudfoldsim()` is available from the **mufold** package. The responses of subjects on distinct items are simulated with the use of a flexible parametric IRF that generalizes proximity relations between item and person parameters.

Assume that we want to simulate a test dataset with responses from n individuals indexed by $i = 1, 2, \dots, n$ on N proximity items (indexed by j) with latent parameters θ_i and β_j respectively. The vector of item parameters $\beta = (\beta_1, \dots, \beta_N)^\top$ is drawn at random from a standard normal distribution. For the person parameters, the user can choose if they will follow a standard normal distribution, or they will be drawn uniformly in the range of item parameters. Simulating person parameters from a standard normal distribution may imply that a number of individuals are located too far to the left or right of the most extreme items (due to sampling variation). These subjects will not agree with any item. These responses are not useful in unfolding analysis since no discriminant information is provided for the items in the scale. The user of **mufold** package is free to include or exclude such type of responses.

Unfolding models are also known as distance models since they model the probability of positive endorsement of item j from individual i as a function of the proximity between θ_i and β_j . We consider a linear transformation τ_{ij} of the squared difference $d_{ij}^2 = (\theta_i - \beta_j)^2$ given by $\tau_{ij} = \gamma_1 + \gamma_2 d_{ij}^2$, where the parameters γ_1 (deterministic parameter) and γ_2 (discrimination parameter) are fixed.

Using τ_{ij} with the standard logistic function one obtains a parametric IRF $f(\tau_{ij}) = \frac{1}{1+e^{-\tau_{ij}}}$. Consequently, the positive binary response of individual i on item j can be considered as the outcome of a Bernoulli trial with "success" probability $1/(1+e^{-\tau_{ij}})$. Hence, the item response variables X_{ij} that contain binary responses from n individuals on N items, follow a Bernoulli distribution according to,

$$X_{ij} \sim \text{Bernoulli} \left(\frac{1}{1+e^{-\tau_{ij}}} \right) \text{ for } i = 1, \dots, n, j = 1, \dots, N. \quad (12)$$

In `mudfoldsim()` function, the model parameters $\gamma(\cdot)$ are user specified with default settings $\gamma_1 = 5$ and $\gamma_2 = -10$ respectively. This specific set up of the model parameters produces nearly deterministic response curves for the subjects which in turn guarantees that the number of observed errors is small.

We note that the IRF proposed by Andrich (1988) is a special case of the one implemented in the `mudfoldsim()` function for $\gamma_1 = 0$ and $\gamma_2 = -1$. This parametric simulation method is implemented in a flexible R function available from the **mufold** package. This function consists of several arguments that allow the user to control the unfolding properties of the simulated data. The function in its default settings can be called easily with the following syntax,

```
mudfoldsim(N, n, gamma1 = 5, gamma2 = -10, zeros = FALSE, parameters = "normal",
seed = NULL)
```

and makes use of six user-specified arguments:

N: An integer corresponding to the number of items to be simulated.

n: The number of persons to be simulated.

gamma1: This argument is passed to the IRF. Controls the γ_1 or discriminative parameter of the IRF. The higher the parameter the larger the number of items that individuals tend to endorse if parameter γ_2 is kept constant.

gamma2: The deterministic parameter (i.e. γ_2) of the IRF. As the value of this parameter decreases, individuals tend to make less “errors” in their responses (i.e. their responses are more in line with the unfolding scale).

zeros: A logical argument that controls if individuals who endorse no items will be simulated. If `zeros=TRUE` the function allows for individuals that are not endorsing any of the items. On the other hand, if `zeros=FALSE` (default) only individuals who endorse at least one item will be part of the simulated data.

parameters: Argument for the person parameters with two options available. In the default option `parameters="normal"` and in this case the person parameters are drawn from a standard normal distribution. On the other hand, the user can set this argument equal to `"uniform"` which implies that subject parameters will be drawn uniformly in the range of the item parameters.

seed: An integer to be used in the `set.seed()` function. If `seed=NULL` (default), then the seed is not set.

The output of the `mudfoldsim()` function is a list containing the simulated data (in a random item order), the parameters used in the IRF, and the matrix of probabilities under which the binary data has been sampled.

Description of the `pick()` function

Since the main `mufold()` function is designed for dichotomous (binary) items, we provide the user with the function `pick()`. The latter, is used to transform quantitative or ordinal type of variables into a binary form. The underlying idea of this function is that the individual selects those items with the highest preference. This transformation can be done in two different ways, either by user specified cut-off value(s) or by assuming a pick K out of N (individuals are asked to explicitly pick K out of N items) response process, where each response vector consists of the K highest valued items. Dichotomization is performed row-wise by default, however the user can also perform the transformation column-wise.

The R function `pick()` can be utilized with the following code,

```
pick(x, k = NULL, cutoff = NULL, byItem = FALSE)
```

and makes use of four parameters. These are,

- x: A `data.frame` or `matrix` with persons in the rows and items in the columns containing quantitative or ordinal type of responses from n individuals/raters on N items. Missing values are not allowed.
- k: This integer ($1 \leq k \leq N$) controls the number of items a person can pick (default `k=NULL`). This argument is used if one wants to transform the data into pick K out of N form. If the parameter `k` is provided by the user, then `cutoff` should be `NULL` and vice versa.
- `cutoff`: The numeric value(s) that will be used as thresholds for the transformation (default `cutoff=NULL`). Any value greater than or equal to the `cutoff` will be 1 and 0 otherwise. The length of this argument should be equal to 1 (indicating same threshold for all rows of `x`) or equal to n (when `byItem=FALSE`) which imposes an explicit cut-off value for each individual in `x`. If `byItem=TRUE` then the length of this parameter should be 1 (global cut-off value) or N (explicit cut-off per item).
- `byItem`: This is a logical argument. If `byItem=TRUE`, the transformation is applied on the columns of `x`. In the default `byItem=FALSE`, the function "picks" items row-wise.

In the default parameter settings of the function `pick()`, the parameters `k` and `cutoff` respectively are equal to `NULL`. In this case, the mean from N responses is used as a person-specific cut-off value (if `byItem=FALSE`). When `byItem=TRUE` (with `k, cutoff` equal to `NULL`) then the item mean over all individuals is used as an item specific cut-off value. The parameters `k` and `cutoff` are responsible for different dichotomization processes and they cannot be used simultaneously, which means that only one of the two arguments can be different than `NULL`.

In the case in which the user chooses to transform the data assuming that persons are asked to pick exactly K out of N items, ties can occur. If x_i is a response vector subject to transformation, in which ties exist, then we select among the tied items at random.

Generally, dichotomization should be avoided since it could distort the data structure and lead into information loss. Models that take into account information different categories should be preferred over dichotomization for polytomous data.

Applications

In this section we provide examples of how to use MUDFOLD method on two datasets, which are provided with the `mudfold` package. The first application is from the field of psychometrics while the second example is a linguistic application.

The commands `install.packages("mudfold")` and `library(mudfold)` will download, install and load the `mudfold` package so it can be used. The command `set.seed(1)` will set the seed for reproducibility.

Loneliness data

In order to demonstrate the functionality of the `mudfold` package we re-analyze questionnaire data following the strategy suggested by Post et al. (2001). For this purpose, we use a unidimensional measurement scale for loneliness that follows the definitions of a Rasch scale and has been constructed by de Jong-Gierveld and Kamphuls (1985). De Jong-Gierveld loneliness scale consists of eleven items, five of which are positive and six are negative. The items in the loneliness scale are given below and the sign next to the items corresponds to the item content.

A:	There is always someone I can talk to about my day to day problems	+
B:	I miss having a really close friend	-
C:	I experience a general sense of emptiness	-
D:	There are plenty of people I can lean on in case of trouble	+
E:	I miss the pleasure of company of others	-
F:	I find my circle of friends and acquaintances too limited	-
G:	There are many people that I can count on completely	+
H:	There are enough people that I feel close to	+
I:	I miss having people around	-
J:	Often I feel rejected	-
K:	I can call on my friends whenever I need them	+

Each item in the scale has three possible levels of response, i.e. “no”, “more or less”, “yes” and dichotomization methods that involve item reverse coding have been proposed by De Jong and van Tilburg (1999). These methods as well as the determination of dimensionality of this scale have been under critical discussion. Following this discussion, Post et al. (2001) reanalyzed the loneliness scale data obtained from the NESTOR study (Knipscheer et al., 1995) using MUDFOLD in a three step analysis routine.

Persons with missing responses are removed from the data ($n_{miss} = 69$). The dataset with the complete responses is included in the R package **mudfold** in R data format. List-wise deletion in this case yields identical results with MICE imputation. Following the routine suggested by Post et al. (2001) responses of each subject are dichotomized setting “yes” versus “no” and “more or less”.

The threshold that is used for the main analysis has been determined on the basis of MUDFOLD scale analysis on datasets with different thresholds. Specifically, the data has been dichotomized using as thresholds the response, (i) “yes”, (ii) “more or less”, (iii) different thresholds per item where the response category “more or less” is collapsed with the smaller category between “yes” and “no”. The results from this analysis showed that dichotomizing the data at the higher preference will yield the best unfolding measurement scale for loneliness.

Dichotomizing the data at “yes” is straightforward with the **pick()** function.

```
data("Loneliness")
dat <- pick(Loneliness, cutoff = 3)
```

In the first step of the analysis, we conduct a MUDFOLD scale search on the transformed binary responses of $n = 3987$ individuals on $N = 11$ items. The λ_1 parameter in the **mudfold()** function is set to $\lambda_1 = 0.1$ since the default value leads to a minimal scale of length three.

```
Lonelift <- mudfold(dat, lambda1 = 0.1, nboot = 100, seed = 1)
```

The function takes about five minutes to run 100 bootstrap iterations. The resulting scale and its associated statistics can be obtained by summarizing the **Lonelift** object.

```
loneliSummary <- summary(Lonelift, boot = TRUE)
```

The MUDFOLD scale for the **Loneliness** data in its estimated rank order is:

```
loneliScale <- loneliSummary$ITEM_STATS$ITEM_DESCRIPTIVES$items
loneliScale
## "G" "H" "D" "K" "C" "E" "I" "F"
```

The scale has length eight, with the first four items positively formulated and the last four negatively formulated. Items A,B, and J are excluded from the scale. This is because some triples (with respect to the item rank order) that include these items have scalability coefficient H_{hjk} lower than λ_2 . Statistics for the resulting MUDFOLD scale and each item explicitly can be accessed directly from the summary object **loneliSummary**. Scale statistics with their bootstrap uncertainty estimates can be obtained with the following command.

```
loneliSummary$SCALE_STATS[1:3, ]
##           value  perc_lower95CI perc_upper95CI boot(mean) boot(bias) boot(se) boot(iter)
## H(scale)    0.536      0.436      0.571     0.511     -0.025     0.031      100
## ISO(scale)   0.078      0.001      1.753     0.384      0.306     0.459      100
## MAX(scale)   0.000      0.000      2.400     0.381      0.381     0.683      100
```

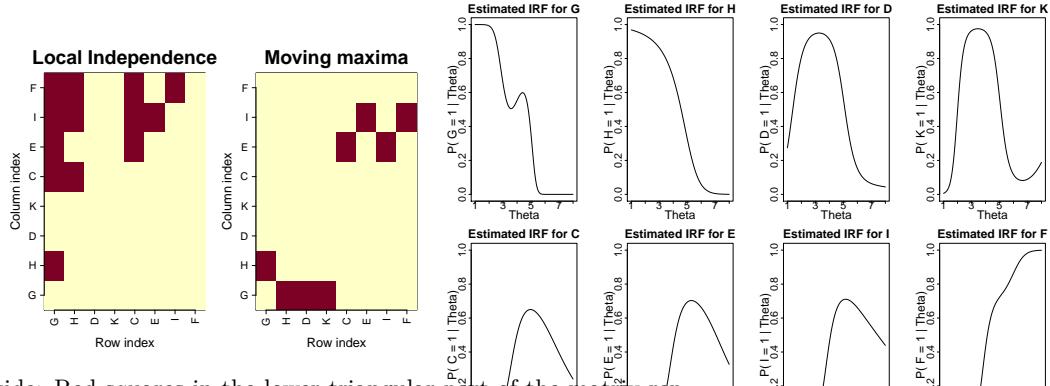


Figure 1: Left hand side: Red squares in the lower triangular part of the matrix represent pairs of conditionally dependent items. Right hand side: Red squares represent the position of the observed maxima in the CAM rows for the **Loneliness** unfolding scale.

Figure 2: The estimated item response function for each item in the **Loneliness** unfolding scale.

The output above, in each row shows a scale statistic and its columns correspond to the bootstrap properties of this statistic. The H coefficient for the scale shows strong evidence towards unidimensionality ($H_{\text{total}}(s) \approx 0.54$, $se = 0.031$), the ISO statistic is low ($\text{ISO}_{\text{total}} \approx 0.08$, $se = 0.459$) denoting small amount of violations of the manifest unimodality, and the MAX statistic is zero ($se = 0.683$) meaning no violations of the stochastic ordering.

Scale diagnostics are given in Figure 1 and 2. Visual inspection if the maxima of the CAM rows are a nondecreasing function of the item ranks, violations of the local independence assumption, and the IRF for each item in the **Loneliness** unfolding scale can be obtained by using the `diagnostics()` function as shown below.

```
par(mfrow = c(1, 2))
# testing for local independence
diagnostics(Lonelift, which = "LI")
# visual inspection of moving maxima
diagnostics(Lonelift, which = "STAR")
par(mfrow=c(2,4))
# visual inspection for IRF unimodality
diagnostics(Lonelift, which = "UM")
par(mfrow = c(1, 1))
```

The H coefficients for each item in the scale are also available in the summary object and can be accessed by:

	loneliSummary\$ITEM_STATS\$H_MUDFOLD_items	value	perc_lower95CI	perc_upper95CI	boot(mean)	boot(bias)	boot(se)	boot(iter)
H(G)	0.54	0.444	0.573	0.510	-0.034	0.035	96	
H(H)	0.52	0.440	0.543	0.495	-0.027	0.025	72	
H(D)	0.51	0.400	0.553	0.498	-0.015	0.032	65	
H(K)	0.51	0.440	0.554	0.495	-0.016	0.025	60	
H(C)	0.55	0.404	0.590	0.513	-0.041	0.049	76	
H(E)	0.57	0.491	0.610	0.555	-0.016	0.029	78	
H(I)	0.55	0.493	0.586	0.541	-0.011	0.022	47	
H(F)	0.52	0.349	0.546	0.464	-0.057	0.058	84	

From the item fit we can see that the H coefficient for each item in the scale is above 0.5 which means that all the items are scalable together. Looking at the column `boot(iter)` of the output above you can get information for the number of times each item was included in a MUDFOLD scale out of $R = 100$ bootstrap iterations. The item G was the most frequently included item (96%) while the items K, I were included less frequently in a MUDFOLD scale compared to the other items (60% and 47% respectively). Typing `loneliSummary$ITEM_STATS$ISO_MUDFOLD_items` into the R console will return a summary of the ISO statistic for each item in the scale. The latter, shows that only small violations of unimodality occur for the items in the scale. The same holds for the MAX statistic (it can be accessed by `loneliSummary$ITEM_STATS$MAX_MUDFOLD_items`), which shows zero values for all the items in the scale.

After the scale is obtained and checked for its conformity to the unfolding principles

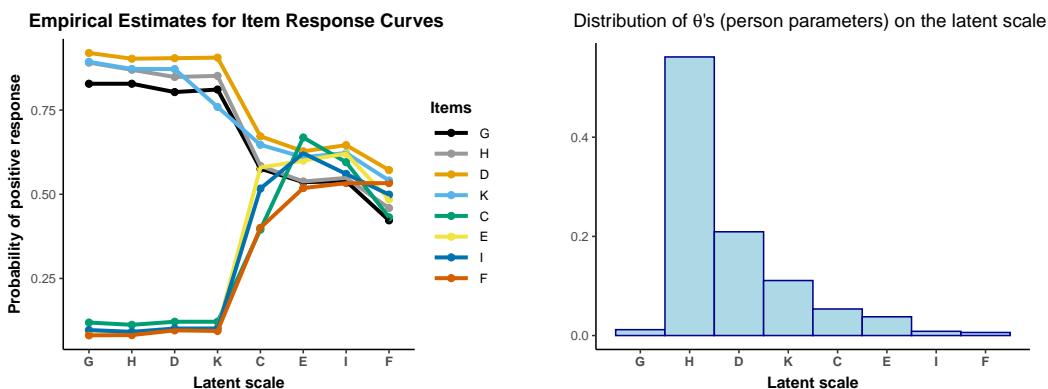


Figure 3: MUDFOLD’s empirical estimates of the IRFs for the Loneliness unfolding scale.

Figure 4: The distribution of estimated person ranks for the Loneliness unfolding scale.

we can visualize the estimated empirical IRFs and the distribution of the estimated person parameters. Plots for the IRFs and the person parameters can be obtained by:

```
plot(Lonelifit,plot.type = "IRF")
plot(Lonelifit,plot.type = "persons")
```

Figures 3 and 4 show the empirical estimates of the IRFs and the distribution of the person parameters respectively. In figure 3 you can see that the scale clearly consists of four positively formulated items in its beginning for which the IRF is decreasing as one moves from the left to the right of the scale, and four negatively formulated items in the end for which the IRF is increasing as one moves from the left to the right of the scale. In figure 4 we can see that the sample under consideration tends to feel less lonely since the distribution of the person parameters is skewed to the right. In such example, clearly any parametric model that assumed a latent normal distribution of the latent person parameters would be inappropriate.

Plato’s seven works data

In this section, we present an application of MUDFOLD method to the **Plato7** data set. This dataset is available from the R package **smacof** (de Leeuw and Mair, 2009) and has been also included in the **mudfold** package. The data can be loaded into the R environment with the command `data("Plato7")`.

Plato7 contains information on the quantity distribution over the sentence ending from seven works of Plato (D. R. Cox, 1959). Specifically, the last five syllables from each sentence in seven Plato’s works are extracted and categorized as short or long. This produces $2^5 = 32$ possible combinations of short-long syllables of length five, which are called clausulas and can be used to identify rhythmic changes in the literary style. The quantity of the clausulas in each work of Plato is recorded in terms of proportions.

The question is whether it is possible using these data to assign a chronological order to the works of Plato. Particularly, it is known that Plato wrote first the **Republic** and last the **Laws**. In between **Republic** and **Laws**, Plato wrote the **Critias**, **Philebus**, **Politicus**, **Sophist** and **Timaeus**. However, the exact order of these five works is unknown. Assuming that the change in Plato’s literary style was monotone in time, we might be able to assign a time order in his works by analyzing the clausula’s distribution in each Plato’s work.

We consider the development of Plato’s literary style as a unidimensional scale, on which clausulas and works are ordered. In this analysis we consider that the quantity of clausula i in Plato’s work j will be governed by a proximity relation. That is, each clausula with a parameter θ_i on a latent literary style continuum tends to prefer (appear most frequently in) the works of Plato with parameters β_j close to θ_i .

Since the data is given in continuous form, we transform the percentages into binary format in order to apply MUDFOLD. We consider the mean quantity of each clausula as

an explicit cut-off value for the transformation. The latter can be seen as a pick *any* out of N response process where the number of items “picked” varies across subjects. We can apply the transformation with the function `pick()` from the **mufold** package in its default settings as follows.

```
dat.Plato <- pick(Plato7)
```

After the transformation, we end up with a matrix containing the binary preferences of $n = 32$ clausulas on $N = 7$ works of Plato. Now we can fit a MUDFOLD scale (with bootstrap for assessing parameter uncertainty) to the transformed data with the default search settings and study its summary.

```
fitPlato <- mufold(dat.Plato, nboot = 100, seed = 1)
summaryPlato <- summary(fitPlato, boot = TRUE)
```

We can check the MUDFOLD scale from the summary object.

```
summaryPlato$SCALE_STATS[1:3, ]
##           value perc_lower95CI perc_upper95CI boot(mean) boot(bias) boot(se) boot(iter)
## H(scale)  0.558      0.457       1.000     0.714     0.156     0.146     100
## ISO(scale) 0.146      0.000       1.129     0.141    -0.005     0.254     100
## MAX(scale) 0.000      0.000       0.850     0.035     0.035     0.191     100
```

The scale shows strong scalability properties with $H_{\text{total}(s)} = 0.56$, and low ISO statistic ($\text{ISO}_{\text{total}} = 0.15$). Since the scale is strong, the next step is to check the rank order of the items in the MUDFOLD scale and their scalability properties.

```
summaryPlato$ITEM_STATS$H_MUDFOLD_items
##           value perc_lower95CI perc_upper95CI boot(mean) boot(bias) boot(se) boot(iter)
## H(Republic) 0.66      0.362       1         0.761     0.097     0.179     70
## H(Sophist)  0.41      0.381       1         0.656     0.241     0.157     70
## H(Politicus) 0.58      0.392       1         0.662     0.078     0.161     62
## H(Philebus)  0.63      0.429       1         0.726     0.094     0.141     83
## H(Laws)     0.51      0.394       1         0.688     0.176     0.148     77
```

The results shows that the MUDFOLD scale has length five and the items **Critias** and **Timaeus** have been excluded from the measurement process. **Republic** is correctly ordered first and **Laws** is correctly ordered last among Plato’s works. Almost all the items are strong unfolding items with $H_j(s)$ higher than 0.5 which means that the items are scalable together in one dimension. The item **Sophist** shows moderate unfolding strength with the lowest item scalability coefficient (i.e. $H_j(s) = 0.41$) while the item **Republic** is the strongest unfolding item in the scale.

Since the ISO statistic for the scale is positive one may wants to check which items are responsible for the small amount of manifest unimodality violations that are observed. Assessing these violations for each item involves checking their ISO statistics.

```
summaryPlato$ITEM_STATS$ISO_MUDFOLD_items
##           value perc_lower95CI perc_upper95CI boot(mean) boot(bias) boot(se) boot(iter)
## ISO(Republic) 0.104      0         0.365     0.036    -0.068     0.076     74
## ISO(Sophist)  0.042      0         0.326     0.039    -0.003     0.078     70
## ISO(Politicus) 0.000      0         0.082     0.005     0.005     0.018     65
## ISO(Philebus)  0.000      0         0.576     0.027     0.027     0.117     87
## ISO(Laws)     0.000      0         0.186     0.019     0.019     0.067     78
```

The obtained summary output for the ISO statistics of the items in the MUDFOLD scale show that **Republic** is the item with the higher manifest unimodality errors in its estimated IRF with an iso statistic value of 0.1. The higher uncertainty is observed for the item **Philebus** that shows a bootstrap standard error of 0.1.

The estimated empirical IRFs and the estimated IRFs for the items in the **Plato7** unfolding scale can be visualized with

```
plot(fitPlato, plot.type = "IRF")
par(mfrow = c(2, 3))
diagnostics(fitPlato, which = "UM")
par(mfrow = c(1, 1))
```

and the output is shown in figures 5 and 6 respectively. From figure 5 it can be seen that the scale consists of two items in the first positions (i.e. **Republic** and **Sophist**) with decreasing empirical IRFs as one moves from the left to the right hand side of the latent scale. These

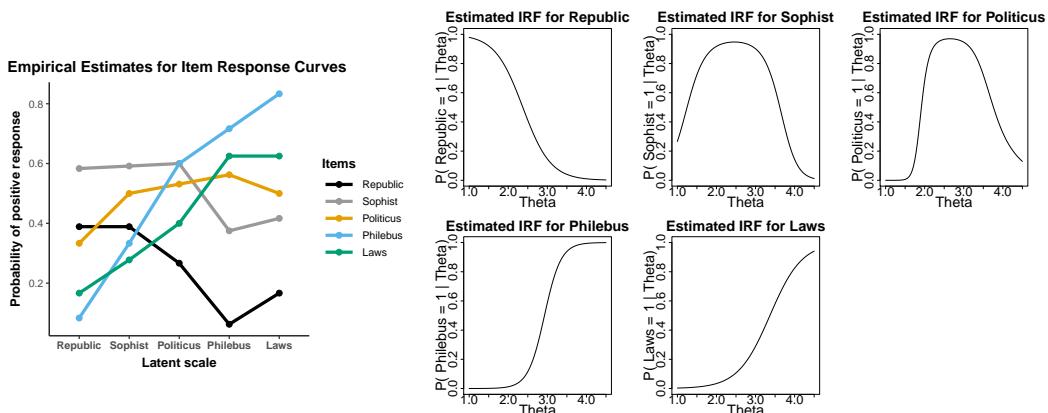


Figure 5: MUDFOLD’s empirical estimates of the IRFs for the items in the `Plato7` unfolding scale.

Figure 6: MUDFOLD’s estimates of the IRFs for the items in the `Plato7` unfolding scale.

two items show small amount of manifest unimodality violations which can be seen at the end of their IRFs where the value of the curves is larger for the item **Laws** compared to item **Philebus**. Third in the scale is the item **Politicus** for which the empirical IRF shows a single-peak shape. **Politicus** is followed by the items **Philebus** and **Laws** with increasing empirical IRFs at positions four and five of the scale. The estimates of the IRFs are shown in figure 6 with no obvious violations of the IRF unimodality.

Other diagnostics can be obtained by the `diagnostics()` function. In this example the bootstrap estimate of the scale with the estimated MUDFOLD scale are slightly different. In such instances an additional element with a summary of the scale estimated by the bootstrap is included in the output. Accessing the summary of the bootstrap scale is straightforward with `summaryPlato$BOOT_SCALE`.

Summary

In this paper we introduced an R package named **mudfold** (Balafas et al., 2019). The latter is available under general public license ($\text{GPL} \geq 2$) from the Comprehensive R Archive Network (CRAN) at <http://CRAN.R-project.org/package=mudfold>. This package implements a nonparametric item response theory model for unfolding proposed by Van Schuur (1984, 1988) and further developed by Post (1992) (see also Johnson (2006)).

The **mudfold** package is an addition to a broad family of R packages that fit IRT models. The approach described here is an additional exploratory and validation method when fitting such models. Moreover it adds to the package **mokken** for the case in which proximity item response data needs to be analysed.

Looking to the future our focus will be on extending the functionality of this package. In detail, we aim on the implementation of a more efficient item selection algorithm which can reduce the computational cost implied from the old fashioned iterative algorithm presented here when the sample size and item number are significantly increasing. Methodologies for handling multicategory type of items (Van Schuur, 1984) are not yet implemented in the package, however, we plan to extend its applicability in the future. Last but not least, a parametric version of MUDFOLD method based on the IRF implemented in the `mudfoldsim()` will offer a complete framework for the analysis of data that have been generated under an unfolding response process.

Bibliography

- D. Andrich. The application of an unfolding model of the pirt type to the measurement of attitude. *Applied psychological measurement*, 12(1):33–51, 1988. URL <https://doi.org/10.1177/014662168801200105>. [p]

- D. Andrich. A hyperbolic cosine irt model for unfolding direct responses of persons to items. In W. J. van der Linden and R. K. Hambleton, editors, *Handbook of Modern Item Response Theory*, pages 399–414. Springer New York, New York, NY, 1997. ISBN 978-1-4757-2691-6. URL https://doi.org/10.1007/978-1-4757-2691-6_23. [p]
- D. Andrich and G. Luo. A hyperbolic cosine latent trait model for unfolding dichotomous single-stimulus responses. *Applied Psychological Measurement*, 17(3):253–276, 1993. URL <https://doi.org/10.1177/014662169301700307>. [p]
- S. Balafas, W. Krijnen, and E. Wit. *mudfold: Multiple UniDimensional unFOLDing*, 2019. URL <https://CRAN.R-project.org/package=mudfold>. R package version 1.1.2. [p]
- M. Boukes and H. G. Boomgaarden. Soft news with hard consequences? introducing a nuanced measure of soft versus hard news exposure and its relationship with political cynicism. *Communication Research*, 42(5):701–731, 2015. URL <https://doi.org/10.1177/0093650214537520>. [p]
- S. V. Buuren, J. P. Brand, C. G. Groothuis-Oudshoorn, and D. B. Rubin. Fully conditional specification in multivariate imputation. *Journal of Statistical Computation and Simulation*, 76(12):1049–1064, 2006. URL <https://doi.org/10.1080/10629360600810434>. [p]
- A. Canty and B. D. Ripley. *boot: Bootstrap R (S-Plus) Functions*, 2017. R package version 1.3-20. [p]
- R. P. Chalmers. mirt: A multidimensional item response theory package for the R environment. *Journal of Statistical Software*, 48(6):1–29, 2012. URL <https://doi.org/10.18637/jss.v048.i06>. [p]
- O. Chernyshenko, S. Stark, F. Drasgow, and B. Roberts. Constructing personality scales under the assumptions of an ideal point response process: Toward increasing the flexibility of personality measures. *Psychological assessment*, 19:88–106, 04 2007. URL <https://doi.org/10.1037/1040-3590.19.1.88>. [p]
- Y.-J. Choi and A. Asilkalkan. R packages for item response theory analysis: Descriptions and features. *Measurement: Interdisciplinary Research and Perspectives*, 17(3):168–175, 2019. URL <https://doi.org/10.1080/15366367.2019.1586404>. [p]
- C. H. Coombs. *A Theory Of Data*. Wiley, 1964. ISBN 978-0471171140. [p]
- L. B. D. R. Cox. On a discriminatory problem connected with the works of Plato. *Journal of the Royal Statistical Society. Series B (Methodological)*, 21(1):195–200, 1959. ISSN 00359246. URL <https://doi.org/10.1111/j.2517-6161.1959.tb00329.x>. [p]
- G. J. De Jong and T. van Tilburg. Manual of the loneliness scale. *Amsterdam: VU University Amsterdam*, 1999. [p]
- J. de Jong-Gierveld and F. Kamphuls. The development of a rasch-type loneliness scale. *Applied psychological measurement*, 9(3):289–299, 1985. URL <https://doi.org/10.1177/014662168500900307>. [p]
- J. de Leeuw and P. Mair. Multidimensional scaling using majorization: SMACOF in R. *Journal of Statistical Software*, 31(3):1–30, 2009. URL <https://doi.org/10.18637/jss.v031.i03>. [p]
- B. Efron et al. Bootstrap methods: Another look at the jackknife. *The Annals of Statistics*, 7(1):1–26, 1979. URL <https://doi.org/10.1214/aos/117634455>. [p]
- T. R. Finserås, S. Pallesen, R. A. Mentzoni, E. Krossbakken, D. L. King, and H. Molde. Evaluating an internet gaming disorder scale using mokken scaling analysis. *Frontiers in Psychology*, 10:911, 2019. ISSN 1664-1078. URL <https://doi.org/10.3389/fpsyg.2019.00911>. [p]

- J. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):1–22, 2010. URL <https://doi.org/10.18637/jss.v033.i01>. [p]
- L. Guttman. A basis for scaling qualitative data. *American Sociological Review*, 9(2):139–150, 1944. ISSN 00031224. URL <https://doi.org/10.2307/2086306>. [p]
- R. Häggli. *Role of Dialogue in Public Opinion Formation*, pages 187–222. Springer International Publishing, Cham, 2020. ISBN 978-3-030-26582-3. URL https://doi.org/10.1007/978-3-030-26582-3_8. [p]
- H. Hoijtink. The measurement of latent traits by proximity items. *Applied psychological measurement*, 15(2):153–169, 1991. URL <https://doi.org/10.1177/014662169101500205>. [p]
- H. Hoijtink. Item response models for nonmonotone items. In K. Kempf-Leonard, editor, *Encyclopedia of Social Measurement*, pages 373 – 378. Elsevier, New York, 2005. ISBN 978-0-12-369398-3. URL <https://doi.org/10.1016/B0-12-369398-5/00464-3>. [p]
- M. S. Johnson. Nonparametric estimation of item and respondent locations from unfolding-type items. *Psychometrika*, 71(2):257–279, 2006. URL <https://doi.org/10.1007/s11336-003-1098-9>. [p]
- M. S. Johnson and B. W. Junker. Using data augmentation and markov chain monte carlo for the estimation of unfolding response models. *Journal of Educational and Behavioral Statistics*, 28(3):195–230, 2003. URL <https://doi.org/10.3102/10769986028003195>. [p]
- S. Karlin. *Total positivity*, volume 1. Stanford University Press, 1968. ISBN 978-0804703147. [p]
- C. P. Knipscheer, J. d. Jong-Gierveld, T. G. van Tilburg, P. A. Dykstra, et al. Living arrangements and social networks of older adults. *Amsterdam: VU University Amsterdam*, 1995. [p]
- P. Lee, S.-H. Joo, S. Stark, and O. S. Chernyshenko. Ggum-rank statement and person parameter estimation with multidimensional forced choice triplets. *Applied Psychological Measurement*, 43(3):226–240, 2019. URL <https://doi.org/10.1177/0146621618768294>. [p]
- R. J. Little and D. B. Rubin. Statistical analysis with missing data. *New York: Wiley*, 1987, 1987. URL <https://doi.org/10.1002/9781119013563>. [p]
- C.-W. Liu and W.-C. Wang. A general unfolding irt model for multiple response styles. *Applied Psychological Measurement*, 43(3):195–210, 2019. URL <https://doi.org/10.1177/0146621618762743>. [p]
- J. Loevinger. The technic of homogeneous tests compared with some aspects of "scale analysis" and factor analysis. *Psychological bulletin*, 45(6):507, 1948. URL <https://doi.org/10.1037/h0055827>. [p]
- G. Luo. A class of probabilistic unfolding models for polytomous responses. *Journal of Mathematical Psychology*, 45(2):224 – 248, 2001. ISSN 0022-2496. URL <https://doi.org/10.1006/jmps.2000.1310>. [p]
- G. Luo, D. Andrich, and I. Styles. The jml estimation of the generalised unfolding model incorporating the latitude of acceptance parameter. *Australian Journal of Psychology*, 50(3):187–198, 1998. URL <https://doi.org/10.1080/00049539808258795>. [p]
- M. D. Maraun and N. T. Rossi. The extra-factor phenomenon revisited: Unidimensional unfolding as quadratic factor analysis. *Applied Psychological Measurement*, 25(1):77–87, 2001. URL <https://doi.org/10.1177/01466216010251006>. [p]

- A. Maydeu-Olivares, A. Hernández, and R. P. McDonald. A multidimensional ideal point item response theory model for binary data. *Multivariate Behavioral Research*, 41(4):445–472, 2006. URL https://doi.org/10.1207/s15327906mbr4104_2. PMID: 26794914. [p]
- R. J. Mokken. *A theory and procedure of scale analysis: With applications in political research*, volume 1. Walter de Gruyter, 1971. ISBN 978-3-11-081320-3. [p]
- R. J. Mokken. Nonparametric models for dichotomous responses. In W. J. van der Linden and R. K. Hambleton, editors, *Handbook of Modern Item Response Theory*, pages 351–367. Springer New York, New York, NY, 1997. ISBN 978-1-4757-2691-6. URL https://doi.org/10.1007/978-1-4757-2691-6_20. [p]
- Y. Noel. A beta unfolding model for continuous bounded responses. *Psychometrika*, 79(4):647–674, Oct 2014. ISSN 1860-0980. URL <https://doi.org/10.1007/s11336-013-9361-1>. [p]
- W. J. Post. *Nonparametric Unfolding Models: A Latent Structure Approach*. M & T series. DSWO Press, 1992. ISBN 978-9066950641. [p]
- W. J. Post and T. A. Snijders. Nonparametric unfolding models for dichotomous data. *Methodika*, 1993. [p]
- W. J. Post, M. A. van Duijn, and B. van Baarsen. Single-peaked or monotone tracelines? on the choice of an irt model for scaling data. In *Essays on item response theory*, pages 391–414. Springer, 2001. URL https://doi.org/10.1007/978-1-4613-0169-1_21. [p]
- G. Rasch. On general laws and the meaning of measurement in psychology. In *Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability, Volume 4: Contributions to Biology and Problems of Medicine*, pages 321–333, Berkeley, Calif., 1961. University of California Press. URL <http://projecteuclid.org/euclid.bsmsp/1200512895>. [p]
- W. D. Rinkel, M. H. Aziz, J. W. Van Neck, M. C. Cabezas, L. A. van der Ark, and J. H. Coert. Development of grading scales of pedal sensory loss using mokken scale analysis on the rotterdam diabetic foot study test battery data. *Muscle & Nerve*, 60(5):520–527, 2019. URL <https://doi.org/10.1002/mus.26628>. [p]
- J. S. Roberts and J. E. Laughlin. A unidimensional item response model for unfolding responses from a graded disagree-agree response scale. *Applied Psychological Measurement*, 20(3):231–255, 1996. URL <https://doi.org/10.1177/014662169602000305>. [p]
- J. S. Roberts and V. M. Thompson. Marginal maximum a posteriori item parameter estimation for the generalized graded unfolding model. *Applied Psychological Measurement*, 35(4):259–279, 2011. URL <https://doi.org/10.1177/0146621610392565>. [p]
- J. S. Roberts, J. R. Donoghue, and J. E. Laughlin. A general item response theory model for unfolding unidimensional polytomous responses. *Applied Psychological Measurement*, 24(1):3–32, 2000. URL <https://doi.org/10.1177/01466216000241001>. [p]
- J. S. Roberts, H.-r. Fang, W. Cui, and Y. Wang. Ggum2004: A windows-based program to estimate parameters in the generalized graded unfolding model. *Applied Psychological Measurement*, 2006. URL <https://doi.org/10.1177/0146621605280141>. [p]
- D. B. Rubin. Inference and missing data. *Biometrika*, 63(3):581–592, 1976. URL <https://doi.org/10.1093/biomet/63.3.581>. [p]
- K. Sijtsma. Nonparametric item response theory models. In K. Kempf-Leonard, editor, *Encyclopedia of Social Measurement*, pages 875 – 882. Elsevier, New York, 2005. ISBN 978-0-12-369398-3. URL <https://doi.org/10.1016/B0-12-369398-5/00459-X>. [p]
- K. Sijtsma and B. W. Junker. Item response theory: Past performance, present developments, and future expectations. *Behaviormetrika*, 33(1):75–102, 2006. URL <https://doi.org/10.2333/bhmk.33.75>. [p]

- S. Stark, O. Chernyshenko, F. Drasgow, and B. Williams. Examining assumptions about item responding in personality assessment: Should ideal point methods be considered for scale development and scoring? *Journal of Applied Psychology*, 91(1):25–39, 2006. ISSN 0021-9010. URL <https://doi.org/10.1037/0021-9010.91.1.25>. [p]
- I. Sulis and M. Porcu. Handling missing data in item response theory. assessing the accuracy of a multiple imputation procedure based on latent class analysis. *Journal of Classification*, 34(2):327–359, Jul 2017. ISSN 1432-1343. URL <https://doi.org/10.1007/s00357-017-9220-3>. [p]
- J. N. Tendeiro and S. Castro-Alvarez. *GGUM: Generalized Graded Unfolding Model*, 2018. URL <https://CRAN.R-project.org/package=GGUM>. R package version 0.3.3. [p]
- L. L. Thurstone. A law of comparative judgment. *Psychological review*, 34(4):273, 1927. URL <https://doi.org/10.1037/h0070288>. [p]
- L. L. Thurstone. Attitudes can be measured. *American journal of Sociology*, pages 529–554, 1928. URL <https://doi.org/10.1086/214483>. [p]
- R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996. URL <https://doi.org/10.1111/j.2517-6161.1996.tb02080.x>. [p]
- S. van Buuren and K. Groothuis-Oudshoorn. mice: Multivariate imputation by chained equations in r. *Journal of Statistical Software*, 45(3):1–67, 2011. URL <https://doi.org/10.18637/jss.v045.i03>. [p]
- L. A. Van der Ark. Mokken scale analysis in R. *Journal of Statistical Software*, 20(11):1–19, 2007. URL <https://doi.org/10.18637/jss.v020.i11>. [p]
- L. A. Van der Ark. New developments in mokken scale analysis in R. *Journal of Statistical Software*, 48(5):1–27, 2012. URL <https://doi.org/10.18637/jss.v048.i05>. [p]
- W. Van Schuur. Stochastic unfolding. In *Sociometric research*, pages 137–158. Springer, 1988. URL https://doi.org/10.1007/978-1-349-19051-5_9. [p]
- W. H. Van Schuur. *Structure in Political Beliefs: A New Model for Stochastic Unfolding with Application to European Party Activities*. CT Press, 1984. ISBN 978-9070758042. [p]
- W. H. Van Schuur. Nonparametric unidimensional unfolding for multicategory data. *Political Analysis*, 4:41–74, 1992. URL <https://doi.org/10.1093/pan/4.1.41>. [p]
- G. R. Warnes, B. Bolker, and T. Lumley. *gtools: Various R Programming Tools*, 2015. URL <https://CRAN.R-project.org/package=gtools>. R package version 3.5.0. [p]
- H. Wickham. Reshaping data with the reshape package. *Journal of Statistical Software*, 21(12):1–20, 2007. URL <https://doi.org/10.18637/jss.v021.i12>. [p]
- 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>. [p]
- S. Wood. *Generalized Additive Models: An Introduction with R*. Chapman and Hall/CRC, 2 edition, 2017. [p]
- S. N. Wood. Fast stable restricted maximum likelihood and marginal likelihood estimation of semiparametric generalized linear models. *Journal of the Royal Statistical Society (B)*, 73(1):3–36, 2011. URL <https://doi.org/10.1111/j.1467-9868.2010.00749.x>. [p]
- A. Zeileis and G. Grothendieck. zoo: S3 infrastructure for regular and irregular time series. *Journal of Statistical Software*, 14(6):1–27, 2005. URL <https://doi.org/10.1007/978-0-387-98141-3>. [p]

Spyros E. Balafas

*Bernoulli Institute for Mathematics, Computer Science & Artificial Intelligence
University of Groningen (RUG)
Bernoulliborg, Rm. 460, Nijenborgh 9
9747 AG Groningen
The Netherlands
s.balafas@rug.nl*

Wim P. Krijnen

*Bernoulli Institute for Mathematics, Computer Science & Artificial Intelligence
University of Groningen (RUG)
Bernoulliborg, Nijenborgh 9
9747 AG Groningen
The Netherlands
w.p.krijnen@rug.nl*

Wendy J. Post

*Orthopedagogy & Clinical Educational Science
University of Groningen (RUG)
Grote Rozenstraat 38
9712 TJ Groningen
The Netherlands
w.j.post@rug.nl*

Ernst C. Wit

*Faculty of Science & Informatics
Universita della Svizzera Italiana (USI)
Via Buffi 13
6900 Lugano
Switzerland
wite@usi.ch*

Mapping Smoothed Spatial Effect Estimates from Individual-Level Data: MapGAM

by Lu Bai, Daniel L. Gillen, Scott M. Bartell, Verónica M. Vieira

Abstract We introduce and illustrate the utility of **MapGAM**, a user-friendly R package that provides a unified framework for estimating, predicting and drawing inference on covariate-adjusted spatial effects using individual-level data. The package also facilitates visualization of spatial effects via automated mapping procedures. **MapGAM** estimates covariate-adjusted spatial associations with a univariate or survival outcome using generalized additive models that include a non-parametric bivariate smooth term of geolocation parameters. Estimation and mapping methods are implemented for continuous, discrete, and right-censored survival data. In the current manuscript, we summarize the methodology implemented in **MapGAM** and illustrate the package using two example simulated datasets: the first considering a case-control study design from the state of Massachusetts and the second considering right-censored survival data from California.

Introduction

In spatial epidemiology studies, mapping crude and adjusted spatial distributions of disease risk is a useful tool for identifying risk factors of public health concern (Elliott and Wartenberg, 2004). The underlying (or crude) geographic pattern of disease is often what is observed by public health practitioners, but these patterns may be due to important spatially-varying predictors such as socioeconomic status, race/ethnicity, or environmental exposures. Individual-level spatial analyses can provide insight regarding disease risk by adjusting for these variables without aggregation bias (also known as ecological bias). Disease risks often have complex spatial patterns that are subject to high variability due to sparsity. Smoothing provides an efficient method to deal with these issues by borrowing strength from adjacent observations to reduce variability while allowing for non-parametric flexibility when estimating the spatial distribution of risk. Generalized additive models (GAMs), originally proposed by Hastie and Tibshirani (1986), are common model-based approaches for mapping point-based epidemiologic data (Webster et al., 2006; Vieira et al., 2008; Baker et al., 2011; Akullian et al., 2014; Bristow et al., 2014; Hoffman et al., 2015). GAMs provide a unified statistical framework that allows for the adjustment of individual-level risk factors when evaluating spatial variability in a flexible way. The flexibility provided by GAMs, together with the intuitive nature of many smoothing techniques, make them an ideal choice for modeling complex spatial associations.

There are a number of R packages implementing GAMs and related models (R Core Team, 2015). The **gam** package (Hastie, 2004) provides an implementation of the GAM framework of Hastie and Tibshirani (1986) by providing two types of commonly used smoothing methods: cubic loess smoothing splines for univariate variables and local kernel smoothing (LOESS) for multivariate variables. The **mgcv** (Wood, 2009; Breslow and Clayton, 1993) package implements cubic smoothing splines and tensor product smooths, an extension of cubic splines to multi-dimensions. **mgcv** also provides various criterion to aid in the selection of model complexity via the choice of effective degrees of freedom and provides functions to fit generalized additive mixed effects models (GAMMs) for correlated data. Package **gamlss** (Rigby and Stasinopoulos, 2005; Stasinopoulos and Rigby, 2007) implements an extension of the GAM that incorporates selected distributions outside of the exponential family. With respect to censored survival data, parametric additive models can be fit using both the **gamlss.cens** package (Stasinopoulos et al., 2015) and the **VGAM** package (W., 2007). Bayesian inferences for the spatial analysis of survival data based on the parametric proportional hazards model are implemented in package **spatsurv** (Taylor et al., 2016; Taylor and Rowlingson, 2014). However, parametric models assume a full distribution of the survival times, and misspecifying the distribution may yield bias for estimates. Cox proportional hazards models, which are semi-parametric without specifying a form for underlying hazard function, are more robust for survival analysis including multiple adjusted variables.

A variety of R packages incorporate Cox proportional hazards models and spatial smoothing term. The R interface to **BayesX** (Umlauf et al., 2015; Belitz et al., 2016; Kneib et al., 2014), **R2BayesX** (Umlauf et al., 2016), provides survival spatial analysis based on structured additive

models (STAR) without specifying the baseline hazard. **mboost**(Hothorn et al., 2016) implements boosting for optimizing penalized likelihood function, and as an extention of **mboost**, **gamboost-MSM**(Reulen, 2014) provides estimates for multistate models. **mgcv** also incorporates 'coxph' family in the model fitting. However these packages use cubic, B- and/or P- splines as smoothing methods; none offer LOESS. LOESS adapts to varying data densities by defining a local neighborhood based on a fixed proportion ("span") of the observations; this is especially useful for spatial analyses as population densities typically vary within any study region.

Therefore, none of the above packages provide an implementation of the Cox proportional hazards additive model for censored survival data that allows for multivariate loess smoothing of covariates such as geolocation parameters, despite the fact that spatial effect estimation in the context of survival outcomes is of great interest in epidemiology studies (Henderson et al., 2002; Bristow et al., 2014). Moreover, displaying spatial predictions on a map with irregular geographic boundaries is a non-trivial effort, often handled by exporting statistical predictions to separate specialized geographic information system (GIS) software such as ArcGIS that requires a paid user license (Webster et al., 2006; Vieira et al., 2008) or by omitting geographic boundaries altogether(Akullian et al., 2014). At best, these limitations and complexities pose a significant barrier to researchers not already well versed in both GAMs and GIS methods and at worst may lead to reporting errors due to the inefficient transfer of estimates between separate software packages.

To address the above deficiencies of current software, **MapGAM** was built to provide a single R package that allows for estimating, predicting, and visualizing covariate-adjusted spatial effects using individual-level data. The package estimates covariate-adjusted spatial associations with a univariate or survival outcome via GAMs that include a non-parametric bivariate smooth term of geolocation parameters. Estimation and mapping methods are implemented for continuous, discrete, and right-censored survival data. In addition, support functions for efficient control sampling in case-control studies and inferential procedures for testing global and pointwise spatial effects are implemented. We have found that a unified system for estimating and visualizing covariate-adjusted spatial effects on outcomes arising from the most commonly encountered epidemiologic study designs greatly facilitates efficient and reproducible analyses in these settings.

This article serves as an introduction and illustration of the **MapGAM** package. The remainder of the manuscript is organized as follows: Section 2 provides an overview of the methodology implemented in **MapGAM** for estimating and visualizing spatial effects in the context of a generalized additive model for continuous, binary or count outcome data. An illustrative example using **MapGAM** to analyze hypothetical case-control data from the state of Massachusetts is also provided. Section 3 considers estimating spatial effects on right-censored survival times via a Cox proportional hazards additive model. The estimation procedures implemented in **MapGAM** are provided and a brief simulation study considers the performance of the proposed fitting methods in various settings. In Section 4 we consider inference procedures associated with spatial modeling and illustrate how to use the package to perform a global test of a spatial effect and calculate confidence intervals for predictions at each spatial prediction point. Section 5 concludes with discussion of the utility of the **MapGAM** package and considers possible extensions of the package in future research.

Spatial effect on a univariate outcome

We consider estimating and visualizing covariate-adjusted spatial effects in the context of a GAM for continuous, binary or count outcome data. The spatial effect can be estimated by fitting a GAM model with a bivariate smoothing term for the two geolocation parameters. Typical models will also include additional adjustment for demographic characteristics and other risk factors that may serve as potential confounding factors in the association between location and the outcome of interest.

Generalized additive model

We consider modeling observations that are distributed on a map with u_i and v_i denoting the geographical parameters for the i^{th} observation, $i = 1, \dots, n$. Let Y_i denote the outcome and X_i denote a vector of adjustment covariates. Further suppose that the distribution of the outcome belongs to the exponential family. The GAM then assumes that

$$g(\mu_i) = \eta_i = \beta_0 + X_i^\top \beta + f(u_i, v_i), \quad (1)$$

where $g(\cdot)$ is the link function for mean of the outcome $\mu_i = \mathbb{E}[Y_i]$ and the variance of the outcome is defined by the assumed probability model and denoted as $V_i \equiv \text{Var}(Y_i) = V(\mu_i, \phi)$; a function of the mean and nuisance parameter ϕ . β denotes a vector of coefficients associated with adjustment covariate X_i and $f(u_i, v_i)$ represents the spatial effects, which is a nonlinear function of location.

When fitting the model, we separate the spatial effect into parametric and nonparametric portions: $f(u_i, v_i) = \gamma_1 u_i + \gamma_2 v_i + s_i$, and the model becomes

$$g(\mu_i) = \eta_i = \beta_0 + \tilde{X}_i^\top \tilde{\beta} + s_i, \quad (2)$$

where $\tilde{X}_i = [X_i, u_i, v_i]$ and $\tilde{\beta} = [\beta^\top, \gamma_1, \gamma_2]^\top$. The parametric part of the spatial effect is fit jointly with other adjustment variables using least squares, while the nonparametric term is fit using a nonparametric smoother. To ensure identifiability, we constrain the model so that $\sum_i s_i = 0$.

A local scoring procedure (Hastie and Tibshirani, 1986) is used to fit the model. Let l denote the log-likelihood function based upon one observations $Y = [Y_1, \dots, Y_n]^\top$, which is a function of $\eta = [\eta_1, \dots, \eta_n]^\top$. To estimate the parameters of the model we seek to maximize the expected log likelihood:

$$\mathbb{E}(l(\hat{\eta}_i, Y_i)) = \max_{\eta_i} \mathbb{E}(l(\eta_i, Y_i)), \text{ for } i = 1, \dots, n \quad (3)$$

where the expectation is taken over the joint distribution of X and Y . This has intuitive appeal since it seeks to choose a model that maximizes the likelihood of all possible future observations. Under standard regularity conditions (namely the ability to interchange integration and differentiation), we obtain

$$\mathbb{E}[dl/d\eta_i]_{\hat{\eta}_i} = 0, \quad (4)$$

While there is no general closed form solution to Eq.(4), a first-order Taylor series expansion leads to an iterative estimating procedure given by

$$\eta_i^{new} = \eta_i^{old} - \mathbb{E}[dl/d\eta_i]|_{\eta_i^{old}} / \mathbb{E}[d^2 l/d\eta_i^2]|_{\eta_i^{old}}, \quad (5)$$

which is equivalent to

$$\eta_i^{new} = \mathbb{E}\left[\eta_i - \frac{dl/d\eta_i}{\mathbb{E}[d^2 l/d\eta_i^2]}\right]_{\eta_i^{old}}. \quad (6)$$

In the exponential family case, we can compute the first and second derivatives of the expected log likelihood as

$$\frac{dl}{d\eta_i} = (Y_i - \mu_i)V_i^{-1}\left(\frac{d\mu_i}{d\eta_i}\right), \quad (7)$$

and

$$\frac{d^2 l}{d\eta_i^2} = (Y_i - \mu_i)V_i^{-1}\left(\frac{d}{d\eta_i}\right)\left[V_i^{-1}\left(\frac{d\mu_i}{d\eta_i}\right)\right] - \left(\frac{d\mu_i}{d\eta_i}\right)^2 V_i^{-1}. \quad (8)$$

Then taking the expectation (conditional on X) of Eq.(8) we obtain

$$E\left[\left.\left(\frac{d^2 l}{d\eta_i^2}\right)\right| X\right] = -\left(\frac{d\mu_i}{d\eta_i}\right)^2 V_i^{-1}. \quad (9)$$

Hence η_i is updated in the GLM case by

$$\eta_i^{new} = E\left[\eta_i + (Y_i - \mu_i)\left.\left(\frac{d\eta_i}{d\mu_i}\right)\right|_{\eta_i^{old}, \mu_i^{old}}\right]. \quad (10)$$

Further, letting $Y_w^{old} = [Y_{w1}^{old}, \dots, Y_{wn}^{old}]^\top$ denote the working response computed in terms of η^{old} and μ^{old} and given by

$$Y_{wi}^{old} = \eta_i + (Y_i - \mu_i)\left.\left(\frac{d\eta_i}{d\mu_i}\right)\right|_{\eta_i^{old}, \mu_i^{old}}, \quad (11)$$

we obtain from Eq.(2), Eq.(10), and Eq.(11),

$$E[Y_{wi}^{old}] = \beta_0^{new} + \tilde{X}_i^\top \tilde{\beta}^{new} + s_i^{new}. \quad (12)$$

The coefficients $\tilde{\beta}_0^{new}$, $\tilde{\beta}^{new}$ and nonparametric term, s , must be estimated in order to obtain an updated value of η^{new} in Eq.(10). If no parametric linear predictor term is included in the model (beyond the spatial smoothing term), the updated s^{new} can be estimated by regressing the working response Y_w^{old} on a bivariate smoother for u and v . However, with the parametric linear predictor

term included in the model, the backfitting algorithm can be used to update β_0 , $\tilde{\beta}$ and s , as is done in the **gam** package. Specifically, we begin by defining $W = [W_1, \dots, W_n]^\top$ as

$$W_i = (d\mu_i/d\eta_i)^2 V_i^{-1} |_{\eta_i^{old}, \mu_i^{old}} \quad (13)$$

and initializing $s = 0$. The backfitting procedure then loops through the following three steps until the mean squared error does not further decrease relative to a defined convergence criteria:

1. Update β_0 and $\tilde{\beta}$ by fitting a linear regression model with $Y_w^{old} - s$ as the response and corresponding weight W ;
2. Update \hat{s} by regressing response $Y_w^{old} - \beta_0 - \tilde{\mathbf{X}}\tilde{\beta}$ on a bivariate smoother for u and v with weight W ;
3. Calculate $s = \hat{s} - \hat{s}^\top \hat{s}$.

Thus, the general algorithm for fitting a generalized additive model within the **MapGAM** package is:

1. Initialize $s = 0$. Initialize β_0 and $\tilde{\beta}$ by fitting a generalized linear regression model with all the adjusted covariates and geolocation parameters included in the model (ie. omitting s).
2. Loop:
 - (a) With the current estimated β_0 , $\tilde{\beta}$ and s , calculate η^{old} as well as working response Y_w^{old} and W using Eq.(2), Eq.(11) and Eq.(13) respectively.
 - (b) Update β_0 , $\tilde{\beta}$ and s via the backfitting algorithm.
3. Repeat 2. until convergence.

A locally weighted scatterplot smoother (LOESS) (Cleveland, 1979, 1981; Cleveland and Devlin, 1988) is utilized as the bivariate smoothing function for the two geolocation parameters u and v in the **MapGAM** package. The smoothing parameter defining the neighborhood used to select the K nearest observations points for smoothing may be user specified or automatically chosen by minimizing AIC (Webster et al., 2006).

Estimating and mapping a spatial effect

In the **MapGAM** package, typical spatial applications will start with the **predgrid()** function to create a regular grid of points within the study area, potentially restricted to points within optional map boundaries (e.g., a country, state, or regional map obtained from the **maps** package or imported from a shapefile). Crude or covariate-adjusted odds ratios, hazard ratios, or other effect estimates are then obtained for each grid point using the **modgam()** function to smooth by geolocation. **modgam()** provides compatible and flexible interfaces, acting as a wrapper function to the **gam()** function in the **gam** package. Specifically, the model can be specified via a formula statement, or for users less familiar with writing model formulas in R, the formula can be omitted in which case the model is specified implicitly by structuring the data so that the first column of the data represents the outcome to be modeled (or the first two columns for survival objects), the next two columns represent the parameters for geolocation, and the remaining columns represent the adjustment covariates to be included in the model. With the model specified, **modgam()** proceeds by calling the **gam()** function to estimate model parameters, then calls **mypredict.gam()** to generate predictions for the specified grid. The **optspan()** function can be used to find an optimal span size (proportion of data size included in the neighborhood) for the LOESS smoother. Optionally, the **modgam()** function can call **optspan()** to choose the optimal span for fitting the model in an automated fashion.

Considering the estimated spatial effect $f(u_i, v_i)$ for the i^{th} location, researchers are often interested in the spatial effect difference (or ratio, log-ratio) comparing each location to a defined reference. To obtain spatial effect estimates, one can specify **type="spatial"**, then **modgam()** provides three options for the choice of reference: the median of $f(u_i, v_i)$, $i = 1, \dots, n$, the mean of $f(u_i, v_i)$, $i = 1, \dots, n$, or an estimated spatial effect value at a user-specified geolocation. Alternatively, specifying **reference="none"** will produce prediction estimates based upon the linear predictor for each covariate combination in the prediction dataset (including the model intercept). To produce estimates of effects for all adjustment covariates, the option **type="all"** may be specified. The result of **modgam()** is an object of class **modgam()** that can be summarized by class-defined printing, summarizing and plotting methods. Specifically, a heatmap of the predicted values from a fitted model can be generated using either the **colormap()** or **plot()** functions. For tailored plots, the **trimdata()** and **sampcont()** functions can be used to restrict data to those areas within

a specified set of map boundaries and to conduct simple or spatiotemporal stratified sampling from eligible controls—a useful feature for analysis of data from large cohorts.

Application to case-control data from Massachusetts

In this section we present an illustrative example using **MapGAM** to analyze hypothetical case-control data from Massachusetts. **Madata** is a simulated case-control study dataset available in the **MapGAM** package. Contained in the dataset are 90 cases and 910 controls with randomly generated geolocations across Massachusetts, geocoded on a Lambert projection (in meters). **Memap** provides a map of Massachusetts using the same projection. The dataset also contains three randomly generated potential adjustment covariates: smoking, mercury exposure and selenium exposure. A summary of the dataset follows:

```
R> data("MADATA")
R> data("MAMAP")
R> summary(MADATA)

      Case          Xcoord        Ycoord       Smoking
Min.   :0.00   Min.   :35354   Min.   :778430   Min.   :0.000
1st Qu.:0.00   1st Qu.:111465  1st Qu.:869089  1st Qu.:0.000
Median :0.00   Median :183100  Median :891067  Median :0.000
Mean    :0.09   Mean   :175054  Mean   :889081  Mean   :0.177
3rd Qu.:0.00   3rd Qu.:236826  3rd Qu.:919684  3rd Qu.:0.000
Max.    :1.00   Max.   :327861  Max.   :954253  Max.   :1.000

      Mercury        Selenium
Min.   :0.1418   Min.   :0.2049
1st Qu.:0.7206   1st Qu.:0.8573
Median :1.0010   Median :1.1836
Mean   :1.1471   Mean   :1.3590
3rd Qu.:1.4017   3rd Qu.:1.6844
Max.   :5.6298   Max.   :5.8963
```

The geolocations of the observations are shown in Figure 1, which can be generated with the following code:

```
R> plot(MAMAP)
R> points(MADATA$Xcoord, MADATA$Ycoord, col = MADATA$Case + 1)
```

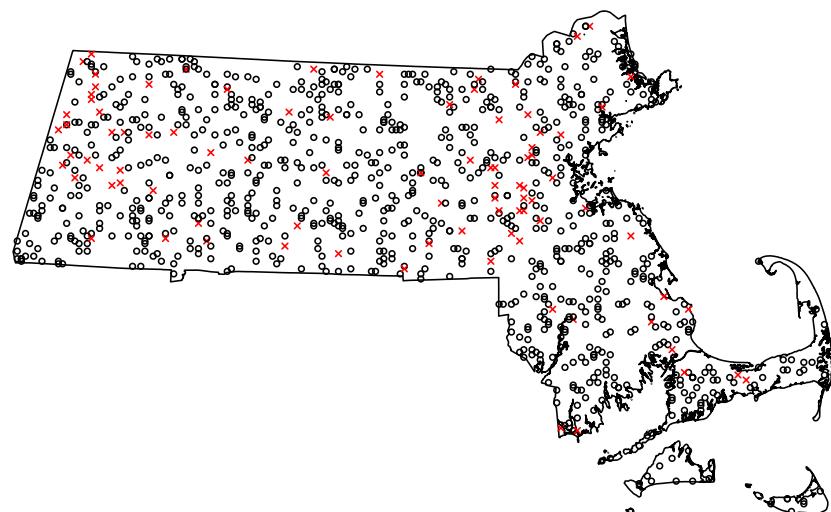


Figure 1: Map of Massachusetts that indicates the location of cases and controls. Data are contained in the MADATA dataset. Depicted are controls (black, ‘o’) and cases (red, ‘x’).

We first start with generating a prediction grid for the map using `predgrid()`.

```
library("PBSmapping")
R> gamgrid <- predgrid(MADATA, map = MAMAP)
```

After defining a prediction grid, `modgam()` is used to fit a GAM model based on the `MAdata` and generate predictions on the defined grid. A formula expression indicates that the indicator `Case` is specified as the response, and two spatial parameters `Xcoord` and `Ycoord` are included in `lo()` to specify a geospatial smoothing term. In addition, potential confounders `Smoking`, `Mercury` and `Selenium` are also adjusted for in the model as linear terms. Argument `sp` is used to specify the span size for the spatial smoothing term. A specification of `sp = null` (the default) implies that an optimal span will be selected. Note that if the model formula is not supplied, the data must be structuring so that the outcome is in the first column, the two spatial parameters are in the second and third columns and the adjustment variables are in other columns. In that case, specifying `m="adjusted"` will include all other columns of the data as linear terms in the model and `m="crude"` will fit only the two spatial parameters (the `m` argument is ignored if a model formula is supplied). For this particular example, the resulting call to `modgam()` using the formula statement is given as follows:

```
R> fit1 <- modgam(Case ~ lo(Xcoord, Ycoord) + Smoking + Mercury +
+ Selenium, data = MAdata, rgrid = gamgrid, sp = NULL,
+ type = "spatial", verbose = FALSE)
R> fit1

Call:
modgam(formula = Case ~ lo(Xcoord, Ycoord) + Smoking + Mercury +
  Selenium, data = MAdata, rgrid = gamgrid, sp = NULL, type = "spatial",
  verbose = FALSE)

Model:
Case ~ lo(Xcoord, Ycoord, span = 0.3, degree = 1) + Smoking +
  Mercury + Selenium
Family: binomial Link: logit

Coefficients:
(Intercept)           -6.911648e+00
lo(Xcoord, Ycoord, span = 0.3, degree = 1)Xcoord      2.363118e-06
lo(Xcoord, Ycoord, span = 0.3, degree = 1)Ycoord      4.376156e-06
Smoking                1.533433e+00
Mercury                 5.729589e-01
Selenium               -6.431932e-01
```

Coefficients in the above output represent log-odds ratios. The interpretation of parametric terms remain the same as the usual logistic regression model. For example, we estimate the odds of disease is estimated to be $e^{1.53} = 4.63$ -fold higher when comparing smokers to non-smokers with similar location and exposure to mercury and selenium.

The interpretation of the smoothed spatial terms is best done graphically. A heatmap of the estimated spatial effect predictions (representing the odds ratio comparing the odds at each location to the median odds across all locations) can be generated using the `modgam` plotting routine via a call to the `plot()` function. This in turn relies upon the `colormap()` function defined within `MapGAM`. The resulting heatmap is displayed in Figure 2. The `exp` argument is used to specify whether the heatmap is drawn on the scale of the odds ratio (`exp=TRUE`) or the log odds ratio (`exp=FALSE`).

```
R> plot(fit1, exp = TRUE, MAdap, contours = "response")
```

Estimating spatial effects for right-censored survival data

To quantify spatial effects on censored survival outcomes, `MapGAM` implements a Cox proportional hazards additive model with a bivariate (two geolocation parameters) smoothing term. The incorporation of a bivariate smoother within the Cox model is not, to the best of our knowledge, currently implemented within `R`. In this section, we briefly introduce the methodology implemented in `MapGAM` as an extension of the GAM methods previously discussed for GLMs, provide a limited simulation study to illustrate the validity of the methodology in selected settings and provide an

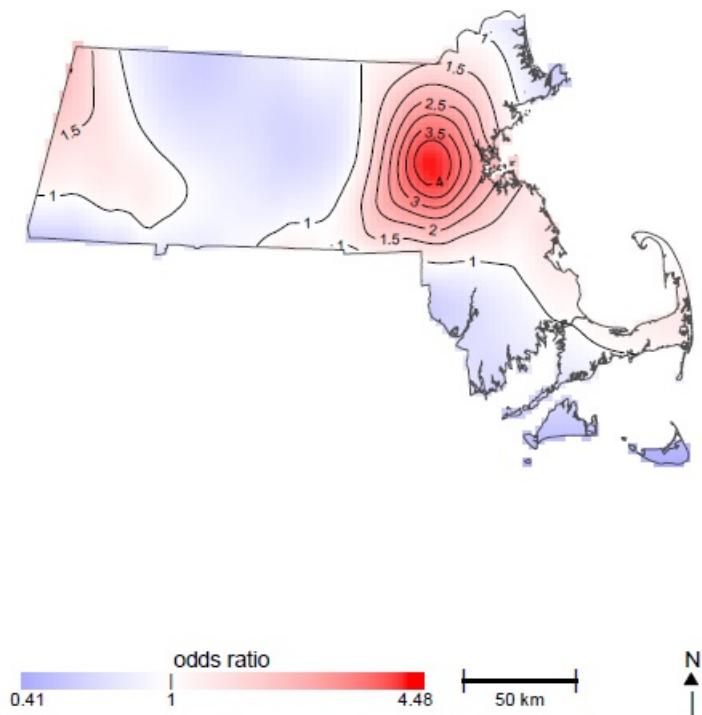


Figure 2: Heatmap of the estimated odds ratio of spatial effect predictions compared to the median estimated odds over all locations. Red colors indicate areas with an increased odds of being a case. Blue colors represent a decreased odds of being a case.

example of applying the **MapGAM** package to estimate spatial effects on censored survival data using hypothetical survival times derived from the state of California.

Fitting the Cox proportional hazards additive model

Suppose we observe right-censored survival data that is distributed on a map with u_i and v_i as the geographical parameters for the i^{th} observation, $i = 1, \dots, n$. Let T_i denote the observed followup time and δ_i denote the indicator of whether or not T_i represents the true failure time for observation i . Further, let \tilde{X}_i be a vector including adjustment covariates X and geolocations (u, v) corresponding to observation i . The Cox proportional hazards additive model used in the **MapGAM** package incorporates a bivariate smoother into the Cox proportional hazards model (Kelsall and Diggle, 1998) as

$$\lambda_i(t) = \lambda_0(t) \exp\{\tilde{X}_i^\top \tilde{\beta} + s_i\}, \quad (14)$$

where $\lambda_i(t)$ represents the hazard function for observation i evaluated at time t and $\lambda_0(t)$ denotes the baseline hazard (ie. the hazard of an observation with all covariate values equal to 0 and location with $s = 0$, where again s is a smooth function of spatial coordinates u and v). Define the linear predictor

$$\eta_i = \tilde{X}_i^\top \tilde{\beta} + s_i. \quad (15)$$

For ease of exposition, consider the case of no tied failure times. Then the partial likelihood and log-partial likelihood are given by

$$PL = \prod_{j \in D} \frac{e^{\eta_j}}{\sum_{k \in R_j} e^{\eta_k}}, \quad (16)$$

and

$$l = \sum_{j \in D} \left[\eta_j - \log \left(\sum_{k \in R_j} e^{\eta_k} \right) \right], \quad (17)$$

respectively, where D represents the set of indices of all unique failures and $R_j = \{k | T_k \geq T_j\}$ denotes the risk set just prior to time T_j . In the event of tied failure times, **MapGAM** defaults to the use of the Efron approximation (Efron, 1977) for the partial likelihood:

$$\prod_{j \in D} \frac{\prod_{k \in F_j} e^{\eta_k}}{\prod_{k=1}^{|F_j|} \left[\sum_{l \in R_j} e^{\eta_l} - \sum_{l \in F_j} e^{\eta_l} (k-1)/|F_j| \right]}, \quad (18)$$

where F_j is the set of indices of the failures occurring at time T_j , and $|F_j|$ is the number of indices in the set F_j .

Letting l denote the log-partial likelihood in Eq.(17), we again seek to solve the maximization problem provided in Eq.(3). The solution can be found iteratively using Eq.(6). We can compute the first and second derivatives of the log-partial likelihood with respect to η_i for observation i as

$$\frac{dl}{d\eta_i} = \delta_i - \sum_{j:i \in R_j} \frac{e^{\eta_i}}{\sum_{k \in R_j} e^{\eta_k}}, \quad (19)$$

and

$$\frac{d^2l}{d\eta_i^2} = - \sum_{j:i \in R_j} \frac{e^{\eta_i}}{\sum_{k \in R_j} e^{\eta_k}} + \sum_{j:i \in R_j} \frac{e^{2\eta_i}}{\left(\sum_{k \in R_j} e^{\eta_k} \right)^2}. \quad (20)$$

The Cox model is a semi-parametric model without any specification for the distribution of the survival times, so it is not possible to calculate a close form for the expectation of the second derivatives for the log partial likelihood as required in Eq.(9). So before updating η , a GAM model can be fitted using the second derivatives as responses to estimate the expectation of the second derivatives of log-partial likelihood.

To this end, we modify the local scoring procedure presented in Section 2.1 by noting that in Eq.(6), with an estimate η^{old} , the new estimate for η can be obtained using the following two steps:

1. Estimate $\mathbb{E}[d^2l/d\eta_i^2]$ by fitting a generalized additive model using $[d^2l/d\eta_i^2]$, $i = 1, \dots, n$ as responses, including the linear predictor of \tilde{X} and a bivariate smoother of geolocation parameters;
2. Estimate η^{new} using the backfitting algorithm described in Section 2.1 with $W_i = -1/\hat{\mathbb{E}}[d^2l/d\eta_i^2]$ as weights and $Y_{wi}^{old} = \eta_i^{old} - [dl/d\eta_i]|_{\eta_i^{old}}/\hat{\mathbb{E}}[d^2l/d\eta_i^2]$ as working responses.

Simulation examples

In this section we assess the performance of our proposed method for fitting the Cox proportional hazards additive model using two simulation studies. In both simulation settings, two spatial parameters (u, v) and adjustment covariate x are generated from a uniform distribution with range from -1 to 1 . Survival times were then simulated from an exponential distribution with a hazard function. The first simulation example assumes a linear effect of all covariates on the log-hazard and that the effect of adjustment covariate x does not interact with the effect of the spatial parameters u and v .

$$\lambda = 0.03 \exp \{ \log(0.7)x + \log(1.2)u + \log(1.5)v \}. \quad (21)$$

In the second simulation example, the spatial parameters have a nonlinear effect on the log-hazard, while the adjustment covariate x has a linear effect that does not interact with the spatial coordinates. The hazard function used in the second simulation example is

$$\lambda = 0.03 \exp \{ \log(0.7)x + \log(1.2)u + \log(1.5)v + \log(0.8)u^2 + \log(1.8)uv \}. \quad (22)$$

The true data-generating heatmaps of the two examples are shown in Figures 3a and 3c, respectively. When we set a seed of 269, with $N = 5000$ sampled data points. The survival times under the first (second) simulation setting range from 0.0011(0.0011) to 316.5(396.8), and have a median of 22.66(24.16). In both settings, censoring times were randomly sampled from a *Uniform*(0, 70) distribution and observed times were taken to be the minimum of the true failure time and censoring time for each observation, yielding approximately 41.6% and 43.9% censoring in scenario 1 and 2, respectively. Code for this simulation is provided in the Appendix. Cox proportional hazards additive models were fit and the spatial effect of the points on an equally-spaced grid (201×201) extended across $u \in [-1, 1]$ and $v \in [-1, 1]$ were predicted using the `modgam` function from the **MapGAM** package. Smoothing span sizes of 0.4 and 0.2 were utilized for scenario 1 and 2, respectively. In each case, these values roughly correspond to the automated span size chosen when optimizing AIC.

Figure 3b and 3d display the estimated spatial effects for example data sets using the first (linear relationship) and second (nonlinear relationship) simulation settings, respectively. Comparing the estimated values in Figures 3b and 3d to the corresponding true data generating values displayed in Figures 3a and 3d, we can see that the additive proportional hazards model implemented in **MapGAM** accurately recreates the true spatial effects (either linear or nonlinear) giving rise to the data. In addition, two scatterplots of the estimated versus true spatial effect are provided in Figure 4a and 4b, again illustrating that the additive proportional hazards method outlined above is able to correctly identify the spatial effects present in the data with minimal bias.

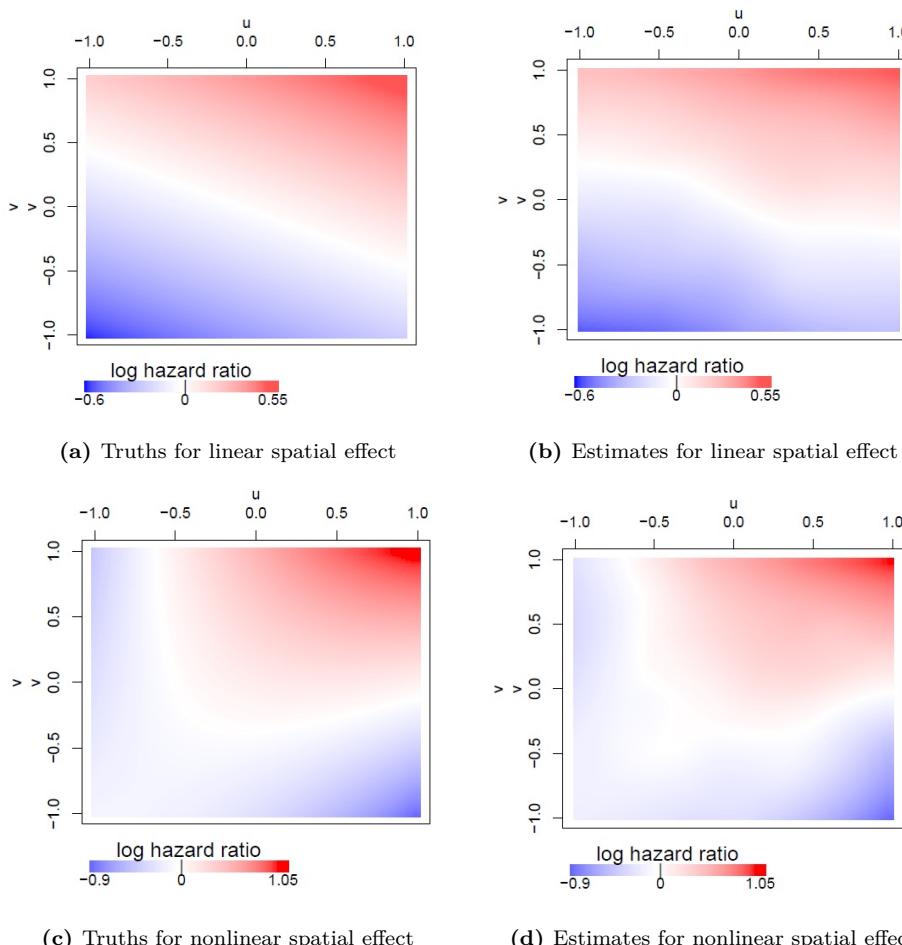


Figure 3: Heatmaps of the the log-hazard ratio comparing the hazard of the location to the median hazard for two simulation examples with 5000 simulated observations. For the first simulation example with linear spatial effect on log-hazards: (a) estimated log-hazard ratio; (b) true log-hazard ratio; For the second simulation example with nonlinear spatial effect on log-hazards: (c) true log-hazard ratio; (d) estimated log-hazard ratio.

Application to right-censored California data

In this section we use the **MapGAM** package to estimate and visualize spatial effects for a dataset simulated from information on censored survival times of California ovarian cancer patients. These are data contained in the object **CADATA** within the **MapGAM** package. The original source is the California advanced-stage invasive epithelial ovarian cancer patients reported to the California Cancer Registry from 1996 to 2006 (Bristow et al., 2014). After removing patients with age <25 and >80 for identifiability reasons, and adding random noise to the geolocation parameters, **CADATA** represents a random draw of size $N = 5,000$ observations from the original dataset. Observed times and failure status were simulated based upon the observed distribution found in the original dataset. Potential covariates available in the dataset include age and insurance type (6 categories

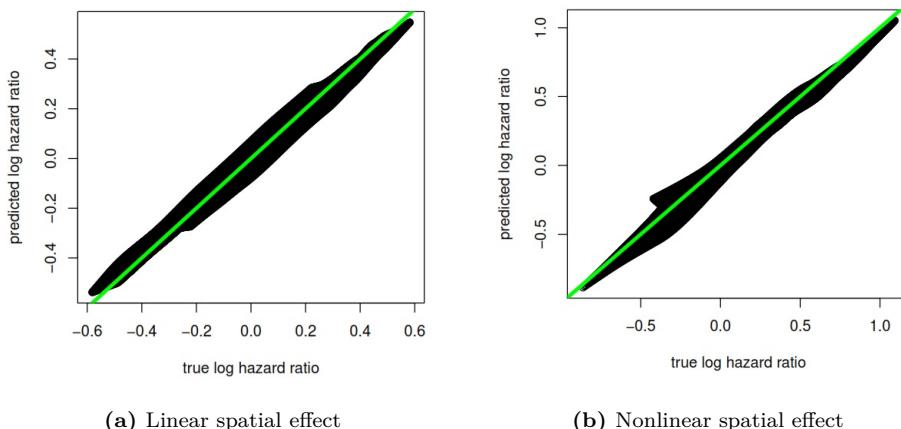


Figure 4: Comparisons of the true log-hazard ratio and the estimated log-hazard ratio for two simulation examples with 5000 simulated observations: (a) result for the first simulation example with linear spatial effect; (b) result for the second simulation example with nonlinear spatial effect.

in total: Managed Care, Medicare, Medicaid, Other Insurance, Not Insured and Unknown). A summary of **CAdata** is as follows:

```
R> data("CAdata")
R> summary(CAdata)

      time          event          X          Y
Min. : 0.004068  Min. :0.0000  Min. :1811375  Min. :-241999
1st Qu.: 1.931247 1st Qu.:0.0000  1st Qu.:2018363  1st Qu.: -94700
Median : 4.749980 Median :1.0000  Median :2325084  Median : -60387
Mean   : 6.496130 Mean  :0.6062  Mean   :2230219  Mean   : 87591
3rd Qu.: 9.609031 3rd Qu.:1.0000  3rd Qu.:2380230  3rd Qu.: 318280
Max.   :24.997764 Max.  :1.0000  Max.   :2705633  Max.   : 770658

      AGE          INS
Min. :25.00  Mcd: 431
1st Qu.:53.00  Mcr:1419
Median :62.00  Mng:2304
Mean   :61.28  Oth: 526
3rd Qu.:71.00  Uni: 168
Max.   :80.00  Unk: 152
```

CAmap is the map file for California State. The geolocations of the observations are plotted in Figure 5.

```
R> data("CAmap")
R> plot(CAmap)
R> points(CAdata$X,CAdata$Y)
```

Below we generate the object **CAgrid** for the state of California using the **predgrid()** function and estimate spatial effects on the relative risk of death from a Cox proportional hazards additive model using the **modgam()** function. As with the previous example, coefficients for parametric terms in the model are interpretable as the would be in a standard (non-GAM) fit of the data. In this case, the coefficients for these terms represent log-hazard ratios. For example, we estimate that the hazard ratio comparing two subpopulations differing in age by 1 year but having similar insurance status is approximately $e^{1.026} = 1.03$. The smoothed spatial terms are again best interpreted graphically. A heatmap of the hazard ratio comparing the estimated hazard at each location to the median hazard across all locations is plotted using the plotting routines defined for **modgam** objects via **plot()**. The resulting heatmap is displayed in Figure 6.

```
R> CAgrid = predgrid(CAdata[, c("X","Y")], map = CAmap,
+   nrow = 186, ncol = 179)
R> fit2 <- modgam(Surv(time, event) ~ AGE + factor(INS) + lo(X, Y),
+   data = CAdata, rgrid = CAgrid, sp = 0.3, verbose = FALSE)
R> plot(fit2, CAmap, exp = T, border.gray = 0.5)
R> fit2
```

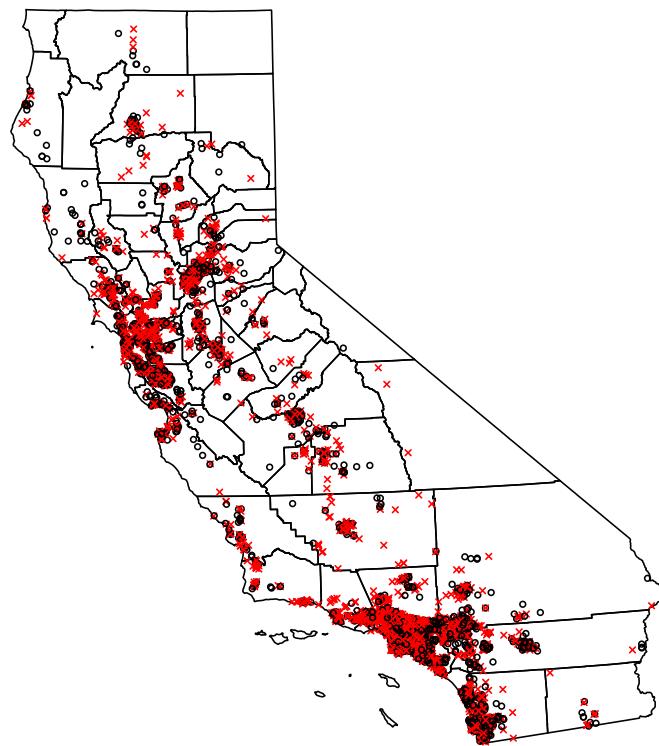


Figure 5: Map of California displaying the geolocations of the observations in CA-data. Depicted in the plot are censored observations (black, ‘o’) and observed event observations (red, ‘x’).

```

Call:
modgam(formula = Surv(time, event) ~ AGE + factor(INS) + lo(X,
Y), data = CAdat, rgrid = CAgid, sp = 0.3, verbose = FALSE)

Model:
Surv(time, event) ~ lo(X, Y) + AGE + factor(INS)
span: 0.3

Coefficients:
AGE factor(INS)Mcr factor(INS)Mng factor(INS)0th factor(INS)Uni
0.02657848      0.03657777      0.05251440      0.16770033      0.26790051
factor(INS)Unk
0.07594159

```

Inference for spatial effects

Making inferences

In addition to providing point estimates associated with each spatial location, **MapGAM** provides pointwise standard errors as well confidence intervals. This inference is returned by the `modgam` function when the option `se.fit=TRUE` is specified. The estimated pointwise standard errors for spatial effects are derived from the sum of two variance curves: one from the parametric terms associated with location, $\gamma_1 u_i + \gamma_2 v_i$, and the other from the non parametric term, s_i ([Chambers and Hastie, 1992](#)). Briefly, variance estimation requires computation of the *operation matrix* G_i for each smooth term s_i , such that $s_i = G_i z$, where z is the working response from the last iteration of the fitting algorithm described in Section 2.1 and is asymptotically distributed as a Gaussian random variable. From this, the covariance matrix for the estimated s_i is given by $G_i \text{Cov}(z) G_i^\top$, which can be estimated by $\hat{\phi} G_i W^{-1} G_i^\top$, where W is a diagonal matrix with elements defined by the weights used in the last iteration of the fitting algorithm and $\hat{\phi}$ is an overdispersion parameter

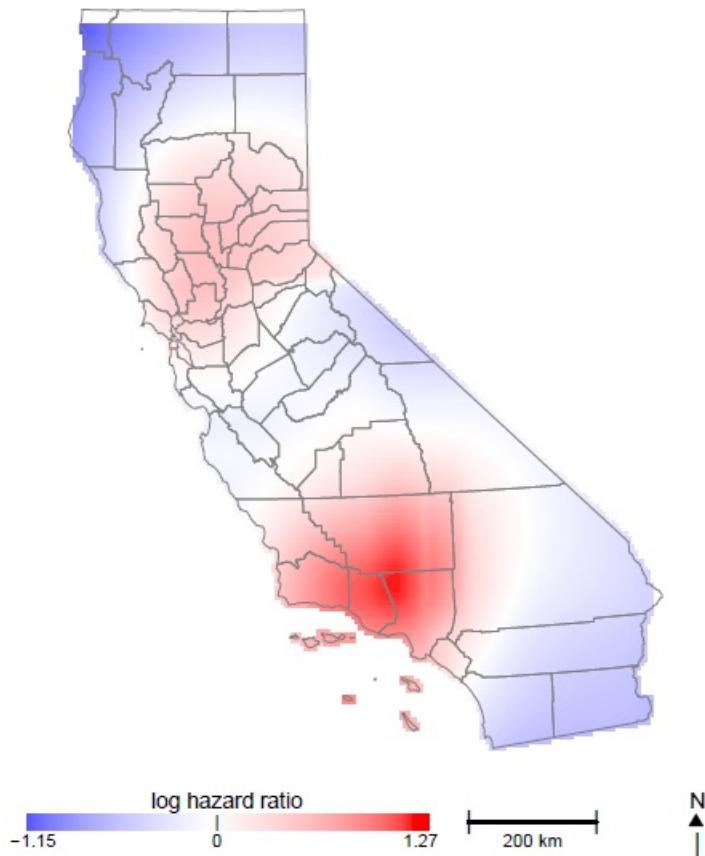


Figure 6: Heatmap of the estimated hazard ratio compared to the median hazard over all locations using the CAdata. Red areas indicate an increased hazard and blue areas indicate a decreased hazard.

estimated using Pearson's Chi square statistic. The operation matrix, G_i , tends to be computationally expensive to obtain for non-parametric or semi-parametric smoothing procedures, and hence approximations are often used when estimating $G_i \hat{Cov}(z) G_i^\top$. One approach is to approximate $\hat{\phi} G_i W^{-1} G_i^\top$ by $\hat{\phi} G_i W^{-1}$, which is generally conservative for non-projection smoothers (Chambers and Hastie, 1992). In this case, G_i can be orthogonally decomposed into $G_i = H_i + N_i$, where H_i can be obtained as the design matrix corresponding to the parametric portion of the linear predictor, and N_i corresponds to the non-parametric portion. Thus, the variance of the estimated smooth term can be approximated via a decomposition of two variance components: (i) the variance from the parametric portion of the linear predictor which captures the correlation all parametric terms that are fitted together, and (ii) the variance from the non-parametric portion of linear predictor reflecting the marginal information obtained in the smoothing terms.

`modgam` conducts a global test for spatial effects via a likelihood ratio test by comparing the deviance between a full model (including the spatial smoother) and a reduced model (omitting the spatial smoother). For the full model, the degrees of freedom of the non-parametric term are computed as $tr(S) - 1$, where S denotes the smoothing matrix, and the degrees of freedom of the parametric portion are $p + 3$ ($p + 2$ for survival data). Thus, the degrees of freedom of the full model are $tr(S) + p + 2$ ($tr(S) + p + 1$), and the degrees of freedom for the likelihood ratio test statistic are $tr(S) + 1$. The function `modgam` will return the p-value for the likelihood ratio test automatically. In addition, `modgam` also performs a permutation test of the global spatial effect and pointwise significance (Kelsall and Diggle, 1998; Webster et al., 2006). The function will return the results of the permutation test when `permute=N.permt` is specified in the function call, where `N.permt` denotes the desired number of permutations used to generate the permutation distribution.

For visualizing inference for spatial effects, the `plot` function will plot all point estimates along

with the associated lower and higher band of confidence intervals provided that `se.fit=TRUE` is specified in the original `modgam` call. By setting "contours = intervals", areas with confidence intervals excluding 0 (on the log estimated effect scale) will be indicated on the map by plotting the contours of an indicator vector created to indicate whether 0 is below, between or above the confidence intervals at the grid points. By setting "contours = permrank", contours will be added to indicate significant areas that had a pointwise permutation based p value less than a specified threshold (default of .05).

An example

Returning to the `CAdat` example presented in Section 3.3, we consider visualizing spatial inference. Setting `se.fit=TRUE`, `modgam` function returns pointwise standard errors and confidence intervals. In `fit3` below, the resulting standard errors can be obtained via the call `fit3$se`. The resulting confidence intervals can be plotted via the `plot` function, and are shown in Figure 7.

```
R> fit3 <- modgam(Surv(time, event) ~ AGE + factor(INS) + lo(X, Y),
+   data = CAdat, rgrid = CAgrid, sp = 0.3, verbose = FALSE,
+   se.fit = TRUE)
R> plot(fit3, CAmap, exp = True, mapmin = 0.2, mapmax = 5,
+   border.gray = 0.7, contours = "interval")
R> fit3

Call:
modgam(formula = Surv(time, event) ~ AGE + factor(INS) + lo(X,
Y), data = CAdat, rgrid = CAgrid, sp = 0.3, se.fit = TRUE,
verbose = FALSE)

Model:
Surv(time, event) ~ lo(X, Y) + AGE + factor(INS)
span: 0.3

Coefficients:
AGE factor(INS)Mcr factor(INS)Mng factor(INS)0th factor(INS)Uni
0.02657848    0.03657777    0.05251440    0.16770033    0.26790051
factor(INS)Unk
0.07594159
```

Concluding remarks

GAMs provide a unified statistical framework that allows for the adjustment of individual-level risk factors when evaluating spatial variability in a flexible way. Given the complex nature of spatial patterns, GAMs provide an improved framework over traditional parametric modeling of spatial patterns. The `MapGAM` package introduced here provides a fairly comprehensive and user-friendly set of tools for both fitting GAMs to a variety of outcomes and visualizing complex spatial effects. Of course, one must be careful of overfitting observed data given the flexibility afforded by the GAM framework. As such, care is needed when choosing the degree of flexibility utilized in model specifications and honest assessments of out-of-sample predictive performance should be considered.

Bivariate LOESS smoothing with standard error estimation is computationally intensive, especially in the context of GAMs and proportional hazards models. For example, with 5000 observations, a span size of 0.2, and a binomial outcome `modgam` took about 1 second to provide estimates without standard errors but about 50 seconds with standard error estimates (`se.fit=TRUE`) on recent personal computers. For the same span size and number of observations but with a proportional hazards model, `modgam` took about 40 seconds without standard errors and about 70 seconds with standard errors. Although slower than we might like, the run times for `se.fit=TRUE` are much faster than the pointwise permutation test we previously employed which required a 1000-fold increase in run times (Webster et al., 2006).

Estimating and mapping spatial distributions of disease risk is extremely useful for identifying health disparities, and mapping risk surfaces that are adjusted for individual-level confounding variables is of great interest to epidemiologists. By developing and actively maintaining a convenient R package, `MapGAM`, we intend to facilitate mapping crude and covariate-adjusted spatial effects for

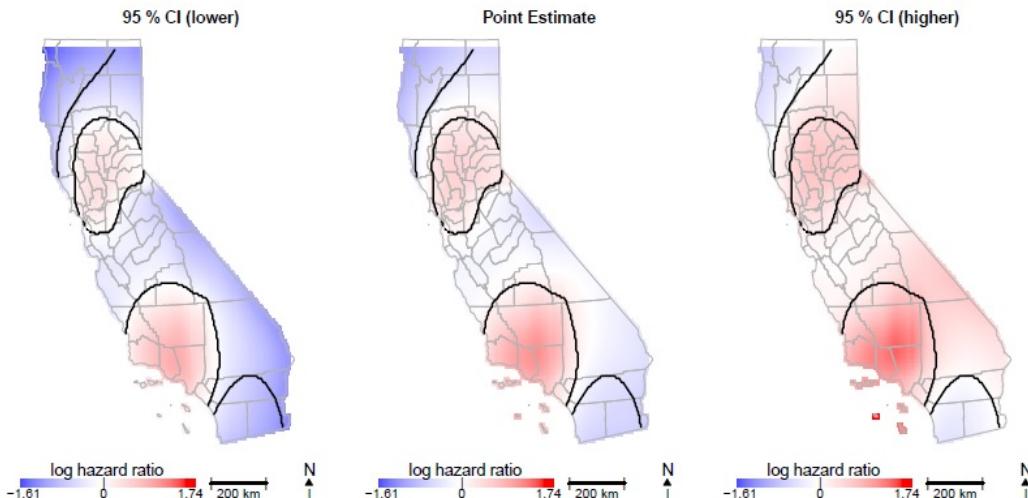


Figure 7: Heatmap of the hazard ratio as well as confidence intervals compared to the median hazard with significant areas circled which were identified by confidence intervals. The left plot illustrates the lower bound of a 95% confidence interval for the hazard ratio at each location. The center plot depicts the estimated hazard ratio at each location. The right plot indicates the upper bound of a 95% confidence interval for the hazard ratio at each location.

the most common probability models used to characterize the relationship of disease risk to spatial location and other factors. In the future we hope to improve the flexibility of the package by expanding the incorporated smoothing methods, including the addition of basis expansion and tensor product methods, allowing for smoothing over more than two dimensions, and expanding the `sampcont` function to include additional sampling methods such as matching. Further research on the development and implementation of adaptive smoothing methods that allow for the amount of smoothing to vary depending on the local extent of a spatial effect is currently in progress, and may be added to the package in a future update. In addition, while spatial correlation is accounted for via the fixed effects smoothed spatial term in the models we have presented, correlation may also arise if repeated measures on sampling units are taken through time. This is currently beyond the scope of the package, but is an area of our current research.

Acknowledgments

Funding for the project was provided by NIH NIEHS Grant No. P42ES007381.

Bibliography

- A. Akullian, P. Kohler, J. Kinuthia, K. Laserson, L. A. Mills, J. Okanda, G. Olilo, M. Ombok, F. Odhiambo, D. Rao, J. Wakefield, and G. John-Stewart. Geographic distribution of hiv stigma among women of childbearing age in rural kenya. *AIDS*, 28:1665–1672, 2014. URL <https://doi.org/10.1097/QAD.0000000000000318>. [p]
- S. Baker, K. E. Holt, A. C. Clements, A. Karkey, A. Arjyal, M. F. Boni, S. Dongol, N. Hammond, S. Koirala, P. T. Duy, T. V. T. Nga, J. I. Campbell, C. Dolecek, B. Basnyat, G. Dougan, and J. J. Farrar. Combined high-resolution genotyping and geospatial analysis reveals modes of endemic urban typhoid fever transmission. *Open Biology*, 1(2):110008, 2011. URL <https://doi.org/10.1098/rsob.110008>. [p]
- C. Belitz, A. Brezger, T. Kneib, S. Lang, and N. Umlauf. **BayesX**: Software for Bayesian Inference in Structured Additive Regression Models, 2016. URL <http://www.BayesX.org/>. Version 1.1. [p]

- N. E. Breslow and D. G. Clayton. Approximate inference in generalized linear mixed models. *Journal of the American Statistical Association*, pages 9–25, 1993. URL <https://doi.org/10.1080/01621459.1993.10594284>. [p]
- R. E. Bristow, J. Chang, A. Ziogas, H. Anton-Culver, and M. Vieira, Veronica. Spatial analysis of adherence to treatment guidelines for advanced-stage ovarian cancer and the impact of race and socioeconomic status. *Gynecologic Oncology*, 134:60–67, 2014. URL <https://doi.org/10.1016/j.ygyno.2014.03.561>. [p]
- J. Chambers and T. J. Hastie. *Statistical Models in S*. Chapman and Hall/CRC, 1992. [p]
- W. S. Cleveland. Robust locally weighted regression and smoothing scatterplots. *Journal of the American Statistical Association*, 74:829–836, 1979. URL <https://doi.org/10.1080/01621459.1979.10481038>. [p]
- W. S. Cleveland. **LOWESS**: A program for smoothing scatterplots by robust locally weighted regression. *The American Statistician*, 35:54, 1981. [p]
- W. S. Cleveland and S. J. Devlin. Locally-weighted regression: An approach to regression analysis by local fitting. *Journal of the American Statistical Association*, 83:596–610, 1988. URL <https://doi.org/10.1080/01621459.1988.10478639>. [p]
- B. Efron. The efficiency of cox's likelihood function for censored data. *Journal of the American Statistical Association*, 72:557–565, 1977. URL <https://doi.org/10.1080/01621459.1977.10480613>. [p]
- P. Elliott and D. Wartenberg. Spatial epidemiology: Current approaches and future challenges. *Environmental Health Perspectives*, 112:998–1106, 2004. URL <https://doi.org/10.1289/ehp.6735>. [p]
- T. Hastie. **gam**: *Generalized Additive Models*, 2004. URL <https://CRAN.R-project.org/package=gam>. R package version 1.12. [p]
- T. Hastie and R. Tibshirani. Generalized additive model. *Statistical Science*, pages 297–318, 1986. URL doi:10.1214/ss/1177013604. [p]
- R. Henderson, S. Shimakura, and D. Gorst. Modeling spatial variation in leukemia survival data. *Journal of the American Statistical Association*, 97:965 – 975, 2002. URL <https://doi.org/10.1198/016214502388618753>. [p]
- K. Hoffman, A. Aschengrau, T. F. Webster, S. M. Bartell, and V. M. Vieira. Associations between residence at birth and mental health disorders: A spatial analysis of retrospective cohort data. *BMC Public Health*, 15(688), 2015. URL <https://doi.org/10.1186/s12889-015-2011-z>. [p]
- T. Hothorn, P. Buehlmann, T. Kneib, M. Schmid, and B. Hofner. **mboost**: *Model-Based Boosting*, 2016. URL <https://cran.r-project.org/web/packages/mboost/>. R package version 2.7-0. [p]
- J. E. Kelsall and P. J. Diggle. Spatial variation in risk of disease: A nonparametric binary regression approach. *Applied Statistics*, pages 559–573, 1998. URL <https://doi.org/10.1111/1467-9876.00128>. [p]
- T. Kneib, F. Heinzel, A. Brezger, D. S. Bove, and N. Klein. **BayesX**: *R Utilities Accompanying the Software Package BayesX*, 2014. URL <https://cran.r-project.org/web/packages/BayesX>. R package Version 0.2-9. [p]
- R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2015. URL <http://www.R-project.org/>. [p]
- H. Reulen. **gamboostMSM**: *Estimating Multistate Models Using gamboost()*, 2014. URL <https://cran.r-project.org/web/packages/gamboostMSM/>. R package version 1.1.87. [p]
- R. A. Rigby and D. M. Stasinopoulos. Generalized additive models for location, scale and shape,(with discussion). *Applied Statistics*, 54:507–554, 2005. URL <https://doi.org/10.1111/j.1467-9876.2005.00510.x>. [p]
- D. M. Stasinopoulos and R. A. Rigby. Generalized additive models for location scale and shape (**GAMLSS**) in R. *Journal of Statistical Software*, 2007. [p]

- M. Stasinopoulos, B. Rigby, and N. Mortan. **gamlss.cens**: *Fitting an Interval Response Variable Using gamlss.family distributions*, 2015. URL <http://CRAN.R-project.org/package=gamlss.cens>. R package version 4.3.1. [p]
- B. M. Taylor and B. S. Rowlingson. **spatsurv**: an R package for bayesian inference with spatial survival models. *Journal of Statistical Software*, 2014. URL <https://doi.org/10.18637/jss.v077.i04>. [p]
- B. M. Taylor, B. S. Rowlingson, and Z. Zheng. **spatsurv**: *Bayesian Spatial Survival Analysis with Parametric Proportional Hazards Models*, 2016. URL <https://cran.r-project.org/web/packages/spatsurv/>. R package version 0.9-14. [p]
- N. Umlauf, D. Adler, T. Kneib, S. Lang, and A. Zeileis. Structured additive regression models: An R interface to BayesX. *Journal of Statistical Software*, 63(21):1–46, 2015. URL <https://doi.org/10.18637/jss.v063.i21>. [p]
- N. Umlauf, T. Kneib, S. Lang, and A. Zeileis. **R2BayesX**: *Estimate Structured Additive Regression Models with BayesX*, 2016. URL <https://cran.r-project.org/web/packages/R2BayesX/>. R package version 1.1-0. [p]
- V. M. Vieira, T. F. Webster, J. M. Weinberg, and A. Aschengrau. Spatial-temporal analysis of breast cancer in upper cape cod, massachusetts. *International Journal of the Health Geographics*, 7(46), 2008. URL <https://doi.org/10.1186/1476-072X-7-46>. [p]
- Y. T. W. **VGAM**: *Vector Generalized Linear and Additive Models*, 2007. URL <https://cran.r-project.org/web/packages/VGAM>. R package version 1.0-2. [p]
- T. Webster, V. Vieira, J. Weinberg, and A. Aschengrau. Method for mapping population-based case-control studies: An application using generalized additive models. *International Journal of Health Geographics*, 5(26), 2006. URL <https://doi.org/10.1186/1476-072X-5-26>. [p]
- S. Wood. *Mixed Gam Computation Vehicle with GCV/AIC/REML Smoothness Estimation*, 2009. URL <https://CRAN.R-project.org/package=mgcv>. R package version 1.8-10. [p]

Appendix

We have conducted simulations to assess the performance of the proposed method for fitting the Cox proportional hazards additive mode in Section 3.2. Data was generated by the function `sim.sample.data` under the settings described in Section 3.2.

```
R> sim.sample.data <- function(f, N = 5000){
+   set.seed(269)
+   u <- runif(N, -1, 1)
+   v <- runif(N, -1, 1)
+   x <- runif(N, -1, 1)
+   lambda <- 0.03 * exp(f(u, v, x))
+   eventTime <- rexp(N, lambda)
+   censTime <- runif(N, 0, 70)
+   time <- ifelse(eventTime <= censTime, eventTime, censTime)
+   event <- (eventTime <= censTime) * 1
+   obs.data <- data.frame(time = time, event = event, u = u, v = v, x = x)
+   new.data <- data.frame(u = rep(seq(-1, 1, 0.01), each = 201),
+                          v = rep(seq(-1, 1, 0.01), 201))
+   truth <- f(new.data$u, new.data$v, 0)
+   list(obs = obs.data, new = new.data, truth = truth - median(truth))
+ }
```

The first simulation example assumes a linear effect of all covariates on the log-hazard as shown in Eq. 21. The following code generates the data for the first simulation example and estimates the spatial effect using `modgam()` function.

```
R> f.linear <- function(u, v, x){
+   log(0.7) * x + log(1.2) * u + log(1.5) * v
+ }
R> data.linear <- sim.sample.data(f.linear)
R> fit.linear <- modgam(Surv(time, event) ~ lo(u, v) + x,
```

```
+ data = data.linear$obs, rgrid = data.linear$new,
+ family = "survival", sp = 0.4)
```

The second simulation example assumes a nonlinear effect of spatial parameters on the log-hazard as shown in Eq. 22. The following code generates the data for the second simulation example and estimates the spatial effect using `modgam()` function.

```
R> f.nonlinear <- function(u,v,x){
+   log(0.7)*x + log(1.2)*u + log(1.5)*v+log(0.8)*u^2+log(1.8)*u*v
+ }
R> data.nonlinear <- sim.sample.data(f.nonlinear)
R> fit.nonlinear <- modgam(Surv(time, event) ~ lo(u, v) + x,
+   rgrid = data = data.nonlinear$obs, data.nonlinear$new,
+   family = "survival", sp = 0.2)
```

Heatmaps of the log-hazard ratio comparing the hazard of the location to the median hazard for the two simulation examples are generated using the following code:

```
R> par(mfrow = c(2, 2))
R> obj.linear <- list(grid = data.linear$new, fit = data.linear$truth)
R> colormap(obj.linear, axes = T, arrow = F, mapmin = -0.6, mapmax = 0.55,
+   legend.name = "log hazard ratio", legend.cex = 1.3, legend.add.line = 0,
+   col.seq = diverge_hsv(201))
R> mtext("(a) Truths for linear spatial effect", side = 1, line = 4)
R> plot(fit.linear, mapmin = -0.6, mapmax = 0.55, axes = T, arrow = F,
+   legend.cex = 1.3)
R> mtext("(b) Estimates for linear spatial effect", side = 1, line = 4)
R> obj.nonlinear <- list(grid = data.nonlinear$new,
+   fit = data.nonlinear$truth)
R> colormap(obj.nonlinear, axes = T, arrow = F, mapmin = -0.9,
+   mapmax = 1.05, legend.name = "log hazard ratio", legend.cex = 1.3,
+   legend.add.line = 0, col.seq = diverge_hsv(201))
R> mtext("(c) Truths for nonlinear spatial effect", side = 1, line = 4)
R> plot(fit.nonlinear, mapmin = -0.9, mapmax = 1.05, axes = T,
+   arrow = F, legend.cex = 1.3)
R> mtext("(d) Estimates for nonlinear spatial effect", side = 1, line = 4)
```

Comparisons of the true log-hazard ratio and the estimated log-hazard ratio for the two simulation examples are plotted using the following code:

```
R> par(mfrow = c(1, 2), mai = c(1.3, 0.8, 0.4, 0.4))
R> plot(data.linear$truth, fit.linear$fit, xlab = "true log hazard ratio",
+   ylab = "predicted log hazard ratio")
R> abline(0, 1, lwd = 4, col = "green")
R> mtext("(a) Linear spatial effect", side = 1, line = 4.5)
R> plot(data.nonlinear$truth, fit.nonlinear$fit,
+   xlab = "true log hazard ratio", ylab = "predicted log hazard ratio")
R> abline(0, 1, lwd = 4, col = "green")
R> mtext("(b) Nonlinear spatial effect", side = 1, line = 4.5)
```

Lu Bai

Department of Statistics

University of California, Irvine

Irvine, California 92697-1250, United States of America

E-mail: bail1@uci.edu

URL: <http://publichealth.uci.edu/spatialepidemiology/>

Daniel L. Gillen

Department of Statistics

University of California, Irvine

Irvine, California 92697-1250, United States of America

E-mail: dgillen@uci.edu

URL: <http://www.ics.uci.edu/~dgillen>

Scott M. Bartell

Program in Public Health

*Department of Statistics
University of California, Irvine
Irvine, California 92697, United States of America
E-mail: sbartell@uci.edu
URL: <http://publichealth.uci.edu/spatialepidemiology/>*

*Verónica M. Vieira
Program in Public Health
University of California, Irvine
Irvine, California 92697, United States of America
E-mail: bail1@uci.edu
URL: <http://publichealth.uci.edu/spatialepidemiology/>*

tsmp: An R Package for Time Series with Matrix Profile

by Francisco Bischoff and Pedro Pereira Rodrigues

Abstract This article describes **tsmp**, an R package that implements the MP concept for TS. The **tsmp** package is a toolkit that allows all-pairs similarity joins, motif, discords and chains discovery, semantic segmentation, etc. Here we describe how the **tsmp** package may be used by showing some of the use-cases from the original articles and evaluate the algorithm speed in the R environment. This package can be downloaded at <https://CRAN.R-project.org/package=tsmp>.

Introduction: time series data mining

A TS is a sequence of real-valued numbers indexed in time order. Usually, this sequence is taken in a regular period of time, which will be assumed to be true in this context. The interests in TS data mining have been growing along with the increase in available computational power. This kind of data is easily obtained from sensors (e.g., ECG), (ir)regular registered data (e.g., weekly sales, stock prices, brachial blood pressure). Even other kinds of data can be converted to TS format, such as shapes (Wei et al., 2006) and DNA sequences (Shieh and Keogh, 2008). TS are generally large, high dimensional and continuously updated which requires algorithms fast enough in order to be meaningful. Besides, unlike other kinds of data, which usually have exact answers, TS are usually analysed in an approximated fashion.

These characteristics have been challenging researchers to find faster and more accurate methods to retrieve meaningful information from TS. This required one or more of these methods: dimensionality reduction, constraints, domain knowledge, parameter tweaks. Only afterwards could the data mining tasks be applied in feasible time. Typical tasks include motif and discord discovery, subsequence matching, semantic segmentation, rule discovery, similarity search, anomaly detection, clustering, classification, indexing, etc. (Fu, 2011).

This paper describes the **tsmp** package (Bischoff, 2018) which uses a novel approach to TS data mining: the MP Yeh et al. (2017b), which is based on the APSS (also known as similarity join). The APSS' task is to, given a collection of data objects, retrieve the nearest neighbour for each object. The remaining part of this paper is organised as follows: In Section 2 we describe the reasoning behind the MP, in Section 3 we present the **tsmp** package with examples, in Section 4 we compare the performance of the R implementation, and in Section 5 we conclude with a brief discussion.

The matrix profile

The reader may be aware of what a DM is. It is widely used in TS for clustering, classification, motif search, etc. But, even for modestly sized datasets, the algorithms can take months to compute even with speed-up techniques such as indexing (Shieh and Keogh, 2008; Fu et al., 2008), lower-bounding (Keogh and Ratanamahatana, 2005), data discretization (Lin et al., 2003) and early abandoning (Faloutsos et al., 1994). At best, they can be one or two orders of magnitude faster.

The MP is an ordered vector that stores the Euclidean distance between each pair within a similarity join set. One (inefficient) way would be to use the full DM of every iteration of a sliding window join and retrieve just the smallest (non-diagonal) value of each row. The MP also has a companion vector called PI, that gives us the position of the nearest neighbour of each subsequence.

This method has a host of interesting and exploitable properties. For example, the highest point on the MP corresponds to the TS discord, the (tied) lowest points correspond to the locations of the best TS motif pair, and the variance can be seen as a measure of the TS complexity. Moreover, the histogram of the values in the MP is the exact answer to the TS density estimation. Particularly, it has implications for TS motif discovery, TS joins, shapelet discovery (classification), density estimation, semantic segmentation, visualisation, rule discovery, clustering, etc. (Yeh et al., 2017b).

Some of the advantages/features of this method:

- It is *exact*, providing no false positives or false dismissals.
- It is *simple* and parameter-free. In contrast, the more general metric space APSS algorithms require building and tuning spatial access methods and/or hash functions.
- It requires an inconsequential space overhead, just $O(n)$ with a small constant factor.

- It is extremely *scalable*, and for *massive* datasets, we can compute the results in an anytime fashion, allowing ultra-fast *approximate* solutions.
- Having computed the similarity join for a dataset, we can incrementally update it very efficiently. In many domains, this means we can effectively maintain exact joins on *streaming* data forever.
- It provides *full joins*, eliminating the need to specify a similarity threshold, which is a near-impossible task in this domain.
- It is *parallelizable*, both on multicore processors and in distributed systems (Zhu et al., 2016).

The `tsmp` package

The `tsmp` package provides several functions that allow for an easy workflow using the MP concept for TS mining. The package is available from the CRAN at <https://CRAN.R-project.org/package=tsmp>. In Section 3.1 we explain how to install this package. In Section 3.2 we describe the syntax for the main functions in `tsmp`, giving an example of a particular model. In Section 3.3 we will further explain the available algorithms for MP computation and its current use. In Section 3.4 we show some examples of MP application for data mining.

Installation

The `tsmp` package can be installed in two ways:

The release version from CRAN:

```
install.packages("tsmp")
```

or the development version from GitHub:

```
# install.packages("devtools")
devtools::install_github("franzbischoff/tsmp")
```

Input arguments and example

The `tsmp` has a simple and intuitive workflow. First, you must compute the MP of the desired TS. Depending on the task, the user might want to follow one of three paths: univariate self-join, AB-join or multivariate self-join. One exception is the SiMPle algorithm that is a multivariate AB-join and will be explained in Section 3.3.

The main function is `tsmp()`, which has the following usage:

```
tsmp(..., window_size, exclusion_zone = 1/2,
      mode = c("stomp", "stamp", "simple", "mstomp", "scrimp"),
      verbose = 2, s_size = Inf, must_dim = NULL, exc_dim = NULL,
      n_workers = 1, .keep_data = TRUE)
```

The first argument `ellipsis` (the three dots) receives one or two TS. For self-joins, the user must input just one TS; two for AB-joins. Multivariate TS may be input as a matrix where each column represents one dimension. Alternatively, the user may input the Multivariate TS as a list of vectors. The second argument `window_size` is the size of the sliding window. These are the most basic parameters you need to set.

Further parameters are:

- `exclusion_zone`, is an important parameter for self-joins. This is used to avoid trivial matches and is a modifier of the `window_size`, i.e., for an `exclusion_zone` of 1/2, and `window_size` of 50, internally the result will be 25.
- `mode`, here the user may choose the algorithm used for the MP calculation. `stomp`, `stamp` and `scrimp` return equal results, although differing in some practical attributes, and they will be further explained in Section 3.3. `mstomp` is designed for Multivariate TS self-join only. `simple` is designed for Multivariate TS for self-join and AB-join, which will also be further explained in Section 3.3.
- `verbose`, controls the verbosity of the function. 0 means no feedback, 1 means text messages only, 2 (the default) means text messages and progress bar, and 3 also plays a sound when finished.

- `s_size`, controls the *anytime* algorithms. This is just a way to end the algorithm in a controlled manner because the *anytime* algorithms can be stopped *anytime* and the result will be returned.
- `must_dim`, is an optional argument for the `mstomp` algorithm. See next item.
- `exc_dim`, as `must_dim`, is an optional argument for the `mstomp` algorithm. These arguments control which dimensions must be included and which must be excluded from the multidimensional MP.
- `n_workers`, controls how many threads will be used for the `stamp`, `stomp`, and `mstomp`. Note that for small datasets, multiple threads add an overhead that makes it slower than just one thread.
- `.keep_data`, `TRUE` by default, keeps the input data inside the output object. This is useful for chained commands.

Example data

We think that the best and simple example to demonstrate the `tsmp` package is the motif search.

The `tsmp` package imports the `%>%` (pipe) operator from the `magrittr` package that makes the `tsmp` workflow easier.

The following code snippet shows an example of the workflow for motif search:

```
R> data <- mp_fluss_data$walkjogrun$data
R> motifs <- tsmp(data, window_size = 80, exclusion_zone = 1/2) %>%
+   find_motif(n_motifs = 3, radius = 10, exclusion_zone = 20) %T>% plot()
```

The `find_motif()` function is an S3 class that can receive as the first argument the output of `tsmp()` function as a univariate or multivariate MP. This allows us to use the pipe operator easily. The `plot()` function is also an S3 class extension for plotting objects from the `tsmp` package and works seamlessly.

Computational methods

There are several methods to compute the MP. The reason for that is the unquenchable need for speed of the UCR's researchers. Before starting, let's clarify that the time complexity of a brute force algorithm has a time complexity of $O(n^2m)$, for n being the length of the reference TS and m the length of the sliding window (query) that is domain dependent.

STAMP

This was the first algorithm used to compute the MP. It uses the MASS ([Mueen et al., 2015](#)) as the core algorithm for calculating the similarity between the query and the reference TS, called the DP. The ultimate MP comes from merging the element-wise minimum from all possible DP. This algorithm has the time complexity of $O(n^2 \log n)$ and space complexity of $O(n)$ ([Yeh et al., 2017b](#)). The *anytime* property is achieved using a random approach where the best-so-far MP is computed using the DP that have been already calculated.

STOMP

This was the second algorithm used to compute the MP. It also uses the MASS to calculate the DP but only for the first iteration of each batch. The researchers noticed that they could reuse the values calculated of the first DP to make a faster calculation in the next iterations. This results on a time complexity of $O(n^2)$, keeping the same space complexity of $O(n)$. This algorithm is also suitable for a GPU framework (although this was not yet implemented in `tsmp` package) ([Zhu et al., 2016](#)). The main drawback of STOMP compared with STAMP is the lack of the *anytime* property. In scenarios where a fast convergence is needed (e.g., finding the top- k motifs) it may be required only 5% of the MP computation to provide a very accurate approximation of the final result.

SCRIMP

The SCRIMP algorithm is still experimental at the time of this article. It combines the best features of STOMP and STAMP, having a time complexity of $O(n^2)$ and the *anytime* property ([UCR, 2016](#)).

SiMPle

The SiMPle algorithm is a variation designed for music analysis and exploration (Silva et al., 2018). Internally it uses STOMP for MP computation and allows multidimensional self-joins and AB-joins. The resulting MP is computed using all dimensions. One major difference is that it doesn't apply any z-normalization on the data, since for music domain this would result in spurious similarities.

mSTOMP

The mSTOMP algorithm was designed to motif search in multidimensional data (Yeh et al., 2017a). Performing motif search on *all* dimensions is almost guaranteed to produce meaningless results, so this algorithm, differently from SiMPle, doesn't compute the MP using all dimensions naively, but the d -dimensional MP for every possible setting of d , simultaneously, in $O(dn^2 \log d)$ time and $O(dn)$ space. The resulting MP allow motif search in multiple dimensions and also to identify which dimensions are relevant for the motifs founded.

Data mining tasks

Motif search

In Section 3.2 we have shown a basic example of the workflow for motif search. Let's take a look at the result of that code:

```
R> motifs

Matrix Profile
-----
Profile size = 9922
Window size = 80
Exclusion zone = 40
Contains 1 set of data with 10001 observations and 1 dimension

Motif
-----
Motif pairs found = 2
Motif pairs indexes = [584, 741] [4799, 5329]
Motif pairs neighbors = [2948, 9900, 8265] [7023, 8861, 2085, 248]
```

As we can see, this is a summary that `tsmp` package automatically generates from the resulting object. One nice property is that the object always holds the original MP and by default also holds the input data so that you can keep mining information from it. If the dataset is too big or you are concerned about privacy, you may set the argument `.keep_data = FALSE`.

In addition to this summary, you can see the results using `plot()` in Figure 1:

```
R> plot(motifs, type = "matrix")
```

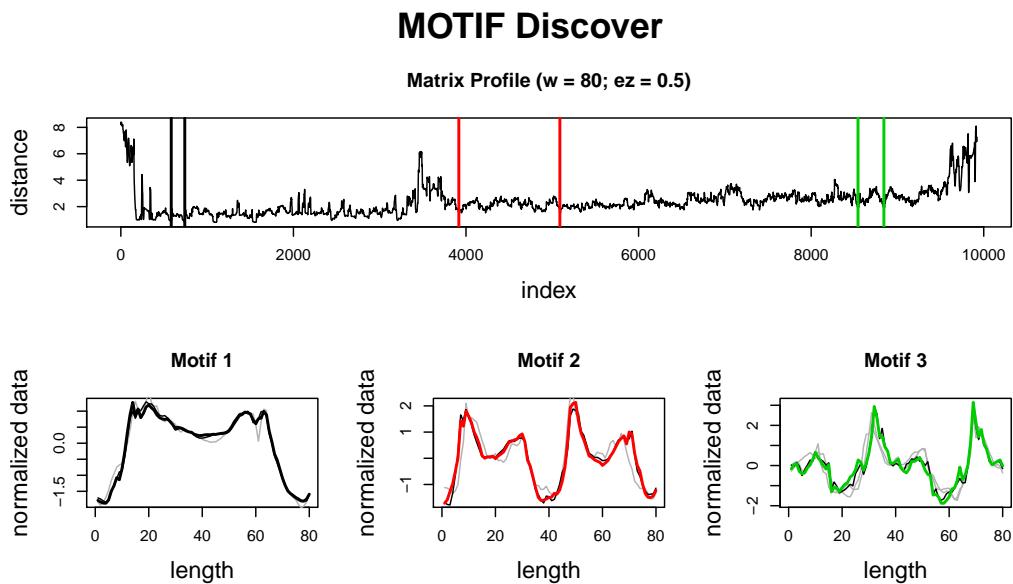


Figure 1: The upper graphic shows the computed MP with each motif pair as a coloured vertical bar. The lower graphics show each motif in colour and the founded neighbours in grey.

This dataset is the *WalkJogRun* PAMAP's dataset (Reiss and Stricker, 2012). It contains the recording of human movements in three states, walking, jogging and running. As we can see, the plot shows the motifs of each state. Experienced readers might say that this is not the purpose of motif search, and we agree. The result shown here was achieved using a large `radius` and `exclusion_zone` to force the algorithm to look for distant motifs. Semantic segmentation is the proper algorithm for this task, and we will show this in the next section.

Semantic segmentation

As previously explained, the resulting object holds the original data and MP. So let's save some time and use the resulting object from the last section to try to find where the human subject started to jog and to run:

```
R> segments <- motifs %>% fluss(num_segments = 2)
R> segments

Matrix Profile
-----
Profile size = 9922
Window size = 80
Exclusion zone = 40
Contains 1 set of data with 10001 observations and 1 dimension

Arc Count
-----
Profile size = 9922
Minimum normalized count = 0.063 at index 3448

Fluss
-----
Segments = 2
Segmentation indexes = 3448 6687
```

We can see that this object now holds information of the FLUSS algorithm (Gharghabi et al., 2017), but the motif information is still there and can be retrieved using `as.motif()`. In Figure 2

we can see the graphic result of the segmentation.

```
R> plot(segments, type = "data")
```

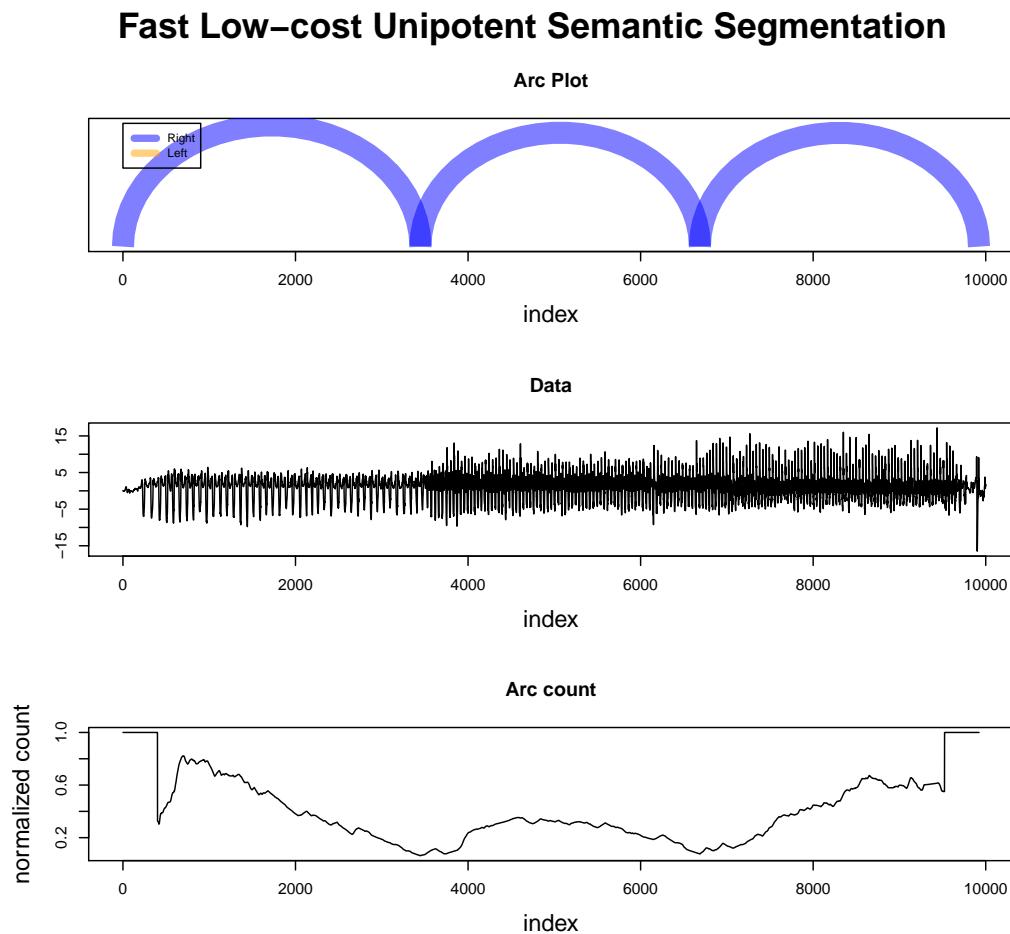


Figure 2: Semantic segmentation using MP. The upper graphic shows the arc plot of predicted semantic changes (ground truth is 3800 and 6800). The middle graphic shows the data. The lower graphic shows the normalised arc counts with correction for the “edge-effect” (Gharghabi et al., 2017).

Time series chains

As a final example of practical application, let's search for a new kind of primitive: time series chains ([Zhu et al., 2018a](#)). This algorithm looks for patterns that are not just similar but evolve through time. The dataset used in this example is a record of the Y-axis of a mobile phone accelerometer while placing it on a walking subject's pocket ([Hoang et al., 2015](#)). The authors of this dataset wanted to analyse the stability of the mobile phone as it slowly settles in the pocket. This is a good example of a pattern that changes through time. Let's start with the workflow for this example:

```
R> chains <- mp_gait_data %>% tsmq(window_size = 50, exclusion_zone = 1/4,
+   verbose = 0) %>% find_chains()
R> chains

Matrix Profile
-----
Profile size = 855
Window size = 50
Exclusion zone = 13
Contains 1 set of data with 904 observations and 1 dimension

Chain
-----
Chains found = 58
Best Chain size = 6
Best Chain indexes = 148 380 614 746 778 811
```

Here we see that the algorithm found 58 chains. *Id est*, it found 58 evolving patterns with at least three elements, and the best one is presented in the last line, a chain with six elements. Figure 3 shows the patterns discovered.

Speed

While this new method for TS data mining is extremely fast, we have to take into consideration that the R environment is not as fast as a low-level implementation such as C/C++. In Table 1 we present the comparison to the MATLAB version that is available at the UCR. [Yeh et al. \(2017b\)](#) shows that the slowest algorithm (STAMP) can be hundreds of times faster than the MK algorithm (the fastest known *exact* algorithm for computing TS motifs) ([Yoon et al., 2015](#)), while the R implementation is just 1.65 to 8.04 times slower than MATLAB's, which is not a problem for an R researcher.

```
R> set.seed(2018)
R> data <- cumsum(sample(c(-1, 1), 40000, TRUE))
```

Algorithm	R Time*	MATLAB Time*	Threads
scrimp	45.30	27.49	1
stomp	52.72	10.27	8
stomp	136.01	16.91	1
stamp	140.25	55.57	8
stamp	262.03	113.18	1

Table 1: Performances of R and MATLAB implementations on an Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz using a random walk dataset. *Median of 5 trials, in seconds.

Conclusion

The examples in Section 3.4 show how straightforward the usage of `tsmp` package is. Regardless, these examples are just a glimpse of the potential of the MP. Several new algorithms based on MP are being developed and will be gradually implemented in the `tsmp` package ([Linardi et al., 2018](#); [Zhu et al., 2018b](#); [Gharghabi et al., 2018](#); [Imani et al., 2018](#)). [Yeh et al. \(2017a\)](#) for example, have developed an algorithm to allow MDS visualisation of motifs. [Gharghabi et al. \(2018\)](#) have developed a new distance measure that better suits repetitive patterns ([Imani et al., 2018](#)).

```
R> plot(chains, ylab = "")
```

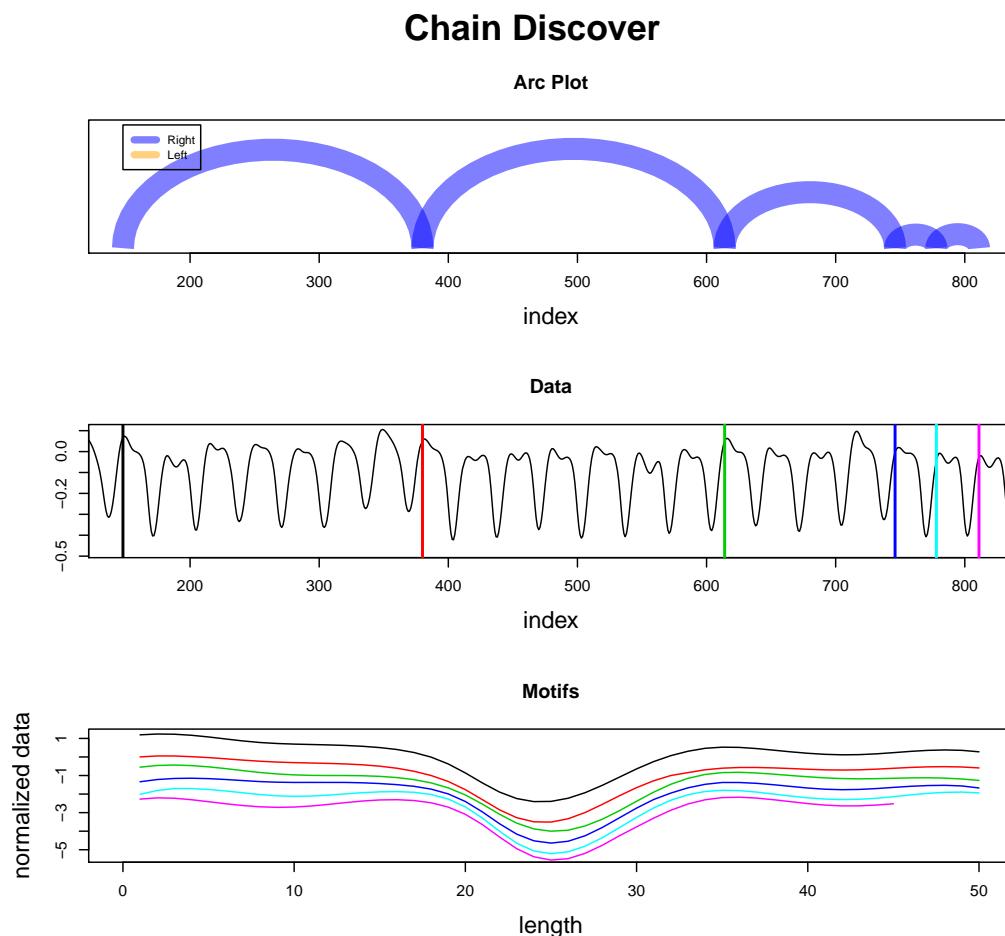


Figure 3: Finding evolving patterns using MP. The upper graphic shows the arc plot of the discovered patterns. The middle graphic shows the data and the position of every pattern as a vertical coloured line. The lower graphic shows the patterns for comparison. They are y-shifted for visualisation only.

The MP has the potential to revolutionise the TS data mining due to its generality, versatility, simplicity and scalability (UCR, 2016). All existing algorithms for MP have been proven to be flexible to be used in several domains using very few parameters and they are also robust, showing good performance with dimensionality reduced data and noisy data. In addition, a yet to be published article shows a fantastic score of $> 10^{18}$ pairwise comparisons a day using GPU for motif discovery (Zimmerman et al., 2018).

The **tsmp** package is the first known MP toolkit available on any statistical language, and we hope it can help researchers to better mining TS and also to develop new methods based on MP.

Acknowledgements

We would like to thank the researchers from UCR for their contribution and permission to use their base code to be implemented in this package. Particularly to Prof. Eamonn Keogh whose work and assistance led to this project. We also acknowledge the participation in project NanoSTIMA (NORTE-01-0145-FEDER-000016) which was financed by the North Portugal Regional Operational Program (NORTE 2020) under the PORTUGAL 2020 Partnership Agreement and through the European Regional Development Fund (ERDF).

Acronyms

- APSS: all-pairs similarity search
- CRAN: Comprehensive R Archive Network
- DM: distance matrix
- DP: distance profile
- ECG: electrocardiogram
- FLUSS: fast low-cost unipotent semantic segmentation
- GPU: graphics processor unit
- MASS: Mueen's algorithm for similarity search
- MDS: multidimensional space
- MP: matrix profile
- mSTOMP: Multivariate scalable time series ordered-search matrix profile
- PI: profile index
- SCRIMP: Scalable column independent matrix profile
- SiMPle: Similarity matrix profile
- STAMP: Scalable time series anytime matrix profile
- STOMP: Scalable time series ordered-search matrix profile
- TS: time series
- UCR: University of California Riverside

Bibliography

- F. Bischoff. *tsmp: Time Series with Matrix Profile*, 2018. URL <https://CRAN.R-project.org/package=tsmp>. R package version 0.3.2. [p]
- C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. *ACM SIGMOD Record*, 23(2):419–429, jun 1994. ISSN 01635808. doi: <https://doi.org/10.1145/191843.191925>. [p]
- A. W. C. Fu, E. Keogh, L. Y. H. Lau, C. A. Ratanamahatana, and R. C. W. Wong. Scaling and time warping in time series querying. *VLDB Journal*, 17(4):899–921, 2008. ISSN 10668888. doi: <https://doi.org/10.1007/s00778-006-0040-z>. [p]
- T. C. Fu. A review on time series data mining. *Engineering Applications of Artificial Intelligence*, 24(1):164–181, 2011. ISSN 09521976. doi: <https://doi.org/10.1016/j.engappai.2010.09.007>. [p]
- S. Gharghabi, Y. Ding, C.-C. M. Yeh, K. Kamgar, L. Ulanova, and E. Keogh. Matrix Profile VIII: Domain Agnostic Online Semantic Segmentation at Superhuman Performance Levels. In *2017 IEEE International Conference on Data Mining (ICDM)*, volume 2017-Novem, pages 117–126. IEEE, nov 2017. ISBN 978-1-5386-3835-4. doi: <https://doi.org/10.1109/ICDM.2017.21>. [p]
- S. Gharghabi, S. Imani, A. Bagnall, A. Darvishzadeh, and E. Keogh. Matrix Profile XII: MPdist: A Novel Time Series Distance Measure to Allow Data Mining in More Challenging Scenarios. In *2018 IEEE International Conference on Data Mining (ICDM)*, 2018. [p]
- T. Hoang, D. Choi, and T. Nguyen. On the Instability of Sensor Orientation in Gait Verification on Mobile Phone. In *Proceedings of the 12th International Conference on Security and Cryptography*, pages 148–159. SCITEPRESS - Science and Technology Publications, 2015. ISBN 978-989-758-117-5. doi: <https://doi.org/10.5220/0005572001480159>. [p]
- S. Imani, F. Madrid, W. Ding, S. Crouter, and E. Keogh. Matrix Profile XIII : Time Series Snippets : A New Primitive for Time Series Data Mining. In *2018 IEEE International Conference on Data Mining (ICDM)*, 2018. [p]
- E. Keogh and C. A. Ratanamahatana. Exact indexing of dynamic time warping. *Knowledge and Information Systems*, 7(3):358–386, mar 2005. ISSN 0219-1377. doi: <https://doi.org/10.1007/s10115-004-0154-9>. [p]
- J. Lin, E. Keogh, S. Lonardi, and B. Chiu. A symbolic representation of time series, with implications for streaming algorithms. In *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery - DMKD '03*, page 2, New York, New York, USA, 2003. ACM Press. ISBN 978-3-642-41397-1. doi: <https://doi.org/10.1145/882085.882086>. [p]
- M. Linardi, Y. Zhu, T. Palpanas, and E. Keogh. Matrix Profile X: VALMOD - Scalable Discovery of Variable-Length Motifs in Data Series. In *Proceedings of the 2018 International Conference on Management of Data - SIGMOD '18*, pages 1053–1066, New York, New York, USA, 2018. ACM Press. ISBN 9781450347037. doi: <https://doi.org/10.1145/3183713.3183744>. [p]
- A. Mueen, Y. Zhu, M. Yeh, K. Kamgar, K. Viswanathan, C. K. Gupta, and E. Keogh. The Fastest Similarity Search Algorithm for Time Series Subsequences under Euclidean Distance and Correlation Coefficient, 2015. URL <https://www.cs.unm.edu/~mueen/FastestSimilaritySearch.html>. [p]
- A. Reiss and D. Stricker. Introducing a New Benchmarked Dataset for Activity Monitoring. In *2012 16th International Symposium on Wearable Computers*, pages 108–109. IEEE, jun 2012. ISBN 978-0-7695-4697-1. doi: <https://doi.org/10.1109/ISWC.2012.13>. [p]
- J. Shieh and E. Keogh. iSAX: indexing and mining terabyte sized time series. In *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD 08*, page 623, New York, New York, USA, 2008. ACM Press. ISBN 9781605581934. doi: <https://doi.org/10.1145/1401890.1401966>. [p]
- D. F. Silva, C.-C. M. Yeh, Y. Zhu, G. Batista, and E. Keogh. Fast Similarity Matrix Profile for Music Analysis and Exploration. *IEEE Transactions on Multimedia*, 14(8):1–1, 2018. ISSN 1520-9210. doi: <https://doi.org/10.1109/TMM.2018.2849563>. [p]
- UCR. UCR Matrix Profile Page, 2016. URL <http://www.cs.ucr.edu/~eamonn/MatrixProfile.html>. [p]

- L. Wei, E. Keogh, and X. Xi. SAXually Explicit Images: Finding Unusual Shapes. In *Sixth International Conference on Data Mining (ICDM'06)*, pages 711–720. IEEE, dec 2006. ISBN 0-7695-2701-7. doi: <https://doi.org/10.1109/ICDM.2006.138>. [p]
- C.-c. M. Yeh, N. Kavantzas, and E. Keogh. Matrix Profile VI : Meaningful Multidimensional Motif Discovery. In *Proceedings - IEEE International Conference on Data Mining, ICDM*, 2017a. [p]
- C. C. M. Yeh, Y. Zhu, L. Ulanova, N. Begum, Y. Ding, H. A. Dau, D. F. Silva, A. Mueen, and E. Keogh. Matrix profile I: All pairs similarity joins for time series: A unifying view that includes motifs, discords and shapelets. *Proceedings - IEEE International Conference on Data Mining, ICDM*, pages 1317–1322, 2017b. ISSN 15504786. doi: <https://doi.org/10.1109/ICDM.2016.89>. [p]
- C. E. Yoon, O. O'Reilly, K. J. Bergen, and G. C. Beroza. Earthquake detection through computationally efficient similarity search. *Science Advances*, 1(11):e1501057–e1501057, dec 2015. ISSN 2375-2548. doi: <https://doi.org/10.1126/sciadv.1501057>. [p]
- Y. Zhu, Z. Zimmerman, N. S. Senobari, C.-c. M. Yeh, and G. Funning. Matrix Profile II : Exploiting a Novel Algorithm and GPUs to Break the One Hundred Million Barrier for Time Series Motifs and Joins. *Icdm*, 54(1):739–748, jan 2016. ISSN 0219-1377. doi: <https://doi.org/10.1109/ICDM.2016.126>. [p]
- Y. Zhu, M. Imamura, D. Nikovski, and E. Keogh. Matrix Profile VII: Time Series Chains: A New Primitive for Time Series Data Mining. *Knowledge and Information Systems*, pages 1–27, jun 2018a. ISSN 0219-1377. doi: <https://doi.org/10.1007/s10115-018-1224-8>. [p]
- Y. Zhu, C.-c. M. Yeh, Z. Zimmerman, K. Kamgar, and E. Keogh. Matrix Profile XI: SCRIMP++: Time Series Motif Discovery at Interactive Speeds. In *2018 IEEE International Conference on Data Mining (ICDM)*, 2018b. [p]
- Z. Zimmerman, K. Kamgar, Y. Zhu, N. S. Senobari, B. Crites, and G. Funning. Scaling Time Series Motif Discovery with GPUs: Breaking the Quintillion Pairwise Comparisons a Day Barrier. *ACM*, 2018. doi: <https://doi.org/10.1145/3357223.3362721>. [p]

Francisco Bischoff

CINTEYSIS - Center for Health Technology and Services Research

MEDCIDS - Community Medicine, Information and Health Decision Sciences Department

Faculty of Medicine of the University of Porto

Rua Dr. Placido Costa, s/n

4200-450 Porto, Portugal

ORCID: 0000-0002-5301-8672

fbischoff@med.up.pt

Pedro Pereira Rodrigues

CINTEYSIS - Center for Health Technology and Services Research

MEDCIDS - Community Medicine, Information and Health Decision Sciences Department

Faculty of Medicine of the University of Porto

Rua Dr. Placido Costa, s/n

4200-450 Porto, Portugal

ORCID: 0000-0001-7867-6682

pprodrigues@med.up.pt

ari: The Automated R Instructor

by Sean Kross, Jeffrey T. Leek, John Muschelli

Abstract We present the **ari** package for automatically generating technology-focused educational videos. The goal of the package is to create reproducible videos, with the ability to change and update video content seamlessly. We present several examples of generating videos including using R Markdown slide decks, PowerPoint slides, or simple images as source material. We also discuss how **ari** can help instructors reach new audiences through programmatically translating materials into other languages.

Introduction

Videos are a crucial way people learn and they are pervasive in online education platforms (Hsin and Cigas, 2013; Hartsell and Yuen, 2006). Producing educational videos with a lecturer speaking over slides takes time, energy, and usually video editing skills. Maintaining the accuracy and relevance of lecture videos focused on technical subjects like computer programming or data science can often require remaking an entire video, requiring extensive editing and splicing of new segments. We present **ari**, the **Automated R Instructor** as a tool to address these issues by creating reproducible presentations and videos that can be automatically generated from plain text files or similar artifacts. By using **ari**, we provide a tool for users to rapidly create and update video content.

In its simplest form a lecture video is comprised of visual content (e.g. slides and figures) and a spoken explanation of the visual content. Instead of a human lecturer, the **ari** package uses a text-to-speech system to synthesize spoken audio for a lecture. Modern text-to-speech systems that take advantage of recent advancements in artificial intelligence research are available from Google, Microsoft, and Amazon. Many of these synthesizers make use of deep learning methods, such as WaveNet (Van Den Oord et al., 2016) and have interfaces in R (Edmondson, 2019; Muschelli, 2019a; Leeper, 2017). Currently in **ari**, synthesis of the the audio can be rendered using any of these services through the **text2speech** package (Muschelli, 2019b). The default is Amazon Polly, which has text-to-speech voice generation in over twenty one languages, implemented in the **aws.polly** package (Leeper, 2017). In addition to multiple languages, the speech generation services provide voices with several pitches representing different genders within the same language. We present the **ari** package with reproducible use case examples and the video outputs with different voices in multiple languages.

The **ari** package relies on the **tuneR** package for splitting and combining audio files appropriately so that lecture narration is synced with each slide (Ligges et al., 2018). Once the audio is generated, it is synced with images to make a lecture video. Multiple open source tools for video editing and splicing exist; **ari** takes advantage of the FFmpeg (<http://www.ffmpeg.org/>) software, a command-line interface to the **libav** library. These powerful tools have been thoroughly tested with a development history spanning almost 20 years. **ari** has been built with presets for FFmpeg which allow output videos to be compatible with multiple platforms, including the YouTube and Coursera players. These presets include specifying the bitrate, audio and video codecs, and the output video format. The numerous additional video specifications for customization can be applied to command-line arguments FFmpeg through **ari**.

We have developed a workflow with **ari** as the centerpiece for automatically generating educational videos. The narration script for lecture videos is stored in a plain text format, so that it can be synthesized into audio files via text-to-speech services. By storing lecture narration in plain text it can be updated, tracked, and collaboratively or automatically updated with version control software like Git and GitHub. If the figures in the lecture slides are created in a reproducible framework, such as generated using R code, the entire video can be reproducibly created and automatically updated. Thus, **ari** is the Automated R Instructor. We will provide examples of creating videos based on the following sets of source files: a slide deck built with R Markdown, a set of images and a script, or a presentation made with Google Slides or PowerPoint. The overview of the processes demonstrated in this paper are seen in Figure 1. We will also demonstrate the **ariExtra** package, which contains functions that connect **ari** to applications outside of the R ecosystem (Muschelli, 2020).

Configuring Ari

Ari relies on several software packages including FFmpeg, one of the most popular libraries for processing audio, video, and image files. Configuring FFmpeg can be challenging, therefore we have provided a Docker image so that Ari users can start producing videos quickly. A guide to getting

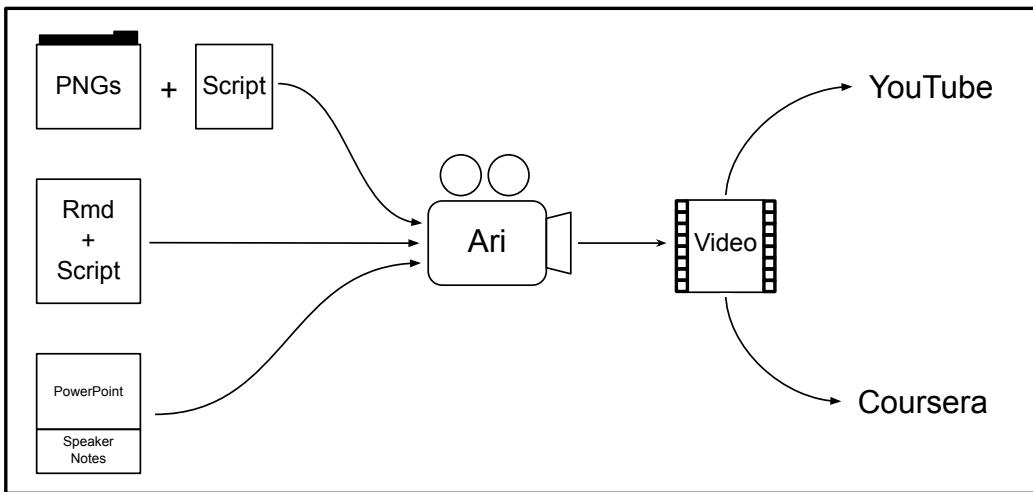


Figure 1: Ari is designed to fit into several existing workflows for creating lectures and presentations. Videos can be created with Ari from a series of images and a narrative script, from an R Markdown document, or from a PowerPoint presentation with speaker notes. Ari is pre-configured so that videos are ready to be uploaded to popular platforms like YouTube or Coursera.

started with Docker and using our Docker image is included with Ari as a vignette which can be accessed via `vignette("Simple-Ari-Configuration-with-Docker")`. Users who are interested in configuring Ari on their own may find the Dockerfile associated with the guide useful, and it is being actively developed at <https://github.com/seankross/ari-on-docker>.

Making videos with `ari: ari_stitch`

The main workhorse of `ari` is the `ari_stitch` function. This function requires an ordered set of images and an ordered set of audio objects, either paths to `wav` files or `tuneR` Wave objects, that correspond to each image. The `ari_stitch` function sequentially “stitches” each image in the video for the duration of its corresponding audio object using FFmpeg. FFmpeg must be installed so that `ari` can combine the audio and images, much like packages such as `animation` which have a similar requirement (Xie, 2013; Xie et al., 2018b). Moreover, on `shinyapps.io`, a dependency on the `animation` package will trigger an installation of FFmpeg so `ari` can be used on `shinyapps.io`. In the example below, 2 images (packaged with `ari`) are overlaid with white noise for demonstration. This example also allows users to check if the output of FFmpeg works with a desired video player.

```

library(tuneR)
library(ari)
result <- ari_stitch(
  ari_example(c("mab1.png", "mab2.png")),
  list(noise(), noise()),
  output = "noise.mp4"
)
isTRUE(result)

[1] TRUE
  
```

The output indicates whether the video was successfully created, but additional attributes are available, such as the path of the output file:

```

attributes(result)$outfile

[1] "noise.mp4"
  
```

The video for this output can be seen at <https://youtu.be/3kgaYf-EV90>.

Synthesizer authentication

The above example uses `tuneR::noise()` to generate audio and to show that any audio object can be used with `ari`. In most cases however, `ari` is most useful when combined with synthesizing audio using a text-to-speech system. Though one can generate the spoken audio in many ways, such as fitting a custom deep learning model, we will focus on using the aforementioned services (e.g. Amazon Polly) as they have straightforward public web APIs. One obstacle in using such services is that users must go through steps to provide authentication, whereas most of these APIs and the associated R packages do not allow for interactive authentication such as OAuth.

The `text2speech` package provides a unified interface to these 3 text-to-speech services, and we will focus on Amazon Polly and its authentication requirements. Polly is authenticated using the `aws.signature` package (Leeper, 2019). The `aws.signature` documentation provides options and steps to create the relevant credentials; we have also provided an additional `tutorial`. Essentially, the user must sign up for the service and retrieve public and private API keys and put them into their R profile or other areas accessible to R. Running `text2speech::tts_auth(service = "amazon")` will indicate if authentication was successful (if using a different service, change the `service` argument). NB: The APIs are generally paid services, but many have free tiers or limits, such as Amazon Polly's free tier for the first year (<https://aws.amazon.com/polly/pricing/>).

Creating speech from text: `ari_spin`

After Polly has been authenticated, videos can be created using the `ari_spin` function with an ordered set of images and a corresponding ordered set of text strings. This text is the “script” that is spoken over the images to create the output video. The number of elements in the text needs to be equal to the number of images. Let us take a part of Mercutio’s speech from Shakespeare’s Romeo and Juliet (Shakespeare, 2003) and overlay it on two images from the Wikipedia page about Mercutio (<https://en.wikipedia.org/wiki/Mercutio>):

```
speech <- c(
  "I will now perform part of Mercutio's speech from Shakespeare's Romeo and Juliet.",
  "O, then, I see Queen Mab hath been with you.
  She is the fairies' midwife, and she comes
  In shape no bigger than an agate-stone
  On the fore-finger of an alderman,
  Drawn with a team of little atomies
  Athwart men's noses as they lies asleep;"
)
mercutio_file <- "death_of_mercutio.png"
mercutio_file2 <- "mercutio_actor.png"

shakespeare_result <- ari_spin(
  c(mercutio_file, mercutio_file2),
  speech,
  output = "romeo.mp4", voice = "Joanna"
)
isTRUE(shakespeare_result)

[1] TRUE
```

The speech output can be seen at <https://youtu.be/SFhvM9g1OkE>. We chose the voice “Joanna” which is designated as a female sounding US-English speaker for the script. Each voice is language-dependent; we can see the available voices for English for Amazon Polly at <https://docs.aws.amazon.com/polly/latest/dg/SupportedLanguage.html>.

Though the voice generation is relatively clear, we chose a Shakespearean example to demonstrate the influence and production value of the variety of dialects available from these text-to-speech services. Compare the video of “Joanna” to the same video featuring “Brian” who “speaks” with a British English dialect:

```
gb_result <- ari_spin(
  c(mercutio_file, mercutio_file2),
  speech,
  output = "romeo_gb.mp4", voice = "Brian"
)
isTRUE(gb_result)
```

```
[1] TRUE
```

The resulting video can be seen at <https://youtu.be/fSS0JSb4VxM>.

The output video format is MP4 by default, but several formats can be specified via specifying the appropriate “muxer” for FFmpeg (see the function `ffmpeg_muxers`). Supported codecs can be founded using the functions `ffmpeg_audio_codecs` and `ffmpeg_video_codecs`. Additional options can be passed to FFmpeg from `ari_stitch` and `ari_spin` to customize the video to the necessary specifications.

We now discuss the number of image and script inputs that `ari` is designed to work with, including text files and a series of PNG images, presentations made with Google Slides or PowerPoint with the script written in the speaker notes section, or an HTML slide presentation created from an R Markdown file, where the script is written in the HTML comments.

Creating videos from R Markdown documents

Many R users have experience creating slide decks with R Markdown, for example using the `rmarkdown` or `xaringan` packages (Allaire et al., 2019; Xie et al., 2018a; Xie, 2018). In `ari`, the HTML slides are rendered using `webshot` (Chang, 2018) and the script is located in HTML comments (i.e. between `<!--` and `-->`). For example, in the file `ari_comments.Rmd` included in `ari`, which is an `ioslides` type of R Markdown slide deck, we have the last slide:

```
x <- readLines(ari_example("ari_comments.Rmd"))
tail(x[x != ""], 4)
\end{Sinput}
\begin{Soutput}
[1] "## Conclusion"
[2] "<!--"
[3] "Thank you for watching this video and good luck using Ari!"
[4] "-->"
```

The first words spoken on this example slide are “**Thank you**”. This setup allows for one plain text, version-controllable, integrated document that can reproducibly generate a video. We believe these features allow creators to make agile videos, that can easily be updated with new material or changed when errors or typos are found. Moreover, this framework provides an opportunity to translate videos into multiple languages, a feature that we will discuss in the future directions.

Using `ari_narrate`, users can create videos from R Markdown documents that create slide decks. An R Markdown file can be passed in, and the output will be created using the `render` function from `rmarkdown` (Allaire et al., 2019). If the slides are already rendered, the user can pass these slides and the original document, where the script is extracted. Passing rendered slides allows with the option for a custom rendering script. Here we create the video for `ari_comments.Rmd`, where the slides are rendered inside `ari_narrate`:

```
# Create a video from an R Markdown file with comments and slides
res <- ari_narrate(
  script = ari_example("ari_comments.Rmd"),
  voice = "Kendra",
  capture_method = "iterative"
)
```

The output video is located at https://youtu.be/rv9fg_qsqc0. In our experience with several users we have found that some HTML slides take more or less time to render when using `webshot`; for example they may be tinted with gray because they are in the middle of a slide transition when the image of the slide is captured. Therefore we provide the `delay` argument in `ari_narrate` which is passed to `webshot`. This can resolve these issues by allowing more time for the page to fully render, however this means it may take more time to create each video. We also provide the argument `capture_method` to allow for finely-tuned control of `webshot`. When `capture_method = "vectorized"`, `webshot` is run on the entire slide deck in a faster process, however we have experienced slide rendering issues with this setting depending on the configuration of an individual’s computer. However when `capture_method = "iterative"`, each slide is rendered individually in `webshot`, which solves many rendering issues, however it causes videos to be rendered more slowly. In the future, other HTML headless rendering engines (`webshot` uses PhantomJS) may be used if they achieve better performance, but we have found `webshot` to work well in most of our applications.

With respect to accessibility, `ari` encourages video creators to type out a script by design. This provides an effortless source of subtitles, rather than relying on other services such as YouTube to

provide speech-to-text subtitles. When using `ari_spin`, if the `subtitles` argument is TRUE, then an SRT file for subtitles will be created with the video.

One issue with synthesis of technical information is that changes to the script are required for Amazon Polly or other services to provide a correct pronunciation. For example, if you want the service to say “RStudio” or “ggplot2”, the phrases “R Studio” or “g g plot 2” must be written exactly that way in the script. These phrases will then appear in an SRT subtitle file, which may be confusing to a viewer. Thus, some post-processing of the SRT file may be needed.

Creating videos from other documents

We created the `ariExtra` (<https://github.com/muschellij2/ariExtra>) package to augment the core functionality of `ari` by extending it to software applications outside of the R ecosystem. These extensions require many additional dependencies, and considering the significant amount of setup already required for `ari`, we believed that this additional functionality should be in a separate package.

To create a video from a presentation made with Google Slides or PowerPoint, the slides should be converted to a set of images. We recommend using the PNG format for these images. To get the script for the video, we suggest putting the script for each slide in the speaker notes section of that slide. Several of the following features for video generation are in our package `ariExtra`. The speaker notes of slides can be extracted using `rgoogleSlides` (Noorazman, 2018) for Google Slides via the API or using `readOffice/officer` (Gohel, 2019; Ewing, 2017) to read from PowerPoint documents. Google Slides can be downloaded as a PDF and converted to PNGs using the `pdftools` package (Ooms, 2019). The `ariExtra` package also has a `pptx_notes` function for reading PowerPoint notes. Converting PowerPoint files to PDF can be done using LibreOffice and the `docxtractr` package (Rudis and Muir, 2019) which contains the necessary wrapper functions.

To demonstrate this, we use an example PowerPoint is located on Figshare (https://figshare.com/articles/Example_PowerPoint_for_ari/8865230). We can convert the PowerPoint to a PDF, then to a set of PNG images, then we extract the speaker notes.

```
pptx <- "ari.pptx"
download.file(paste0(
  "https://s3-eu-west-1.amazonaws.com/",
  "pfigshare-u-files/16252631/ari.pptx"
),
destfile = pptx
)
pdf <- docxtractr::convert_to_pdf(pptx) # >= 0.6.2
pngs <- pdftools::pdf_convert(pdf, dpi = 300)
notes <- ariExtra::pptx_notes(pptx)
notes

[1] "Sometimes it's hard for an instructor to take the time to record their lectures.  
For example, I'm in a coffee shop and it may be loud."
[2] "Here is an example of a plot with really small axes. We plot the x versus the y  
variables and a smoother between them."
```

The `ariExtra` package also can combine these processes and take multiple input types (Google Slides, PDFs, PPTX) and harmonize the output. The `pptx_to_ari` function combines the above steps:

```
doc <- ariExtra::pptx_to_ari(pptx)

Converting page 1 to /var/folders/1s/wrtqcpnx685_zk570bnx9_rr0000gr/T/
/Rtmpo6aD9u/filede6236136195.png... done!
Converting page 2 to /var/folders/1s/wrtqcpnx685_zk570bnx9_rr0000gr/T/
/Rtmpo6aD9u/filede62326b98ef.png... done!

doc[c("images", "script")]

$images
[1] "/private/var/folders/1s/wrtqcpnx685_zk570bnx9_rr0000gr/T/
Rtmpo6aD9u/filede6236058cc5_files/slide_1.png"
[2] "/private/var/folders/1s/wrtqcpnx685_zk570bnx9_rr0000gr/T/
```

```
Rtmpo6aD9u/filede6236058cc5_files/slide_2.png"
$script
[1] "Sometimes it's hard for an instructor to take the time to record their lectures.
For example, I'm in a coffee shop and it may be loud."
[2] "Here is an example of a plot with really small axes. We plot the x versus the
y-variables and a smoother between them."
```

This output can then be passed to `ari_spin`.

We will now demonstrate rendering the video with the “Kimberly” voice while using the `divisible_height` argument to forcibly scale the height of the images to be divisible by 2. This is required by the `x264` (default) codec which we have specified as a preset:

```
pptx_result <- ari_spin(pngs, notes,
  output = "pptx.mp4", voice = "Kimberly",
  divisible_height = TRUE, subtitles = TRUE
)
isTRUE(pptx_result)
```

You can see the output at <https://youtu.be/TBb3Am6xsQw>. Here we can see the first few lines of the subtitle file:

```
[1] "1"
[2] "00:00:00,000 --> 00:00:02,025"
[3] "Sometimes it's hard for an instructor to"
[4] "2"
[5] "00:00:02,025 --> 00:00:04,005"
[6] "take the time to record their lectures."
```

For Google Slides, the slide deck can be downloaded as a PowerPoint and the previous steps can be used, however it can also be downloaded directly as a PDF. We will use the same presentation, but uploaded to Google Slides. The `ariExtra` package has the function `gs_to_ari` to wrap this functionality (as long as link sharing is turned on), where we can pass the Google identifier:

```
gs_doc <- ariExtra::gs_to_ari("14gd2Di0CVKRNpFfLrryrGG7D3S8pu9aZ")
\end{Sinput}
\begin{Soutput}
Converting page 1 to
/var/folders/zw/14fv__6n4tnbk3xb31dnbt5m0000gn/T//RtmpphWBAj/filebd69651ed561.png...
done!
Converting page 2 to
/var/folders/zw/14fv__6n4tnbk3xb31dnbt5m0000gn/T//RtmpphWBAj/filebd694b4b0724.png...
done!
```

Note, as Google provides a PDF version of the slides, this obviates the LibreOffice dependency.

Alternatively, the notes can be extracted using `rgoogleSlides` via the Google Slides API, however this requires authentication so we will omit it here. Thus, we should be able to create videos using R Markdown, Google Slides, or PowerPoint presentations in an automatic fashion.

Summary

The `ari` package combines multiple open-source tools and APIs to create reproducible workflows for creating videos. These videos can be created using R Markdown documents, Google Slides, PowerPoint presentations, or simply a series of images. The audio overlaid on the images can be separate or contained within the storage of the images. These workflows can then be reproduced in the future and easily updated. As the current voice synthesis options are somewhat limited in the tenacity and inflection given, we believe that educational and informational videos are the most applicable area.

Future directions

The `ari` package is already being used to build data science curricula (Kross and Guo, 2019) and we look forward to collaborating with video creators to augment `ari` according to their changing needs. In the following section we outline possible directions for the future of the project.

Since **ari** is designed for teaching technical content, we plan to provide better support for the pronunciation of technical terms like the names of popular software tools. These names are usually not pronounced correctly by text-to-speech services because they are not words contained in the training data used in the deep learning models that these services are built upon. To address this concern we plan to compile a dictionary of commonly used technical terms and the phonetic phrasing and spelling of these terms that are required to achieve the correct pronunciation from text-to-speech services.

In addition to still images and synthesized voices, we would like to develop new technologies for incorporating other automatically generated videos into lectures generated by **ari**. As computer programming, statistics, and data science instructors we often rely on live coding (Chen and Guo, 2019) to demonstrate software tools to our students. Live coding videos suffer from many of the same problems as other kinds of technical videos as we addressed in the introduction. Therefore we plan to build a system for automating the creation of live coding videos. These videos would also be created using plain text documents like R Markdown. They would integrate synthesized narration with code chunks that would be displayed and executed according to specialized commands that would specify when code should be executed in an IDE like RStudio. These commands could also control which panes and tabs of the IDE are visible or emphasized.

As programmatic video creation software improves, we plan to extend **ari** so it can expand its compatibility with different technologies. For example we believe the heavy reliance on an FFmpeg installation can be mitigated in the future with advances in the **av** package. Though the **av** package has powerful functionality and is currently porting more from **libav** and therefore FFmpeg, it currently does not have the capabilities required for **ari**. Although third party installation from <https://ffmpeg.org/> can be burdensome to a user, package managers such as **brew** for OSX and **choco** for Windows provide an easier installation and configuration experience.

Although we rely on Amazon Polly for voice synthesis, other packages provide voice synthesis, such as **mscsts** for Microsoft and **googleLanguageR** for Google. We created the **text2speech** package to harmonize these synthesis options for **ari**. Thus, switching from one voice generation service to another simply involves switching the **service** and **voice** arguments in **ari**, assuming the service is properly authenticated. This ease of switching allows researchers to compare and test which voices and services are most effective at delivering content.

We see significant potential in how **ari** could expand global learning opportunities. Video narration scripts can be automatically translated into other languages with services like the [Google Translation API](#), where **googleLanguageR** provides an interface. Amazon Polly can speak languages other than English, meaning that one can write a lecture once and generate lecture videos in multiple languages. Therefore this workflow can greatly expand the potential audience for educational videos with relatively little additional effort from lecture creators. We plan to flesh out these workflows so that video creators can manage videos in multiple languages. We hope to add functionality so that communities of learners with language expertise can easily suggest modifications to automatically translated videos, and tooling so suggestions can be incorporated quickly.

Bibliography

- J. Allaire, Y. Xie, J. McPherson, J. Luraschi, K. Ushey, A. Atkins, H. Wickham, J. Cheng, W. Chang, and R. Iannone. *rmarkdown: Dynamic Documents for R*, 2019. URL <https://rmarkdown.rstudio.com>. R package version 1.12. [p]
- W. Chang. *webshot: Take Screenshots of Web Pages*, 2018. URL <https://CRAN.R-project.org/package=webshot>. R package version 0.5.1. [p]
- C. Chen and P. J. Guo. Improv: Teaching programming at scale via live coding. In *Proceedings of the Sixth Annual ACM Conference on Learning at Scale*, L@S '19, New York, NY, USA, 2019. ACM. URL <https://doi.org/10.1145/3330430.3333627>. [p]
- M. Edmondson. *googleLanguageR: Call Google's 'Natural Language' API, 'Cloud Translation' API, 'Cloud Speech' API and 'Cloud Text-to-Speech' API*, 2019. URL <http://code.markedmondson.me/googleLanguageR/>, <https://github.com/ropensci/googleLanguageR>. [p]
- M. Ewing. *readOffice: Read Text Out of Modern Office Files*, 2017. URL <https://CRAN.R-project.org/package=readOffice>. R package version 0.2.2. [p]
- D. Gohel. *officer: Manipulation of Microsoft Word and PowerPoint Documents*, 2019. URL <https://CRAN.R-project.org/package=officer>. R package version 0.3.5. [p]

- T. Hartsell and S. C.-Y. Yuen. Video streaming in online learning. *AACE Journal*, 14(1):31–43, 2006. [p]
- W.-J. Hsin and J. Cigas. Short videos improve student learning in online education. *Journal of Computing Sciences in Colleges*, 28(5):253–259, 2013. [p]
- S. Kross and P. J. Guo. End-user programmers repurposing end-user programming tools to foster diversity in adult end-user programming education. In *Proceedings of VL/HCC 2019: IEEE Symposium on Visual Languages and Human-Centric Computing*, VL/HCC ’19, 2019. ISBN 978-1-4503-5886-6. [p]
- T. J. Leeper. *aws.polly: Client for AWS Polly*, 2017. R package version 0.1.5. [p]
- T. J. Leeper. *aws.signature: Amazon Web Services Request Signatures*, 2019. R package version 0.5.1. [p]
- U. Ligges, S. Krey, O. Mersmann, and S. Schnackenberg. *tuneR: Analysis of Music and Speech*, 2018. URL <https://CRAN.R-project.org/package=tuneR>. [p]
- J. Muschelli. *mscstts: R Client for the Microsoft Cognitive Services 'Text-to-Speech' REST API*, 2019a. URL <https://CRAN.R-project.org/package=mscstts>. R package version 0.5.1. [p]
- J. Muschelli. *text2speech: Text to Speech*, 2019b. URL <https://github.com/muschellij2/text2speech>. R package version 0.2.4. [p]
- J. Muschelli. *ariExtra: Tools for Creating Automated Courses*, 2020. URL <https://CRAN.R-project.org/package=ariExtra>. R package version 0.2.7. [p]
- H. B. Noorazman. *rgoogleSlides: R Interface to Google Slides*, 2018. URL <https://CRAN.R-project.org/package=rgoogleSlides>. R package version 0.3.1. [p]
- J. Ooms. *pdftools: Text Extraction, Rendering and Converting of PDF Documents*, 2019. URL <https://CRAN.R-project.org/package=pdftools>. R package version 2.2. [p]
- B. Rudis and C. Muir. *docxtractr: Extract Data Tables and Comments from Microsoft Word Documents*, 2019. URL <http://gitlab.com/hrbrmstr/docxtractr>. R package version 0.6.2. [p]
- W. Shakespeare. *Romeo and Juliet*. Cambridge University Press, 2003. [p]
- A. Van Den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu. WaveNet: A generative model for raw audio. *SSW*, 125, 2016. [p]
- Y. Xie. *animation: An R package for creating animations and demonstrating statistical methods*. *Journal of Statistical Software*, 53(1):1–27, 2013. URL <http://www.jstatsoft.org/v53/i01/>. [p]
- Y. Xie. *xaringan: Presentation Ninja*, 2018. URL <https://CRAN.R-project.org/package=xaringan>. R package version 0.8. [p]
- Y. Xie, J. Allaire, and G. Grolemund. *R Markdown: The Definitive Guide*. Chapman and Hall/CRC, Boca Raton, Florida, 2018a. URL <https://bookdown.org/yihui/rmarkdown>. ISBN 9781138359338. [p]
- Y. Xie, C. Mueller, L. Yu, and W. Zhu. *animation: A Gallery of Animations in Statistics and Utilities to Create Animations*, 2018b. URL <https://yihui.name/animation>. R package version 2.6. [p]

Sean Kross

Department of Cognitive Science, University of California, San Diego
9500 Gilman Dr.
La Jolla, CA 92093
seankross@ucsd.edu

Jeffrey T. Leek

Department of Biostatistics, Johns Hopkins Bloomberg School of Public Health
615 N Wolfe Street

Baltimore, MD 21231
jtleek@jhu.edu

John Muschelli
Department of Biostatistics, Johns Hopkins Bloomberg School of Public Health
615 N Wolfe Street
Baltimore, MD 21231
jmusche1@jhu.edu

CopulaCenR: Copula based Regression Models for Bivariate Censored Data in R

by Tao Sun and Ying Ding

Abstract Bivariate time-to-event data frequently arise in research areas such as clinical trials and epidemiological studies, where the occurrence of two events are correlated. In many cases, the exact event times are unknown due to censoring. The copula model is a popular approach for modeling correlated bivariate censored data, in which the two marginal distributions and the between-margin dependence are modeled separately. This article presents the R package [CopulaCenR](#), which is designed for modeling and testing bivariate data under right or (general) interval censoring in a regression setting. It provides a variety of Archimedean copula functions including a flexible two-parameter copula and different types of regression models (parametric and semiparametric) for marginal distributions. In particular, it implements a semiparametric transformation model for the margins with proportional hazards and proportional odds models being its special cases. The numerical optimization is based on a novel two-step algorithm. For the regression parameters, three likelihood-based tests (Wald, generalized score and likelihood ratio tests) are also provided. We use two real data examples to illustrate the key functions in [CopulaCenR](#).

Introduction

Bivariate data arise frequently in many research areas such as health, epidemiology, and economics. For example, bivariate time-to-event endpoints are often used in clinical trials studying bilateral diseases (e.g., eye diseases) or complex diseases (e.g., cancer and psychiatric disorders). The two events are correlated as they come from the same subject. In many situations, the two event times cannot be precisely observed, leading to bivariate censored data. Specifically, bivariate right-censored data occur when the study ends prior to the occurrence of one or both events. An example of such data comes from a clinical study assessing the treatment effect on preventing blindness in Diabetic Retinopathy patients where each patient had one eye randomized to the treatment and the other eye received no treatment (Huster et al., 1989), and the time-to-blindness are bivariate and right-censored. We will illustrate the analysis of this study in Section 4. In another situation, bivariate interval-censored data occur when the status of both events are periodically examined at intermittent assessment times. In this case, the right censoring could also happen if the event still does not occur at the last assessment time. A special case of interval-censored data is the current status data if there is only one assessment time and the event is only known to occur or not by its assessment time. An example of bivariate interval-censored data will be demonstrated in Section 4, which came from a clinical trial studying the progression of a bilateral eye disease, Age-related Macular Degeneration (AMD), where the progression time to late-AMD are interval or right censored (AREDS Group, 1999). More examples can be found in books Hougaard (2000) and Sun (2007).

The development of our package is motivated by researches that are interested in (1) discovering covariates that are significantly associated with the bivariate censored outcomes, and (2) characterizing the joint and conditional risks of two events. For the bivariate events, the joint and conditional risks could be clinically more important than the marginal risk (of a single event). For example, the joint 5-year progression-free probability for both eyes helps identify patients with a high risk of progressing to late-AMD. For another example, for patients having one eye already progressed, the conditional 5-year progression-free probability for the non-progressed eye (given its fellow eye already progressed) provides important information for both clinicians and the patient since patients with both eyes progressed to the late stage of the disease may lose the ability to live independently.

There are three major approaches to fit regression models for bivariate censored data. The simplest way is to fit a marginal model and estimate the variance-covariance by a robust sandwich estimator (for example, Wei et al., 1989). This approach takes a working independence assumption, and thus cannot generate joint or conditional distributions. The second approach is based on frailty models (for example, Oakes, 1982), which are essentially mixed effects models and account for the dependence between two events by a latent frailty variable. However, the covariate effects in frailty models are usually interpreted on a conditional level (by conditioning on the frailty term), which is not straightforward. The third approach is to use copula models (for example, Clayton,

1978). Unlike the marginal or frailty approaches, the copula approach models the joint survival distribution by directly connecting the two marginal distributions through a copula function. One unique advantage of the copula is that it separately models the marginal distributions and the dependence parameter(s), allowing flexibility in marginal models and straightforward interpretation for covariate effects. Moreover, the challenge from censoring can be naturally handled through the marginal distributions within the copula function. Besides, the joint and conditional distributions can be obtained based on the copula model.

Along with these three major approaches, multiple endeavors have been devoted to the development of software, mostly R (R Core Team, 2019) packages, to build regression models for bivariate censored data. For bivariate right-censored data, the `survival` (Therneau, 2018b) package can fit parametric or semiparametric Cox (Cox, 1972) marginal and frailty models. Also, packages such as `parfm` (Munda et al., 2012) and `frailtypack` (Rondeau et al., 2012) implement proportional hazards (PH) frailty models under the parametric and semiparametric settings. Other R packages such as `coxme` (Therneau, 2018a) and `phmm` (Donohue and Xu, 2019) also fit PH frailty models for right-censored data. For bivariate interval-censored data, the `survival` and `frailtypack` packages provide marginal and frailty models under the parametric or semiparametric (Cox PH) situation, respectively. The C++ program `IntCens` (codes located under <https://dlin.web.unc.edu/software/intcens/>) implements a class of semiparametric frailty models, including both PH and proportional odds (PO) models.

To the best of our knowledge, there exists no R package for fitting copula-based regression models for both bivariate right-censored and interval-censored data. The existing copula packages for bivariate data handle either the non-censoring (i.e., complete data) or the right-censoring situation. In the non-censoring situation, the package `copula` (Hofert et al., 2018) by Yan (2007) and Kojadinovic and Yan (2010) implements multivariate copula models without covariates for complete data and obtains the maximum likelihood estimator for the copula dependence parameter. It gives useful codes for implementing regression models in bivariate complete data in the appendix of Yan (2007). It also provides copula goodness-of-fit tests for model selection purpose. The package `VineCopula` (Schepsmeier et al., 2018) can also model bivariate or multivariate complete data without covariates through the vine copula models (Aas et al., 2009). Packages such as `CopulaRegression` (Nicole Kraemer, 2014) and `gcmr` (Masarotto and Varin, 2017) can provide copula-based regression models with parametric margins for bivariate or multivariate complete data and provide maximum likelihood estimators for model parameters. The package `gamCopula` (Nagler and Vatter, 2020) implements a generalized additive model that can take into account the effect of the predictors on the dependence structure of bivariate and vine copula models (Vatter and Chavez-Demoulin, 2015). For the right-censoring situation, the `Copula.surv` package (Emura, 2018) can estimate the Clayton copula dependence parameter in bivariate right-censored data without covariates and also perform a goodness-of-fit test for a fitted Clayton model (Emura et al., 2010). The `Sunclarco` package (Prenen et al., 2017b) provides Clayton or Gumbel copula-based regression models with parametric (Weibull and piecewise constant) or Cox semiparametric margins for multivariate right-censored data (Prenen et al., 2017a). The package `GJRM` (Marra and Radice, 2020) can fit both marginal and copula regression models in complete and right-censored data (Marra and Radice, 2017; Marra et al., 2017; Marra and Radice, 2019). By far, there is no copula-based R package for bivariate interval-censored data.

We develop the `CopulaCenR` package, which fits copula-based regression models for both bivariate right-censored and interval-censored data. The package is available from the Comprehensive R Archive Network (CRAN) at <https://CRAN.R-project.org/package=CopulaCenR>. The main advantage of `CopulaCenR` relies on the diverse choice of copula and marginal models for both bivariate right-censored and interval-censored data. Specifically, it provides a class of Archimedean copulas that correspond to a variety of dependence structures, as illustrated in Table 1. In particular, in addition to these frequently used one-parameter Archimedean copulas, a two-parameter copula function (Copula2) is also included. This Copula2 has more flexibility in modeling dependence structure, as we show in Section 2. Furthermore, `CopulaCenR` implements a list of parametric and semiparametric marginal regression models, as illustrated in Table 2. For parameter estimation, the package utilizes a novel two-step procedure that is computationally stable and efficient. For the inference of regression parameters, three likelihood-based tests such as Wald, generalized score and likelihood ratio tests are provided.

We will describe the major features of `CopulaCenR` in Section 2 and presents the model and estimation procedure in Section 3. We will demonstrate two real data examples in Section 4 using the version 1.1.2 of `CopulaCenR`. Finally, we will conclude and discuss in Section 5.

Package Features

The most popular copula family for bivariate censored data is the Archimedean copula family, which has an explicit form of

$$C_\eta(u, v) = \phi_\eta\{\phi_\eta^{-1}(u) + \phi_\eta^{-1}(v)\},$$

where u and v are two uniformly distributed margins; ϕ_η is the generator function, which is a continuous, strictly decreasing and convex function; ϕ_η^{-1} is the inverse of ϕ_η . One generator function uniquely determines an Archimedean copula. The copula parameter η has a one-to-one correspondence with the popular dependence measure Kendall's τ . Another property of the copula is the tail dependence (i.e., τ_L and τ_U for lower and upper tail dependence), which measure the dependence between two margins in the lower and upper tails. More details about Archimedean copulas can be found in [Nelsen \(2006\)](#).

[Table 1](#) lists six Archimedean family copula models that are implemented in [CopulaCenR](#). Two most frequently used Archimedean copulas are Clayton ([Clayton, 1978](#)) and Gumbel ([Gumbel, 1960](#)) models, which account for the lower or upper tail dependence between two margins using a single parameter η . Other Archimedean copulas, such as Frank ([Frank, 1979](#)), Joe ([Joe, 1993](#)) and Ali-Mikhail-Haq (AMH) ([Ali et al., 1978](#)), are also one-parameter copulas. In addition to these five copulas, we also include a flexible two-parameter Archimedean copula model ([Joe and Hu, 1996](#); [Joe, 1997](#)), namely, Copula2 (also called the “BB1” family), which is formulated as

$$C_{\alpha,\kappa}(u, v) = [1 + \{(u^{-1/\kappa} - 1)^{1/\alpha} + (v^{-1/\kappa} - 1)^{1/\alpha}\}^\alpha]^{-\kappa}, \quad \alpha \in (0, 1], \quad \kappa \in (0, \infty). \quad (1)$$

The two dependence parameters (α and κ) are explicitly connected to Kendall's τ with $\tau = 1 - 2\alpha\kappa/(2\kappa + 1)$, and they account for the correlation between u and v at upper and lower tails. In particular, when $\alpha = 1$, Copula2 becomes the Clayton copula, and when $\kappa \rightarrow \infty$, it becomes the Gumbel copula. Thus, the two-parameter copula model provides more flexibility in modeling the between-margin dependence than the one-parameter copulas such as Clayton or Gumbel ([Joe, 2014](#)). [Figure 1](#) illustrates the scatter plots of bivariate event times generated from the six copula models in [Table 1](#).

Family	Parameter Space	Generator $\phi_\eta(t), t \in [0, \infty)$	Generator Inverse $\phi_\eta^{-1}(s), s \in (0, 1]$	τ_L	τ_U	Kendall's τ
Clayton	$\eta > 0$	$(1+t)^{-1/\eta}$	$s^{-\eta} - 1$	$2^{-1/\eta}$	0	$\eta/(2+\eta)$
Gumbel	$\eta \geq 1$	$\exp(-t^{1/\eta})$	$(-\log s)^\eta$	0	$2 - 2^{1/\eta}$	$1 - 1/\eta$
Frank	$\eta \geq 0$	$-\eta^{-1} \log(1 + e^{-t}(e^{-\eta} - 1))$	$-\log\{(e^{-\eta s} - 1)/(e^{-\eta} - 1)\}$	0	0	$1 + 4\{D_1(\eta) - 1\}/\eta$
AMH	$\eta \in [0, 1)$	$(1 - \eta)/(e^t - \eta)$	$\log\{[1 + \eta(s-1)]/s\}$	0	0	$1 - 2\{(1 - \eta)^2 \log(1 - \eta) + \eta\}/(3\eta^2)$
Joe	$\eta \geq 1$	$1 - (1 - e^{-t})^{1/\eta}$	$-\log\{1 - (1-s)^\eta\}$	0	$2 - 2^{1/\eta}$	$1 - 4 \sum_{k=1}^{\infty} 1/\{k(\eta k + 2)[\eta(k-1) + 2]\}$
Copula2	$\alpha \in (0, 1], \kappa > 0$	$\{1/(1+t^\alpha)\}^\kappa$	$(s^{-1/\kappa} - 1)^{1/\alpha}$	$2^{-\alpha\kappa}$	$2 - 2^\alpha$	$1 - 2\alpha\kappa/(2\kappa + 1)$

τ_L and τ_U are the lower and upper tail dependence measures.

$D_1(\cdot)$ is the Debye function written as $D_1(\eta) = \frac{1}{\eta} \int_0^\eta \frac{t}{e^t - 1} dt$.

Table 1: Summary of implemented Archimedean copula families.

To fit a copula-based regression model, one also needs to choose a regression model for the margins. [Table 2](#) lists the available marginal models in [CopulaCenR](#). For bivariate right-censored data, users can fit either a parametric marginal model via the function `rc_par_copula` or a semi-parametric Cox PH model via the function `rc_spCox_copula` ([Sun et al., 2019](#)). Specifically, the parametric models incorporate both the PH (e.g., Weibull, Gompertz) and the PO (e.g., Loglogistic) models. For bivariate interval-censored data, one can choose to fit a parametric marginal model via the function `ic_par_copula`. Moreover, the package can also fit a semiparametric transformation model via the function `ic_spTran_copula`. It contains a variety of marginal models including the PH and PO models, as we explain in [Section 3.3](#). A novel two-step sieve estimation procedure is implemented ([Sun and Ding, 2019](#)).

Type	Models	Survival Distributions $S(t)$	Corresponding R Functions
Parametric	Weibull	$\exp\{-(t/\lambda)^k e^{Z^\top \beta}\}$	
	Gompertz	$\exp\{-\frac{b}{a}(e^{at} - 1)e^{Z^\top \beta}\}$	<code>rc_par_copula, ic_par_copula</code>
	Loglogistic	$\{1 + (t/\lambda)^k e^{Z^\top \beta}\}^{-1}$	
Semiparametric	Cox	$\exp\{-\Lambda(t)e^{Z^\top \beta}\}$	<code>rc_spCox_copula</code>
	Transformation	$\exp[-G\{\Lambda(t)e^{Z^\top \beta}\}]$	<code>ic_spTran_copula</code>

Table 2: Summary of implemented marginal models.

For the inference of the covariate effects, three types of likelihood-based tests are implemented in [CopulaCenR](#): the Wald test (built within `rc_par_copula`, `rc_spCox_copula`, `ic_par_copula`, and `ic_spTran_copula`), the generalized score test (`score_copula`) and the likelihood-ratio test

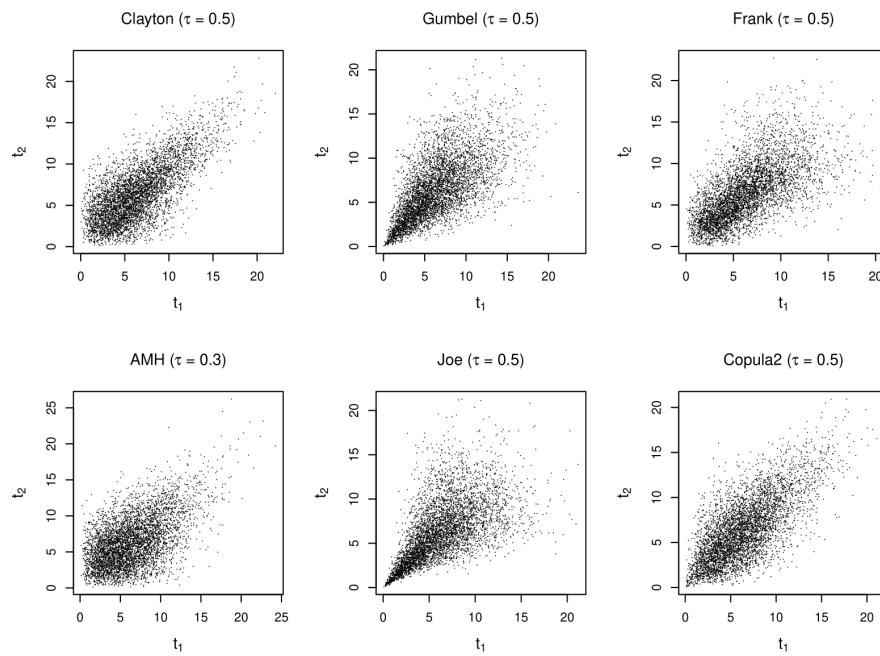


Figure 1: Scatter plots of bivariate event times generated from various copula models.

(`lrt_copula`).

After a copula model being fitted, fitted values (i.e., linear predictors, survival probabilities) can be extracted by the general S3 function `fitted`. For new observations, the linear predictors and survival probabilities can be obtained using the function `predict`. Moreover, the user can plot three types of distributions (joint, conditional and marginal) using the general functions `plot` and `lines`. In particular, an interactive 3D contour will be plotted to visualize the joint distribution.

Besides, the package provides a bivariate event time generating function `data_sim_copula`, which can generate random bivariate event times based on a specified copula function, a marginal distribution, and covariate values.

In summary, the key functions of `CopulaCenR` are

- `rc_par_copula`: for fitting parametric regression models to bivariate right-censored data;
- `rc_spCox_copula`: for fitting a semiparametric Cox regression model to bivariate right-censored data;
- `ic_par_copula`: for fitting parametric regression models to bivariate interval-censored data;
- `ic_spTran_copula`: for fitting a semiparametric transformation model to bivariate interval-censored data;
- `score_copula`: for performing the generalized score test on covariate effects;
- `lrt_copula`: for performing the likelihood ratio test (LRT) on covariate effects between two nested models;
- `tau_copula`: for calculating Kendall's τ from copula parameter estimates;
- `plot,lines`: S3 methods for plotting joint, conditional and marginal distributions based on a fitted copula model;
- `fitted,predict`: S3 methods for extracting fitted values and predicting new observations;
- `summary,print,coef,logLik,AIC,BIC`: other S3 functions for a fitted object;
- `data_sim_copula`: for generating bivariate event times through a specified copula model and marginal distributions.

We use two real data examples to illustrate the implementation of these functions in Section 4.

Methods

Copula model for bivariate censored data

Let (T_1, T_2) be the true bivariate event times, with marginal survival functions $S_j(t_j) = \Pr(T_j > t_j)$, $j = 1, 2$, and joint survival function $S(t_1, t_2) = \Pr(T_1 > t_1, T_2 > t_2)$. Assume there are n independent subjects in a study. When (T_1, T_2) are subject to right-censoring, for subject $i = 1 \dots n$, we observe $D_i = \{(Y_{ij}, \Delta_{ij}, Z_{ij}) : Y_{ij} = \min(T_{ij}, C_{ij}), \Delta_{ij} = I(T_{ij} \leq C_{ij}), j = 1, 2\}$, where C_{ij} is the censoring time of T_{ij} , Δ_{ij} is the censoring indicator and Z_{ij} is the covariate vector. When (T_1, T_2) are under interval-censoring, we observe $D_i = \{(L_{ij}, R_{ij}, Z_{ij}), j = 1, 2\}$ for subject i , where $[L_{ij}, R_{ij}]$ is the time interval that T_{ij} lies in and Z_{ij} is the covariate vector.

By the Sklar's theorem (Sklar, 1959), so long as the marginal survival functions S_j are continuous, there exists a unique function C_η that connects two marginal survival functions into the joint survival function: $S(t_1, t_2) = C_\eta\{S_1(t_1), S_2(t_2)\}$, $t_1, t_2 \geq 0$. Here, the function C_η is called a copula and its parameter η measures the dependence between the two margins. A signature feature of the copula is that it allows the dependence to be modeled separately from the marginal distributions.

Joint likelihood functions for bivariate censored data

In this section, we present the joint likelihood functions for bivariate right-censored data and bivariate interval-censored data, respectively.

Define the density function for copula $C_\eta(u, v)$ as $c_\eta(u, v) = \partial^2 C_\eta(u, v) / \partial u \partial v$. Let $f(t_1, t_2) = \partial^2 S(t_1, t_2) / \partial t_1 \partial t_2 = c_\eta\{S_1(t_1), S_2(t_2)\} f_1(t_1) f_2(t_2)$ denote the corresponding density function of $S(t_1, t_2)$. Denote by $\theta = (\beta^\top = (\beta_1^\top, \beta_2^\top), \eta, S_{01}, S_{02})^\top$ all the unknown parameters in $S(t_1, t_2)$, where β_j is the regression coefficient vector and S_{0j} is the baseline survival function for the j th margin. Then, the joint likelihood for the observed data $D = \{D_i\}_{i=1}^n$ can be written as

$$\begin{aligned} L_n(\theta|D) &= \prod_{i=1}^n f(y_{i1}, y_{i2}|Z_{i1}, Z_{i2})^{\delta_{i1}\delta_{i2}} \times \left[-\frac{\partial S(y_{i1}, y_{i2}|Z_{i1}, Z_{i2})}{\partial y_{i1}} \right]^{\delta_{i1}(1-\delta_{i2})} \\ &\quad \times \left[-\frac{\partial S(y_{i1}, y_{i2}|Z_{i1}, Z_{i2})}{\partial y_{i2}} \right]^{(1-\delta_{i1})\delta_{i2}} \times S(y_{i1}, y_{i2}|Z_{i1}, Z_{i2})^{(1-\delta_{i1})(1-\delta_{i2})} \\ &= \prod_{i=1}^n [c_\eta\{S_1(y_{i1}|Z_{i1}), S_2(y_{i2}|Z_{i2})\} f_1(y_{i1}|Z_{i1}) f_2(y_{i2}|Z_{i2})]^{\delta_{i1}\delta_{i2}} \\ &\quad \times \left[-\frac{\partial C_\eta\{S_1(y_{i1}|Z_{i1}), S_2(y_{i2}|Z_{i2})\}}{\partial y_{i1}} \right]^{\delta_{i1}(1-\delta_{i2})} \\ &\quad \times \left[-\frac{\partial C_\eta\{S_1(y_{i1}|Z_{i1}), S_2(y_{i2}|Z_{i2})\}}{\partial y_{i2}} \right]^{(1-\delta_{i1})\delta_{i2}} \\ &\quad \times C_\eta\{S_1(y_{i1}|Z_{i1}), S_2(y_{i2}|Z_{i2})\}^{(1-\delta_{i1})(1-\delta_{i2})}, \end{aligned} \tag{2}$$

where $(\delta_{i1}, \delta_{i2}) \in \{(0, 0), (0, 1), (1, 0), (1, 1)\}$.

Similarly, using the notation introduced in Section 3.1, the joint likelihood function for bivariate interval-censored data from n subjects can be written as

$$\begin{aligned} L_n(\theta|D) &= \prod_{i=1}^n \Pr(L_{i1} < T_{i1} \leq R_{i1}, L_{i2} < T_{i2} \leq R_{i2}|Z_{i1}, Z_{i2}) \\ &= \prod_{i=1}^n \left[\Pr(T_{i1} > L_{i1}, T_{i2} > L_{i2}|Z_{i1}, Z_{i2}) - \Pr(T_{i1} > L_{i1}, T_{i2} > R_{i2}|Z_{i1}, Z_{i2}) \right. \\ &\quad \left. - \Pr(T_{i1} > R_{i1}, T_{i2} > L_{i2}|Z_{i1}, Z_{i2}) + \Pr(T_{i1} > R_{i1}, T_{i2} > R_{i2}|Z_{i1}, Z_{i2}) \right] \\ &= \prod_{i=1}^n \left[C_\eta\{S_1(L_{i1}|Z_{i1}), S_2(L_{i2}|Z_{i2})\} - C_\eta\{S_1(L_{i1}|Z_{i1}), S_2(R_{i2}|Z_{i2})\} \right. \\ &\quad \left. - C_\eta\{S_1(R_{i1}|Z_{i1}), S_2(L_{i2}|Z_{i2})\} + C_\eta\{S_1(R_{i1}|Z_{i1}), S_2(R_{i2}|Z_{i2})\} \right]. \end{aligned} \tag{3}$$

The right interval R_{ij} can take values in $(0, \infty]$. For a given subject i , if $R_{ij} = \infty$ (i.e., T_{ij} is right-censored), then any term involving R_{ij} becomes 0 and the joint survival function for subject i reduces to only one (if both R_{i1} and R_{i2} are ∞) or two (if one R_{ij} is ∞) terms. The special case of bivariate current status data (i.e., only one assessment time for each subject) can also fit into this framework, where for each T_{ij} , either $L_{ij} = 0$ (T_{ij} is smaller than the assessment time, which is R_{ij} in this case) or $R_{ij} = \infty$ (T_{ij} is greater than the assessment time, which is L_{ij} in this case). Therefore, the likelihood function (3) can handle the bivariate data under general interval-censoring.

Marginal models

We implement several popular parametric marginal models in **CopulaCenR**, as shown in Table 2. For example, the marginal Weibull survival distribution can be written as

$$S_j(t_j|Z_j) = \exp\{-(t_j/\lambda_j)^{k_j} e^{Z_j^\top \beta_j}\}, \quad j = 1, 2,$$

where λ_j and k_j are the scale and shape parameters of the baseline Weibull distribution, and β_j are the covariate effects. The model follows the PH assumption. In this case, the parameter set θ becomes $(\beta^\top, \eta, \lambda_1, k_1, \lambda_2, k_2)^\top$. Other parametric distributions including Gompertz and Loglogistic are also implemented in the package.

More generally, we implement the semiparametric Cox PH marginal model for bivariate right-censored data. The model does not specify the marginal distribution for the baseline hazards function. Instead, the baseline hazards are treated as piecewise constants between all uncensored event times as suggested by Breslow (1972). The model is expressed as

$$S_j(t_j|Z_j) = \exp\{-\Lambda_j(t_j)e^{Z_j^\top \beta_j}\}, \quad j = 1, 2,$$

in which the Breslow baseline cumulative hazard function $\Lambda_j(t)$ is given by

$$\Lambda_j(t) = \sum_{i=1}^n \frac{I(Y_{ij} \leq t)\delta_{ij}}{\sum_{k \in R_{ij}} \exp Z_k^\top \beta_j},$$

where $R_{ij} = \{k : Y_k \geq Y_{ij}\}$ denotes the at-risk set at time Y_{ij} .

We also consider a class of semiparametric linear transformation models for the marginal distribution of the interval-censored data. The model is expressed as:

$$S_j(t_j|Z_j) = \exp[-G_j\{\Lambda_j(t_j)e^{Z_j^\top \beta_j}\}], \quad j = 1, 2. \quad (4)$$

$\Lambda_j(\cdot)$ is an unknown and non-decreasing function of t , which is not necessarily the baseline cumulative hazards function. In **CopulaCenR**, we approximate Λ_j in a sieve space constructed by Bernstein polynomials. A Bernstein basis polynomial with degree m is expressed as:

$$B_k(t, m, l, u) = \binom{m}{k} \left(\frac{t-l}{u-l}\right)^k \left(1 - \frac{t-l}{u-l}\right)^{m-k}, \quad k = 0, \dots, m, \quad (5)$$

where l and u are the lower and upper bounds of all observed times. One big advantage of Bernstein polynomials is that they do not require the specification of interior knots, as seen from (5), making them easy to work with. More details can be found in Sun and Ding (2019).

In model (4), $G_j(\cdot)$ is a pre-specified strictly increasing function, such as the Box-Cox and the logarithmic transformation functions. The package uses a $G(\cdot)$ function as specified in Zhou et al. (2017):

$$G_j(x) = \begin{cases} \frac{(1+x)^r - 1}{r}, & 0 < r \leq 2, \\ \frac{\log\{1+(r-2)x\}}{r-2}, & r > 2. \end{cases} \quad (6)$$

Note that the model (4) contains a class of survival models. For example, when $G(x) = x$ at $r = 1$, the marginal function $S_j(t|Z)$ becomes $\exp\{-\Lambda_j(t)e^{Z^\top \beta_j}\}$, which is essentially a PH model. Likewise, when $G_j(x) = \log(1+x)$ at $r = 3$, $S_j(t|Z)$ becomes $\{1 + \Lambda_j(t)e^{Z^\top \beta_j}\}^{-1}$, which is a PO model. In practice, the value of r can be either selected according to model AIC or treated unknown and estimated together with other model parameters.

Two-step estimation procedure

In this section, we illustrate the estimation procedure for the unknown parameter θ . For simplicity, we use the general notation $\theta = (\beta_1^\top, \beta_2^\top, \eta, S_{01}, S_{02})^\top$ throughout this section. In principle, we can maximize the joint log-likelihood function based on formula (2) or (3) directly, written as $l_n(\theta|D) = \log L_n(\theta|D) = \sum_{i=1}^n \log L(\theta|D_i)$. Due to the complex structure of the log-likelihood function, we implement a novel two-step estimation procedure, which is proven to be computationally more stable and efficient than the one-step procedure, as shown in Sun et al. (2019) and Sun and Ding (2019). Essentially, the two-step procedure implements an extra step to obtain appropriate initial values for all the unknown parameters. The estimation procedure is described below:

1. Obtain initial estimates of θ_n :

- (a) $(\hat{\beta}_{jn}^{(1)}, \hat{S}_{0j}^{(1)}) = \arg \max_{(\beta_j, S_{0j})} l_{jn}(\beta_j, S_{0j})$, where l_{jn} denotes the log-likelihood for the marginal model, $j = 1, 2$;
- (b) $\hat{\eta}_n^{(1)} = \arg \max_{(\eta)} l_n\{\hat{\beta}_n^{(1)} = (\hat{\beta}_{1n}^{(1)}, \hat{\beta}_{2n}^{(1)}), \eta, \hat{S}_{01}^{(1)}, \hat{S}_{02}^{(1)}\}$, where $\hat{\beta}_{jn}^{(1)}$ and $\hat{S}_{0j}^{(1)}$ are the initial estimates from (a), and l_n is the joint log-likelihood.

2. Simultaneously maximize the joint log-likelihood to get final estimates:

$$\hat{\theta}_n = (\hat{\beta}_n, \hat{\eta}_n, \hat{S}_{01}, \hat{S}_{02}) = \arg \max_{(\beta, \eta, S_{01}, S_{02})} l_n(\beta, \eta, S_{01}, S_{02}) \text{ with initial values } (\hat{\beta}_n^{(1)}, \hat{\eta}_n^{(1)}, \hat{S}_{01}^{(1)}, \hat{S}_{02}^{(1)}) \text{ obtained from step 1(a) and 1(b).}$$

[Remark 1.] In the case of semiparametric Cox PH margins (with the Breslow baseline cumulative hazard estimator), although the maximum likelihood estimators from step 2 are consistent and asymptotically normal, the Hessian matrix cannot be directly used for estimating the variance-covariance matrix of $(\hat{\beta}, \hat{\eta})$ (Sun et al., 2019). Therefore, the bootstrap procedure is implemented in the package for producing a valid variance-covariance estimator.

[Remark 2.] In the case of semiparametric transformation model margins (with the use of Bernstein polynomials), the two-step estimation procedure becomes a two-step “sieve” estimation procedure. Sun and Ding (2019) rigorously proved the asymptotic properties of the sieve maximum likelihood estimators.

The main model-fitting functions (`rc_par_copula`, `rc_spCox_copula`, `ic_par_copula` and `ic_spTran_copula`) provide a built-in optimization option, which is a wrapper to the optimization routines `optim` and `nlm` in R.

Likelihood-based tests for covariate effects

We now separate β into two parts: β_g and β_{ng} , where β_g is the parameter set of interest for hypothesis testing and β_{ng} denotes the rest of the regression coefficients. In certain cases, β_g can be the entire regression parameter β . The package implements three likelihood-based tests including the Wald test, the generalized score test (Cox and Hinkley, 1979) and the likelihood ratio test, which are asymptotically equivalent and follow the chi-squared distribution with $df = \dim(\beta_g)$. In particular, the generalized score test is usually faster than the other two tests for large-scale testings such as the genome-wide association study (GWAS) (Sun et al., 2019; Sun and Ding, 2019). Due to the complex structure of the joint log-likelihood, instead of analytically deriving the first and second order derivatives, we use the Richardson’s extrapolation (Lindfield et al., 1989) to approximate the score function and observed Fisher information numerically.

Examples

Bivariate event time generation

The package `CopulaCenR` provides a user-friendly function `data_sim_copula` for generating random bivariate event times based on a specified copula model, marginal distributions and covariate values. The arguments `n`, `copula`, and `eta` assign the sample size, the copula type, and the dependence parameter value. For marginal distributions, the argument `dist` can be one of the three parametric distributions in Table 2 (i.e., Weibull, Loglogistic and Gompertz), and their distribution parameters are given through `baseline`. For Weibull and Loglogistic, the baseline parameters are λ (scale) and k (shape); whereas a (shape) and b (rate) for the Gompertz distribution. In this current version, we assume that the two margins share the same set of covariates and effects, which are assigned by

`var_list` and `COV_beta`, respectively. Lastly, `x1` and `x2` input a data frame of covariate values for the two margins, respectively. Figure 2 illustrates a scatter plot of 500 simulated bivariate event times from a Clayton model with Weibull margins, as demonstrated in the code below.

```
library(CopulaCenR)
set.seed(1)
dat <- data_sim_copula(n = 500, copula = "Clayton", eta = 3, dist = "Weibull",
                       baseline = c(0.1,2), var_list = c("var1", "var2"), COV_beta = c(0.1, 0.1),
                       x1 = cbind(rnorm(500, 6, 2), rbinom(500, 1, 0.5)),
                       x2 = cbind(rnorm(500, 6, 2), rbinom(500, 1, 0.5)))

head(dat)

  id ind      var1 var2     time
1  1   1  6.130533    1 8.062168
2  1   2  6.154606    1 7.472649
3  2   1  8.070653    1 6.317247
4  2   2  5.406263    1 5.904064
5  3   1 10.520432    1 4.195788
6  3   2  3.633516    1 4.771523

plot(x = dat$time[dat$ind == 1], y = dat$time[dat$ind == 2],
      xlab = expression(t[1]), ylab = expression(t[2]), cex.axis = 1, cex.lab = 1.3)
```

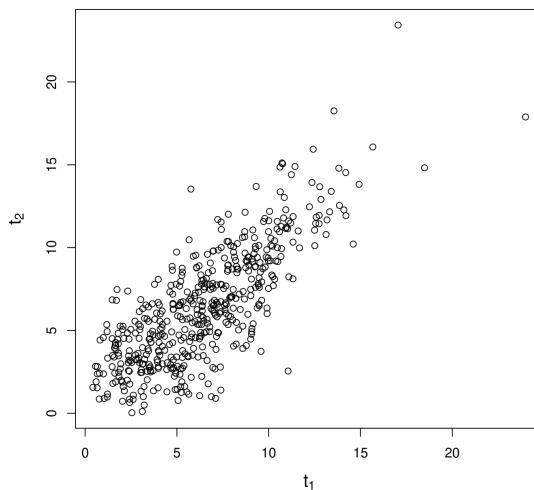


Figure 2: Simulated bivariate event times from the Clayton copula with Weibull margins.

Fitting copula models for bivariate right-censored data

The bivariate right-censored input dataset shall be a data frame including the covariates and four additional key input columns:

- id: the subject/cluster id,
- ind: the margin indicator (1, 2),
- obs_time: the exact observed time,
- status: censoring indicator (1 for event, 0 for right-censoring).

We use the DRS (Diabetic Retinopathy Study) data as an example. The DRS data contain bivariate right-censored time to blindness from 197 diabetic retinopathy patients. These patients were from a 50% random sample of the patients with "high-risk" diabetic retinopathy as defined by the DRS (Huster et al., 1989). Each patient had one eye randomized to one of the two laser treatments and the other eye received no treatment. For each eye, the event of interest was the time

from initiation of treatment to the time to blindness in months. Censoring was caused by death, dropout, or end of the study. The data can be loaded by

```
data("DRS", package = "CopulaCenR")
head(DRS)

id  ind obs_time status treat age type
5   1    46.23     0     0  28    2
5   2    46.23     0     2  28    2
14  1    42.50     0     2  12    1
14  2    31.30     1     0  12    1
16  1    42.27     0     1  9     1
16  2    42.27     0     0  9     1
```

There are three covariates: `treat` is treatment with 0 for no treatment, 1 for xenon laser treatment and 2 for argon laser treatment; `age` is the age at diagnosis of diabetes; `type` is the type of diabetes with 1 for juvenile (age ≤ 20 at diagnosis) and 2 for adult. The primary question of the DRS study was to assess the treatment effectiveness while accommodating the dependence between two eyes.

We now demonstrate how to fit a Clayton copula model with Weibull margins to the DRS data using the function `rc_par_copula`. We are interested in the treatment effect, as indicated in argument `var_list`. The arguments `copula` and `m.dist` specify the fitted copula model and marginal baseline distributions. The default optimization method is `BFGS` (Nash, 1990). Other optimization methods and control parameters can also be applied (see `?optim`).

```
library(CopulaCenR)
clayton_wb <- rc_par_copula(data = DRS, var_list = "treat", copula = "Clayton",
                               m.dist = "Weibull", method = "BFGS")

summary(clayton_wb)

Copula:  Clayton
Margin:  Weibull

      estimate        SE      stat     pvalue
lambda 90.6440318 13.1887218 47.2360 6.293e-12 ***
k       0.8062766  0.0586207 189.1758 < 2.2e-16 ***
treat1 -0.5714498  0.1997080   8.1878  0.004217 **
treat2  0.0052997  0.1739106   0.0009  0.975689
eta     0.6205855  0.2610638   5.6508  0.017447 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
(The Wald tests are testing whether each coefficient is 0)

Final llk:   -839.7212
Convergence is completed successfully
```

The estimation and Wald test results suggest the xenon treatment significantly reduced the risk of blindness compared to controls ($p = 0.004217$ for `treat1`). We also compared our estimates with previous findings in Huster et al. (1989). Due to the differences in the model parameterization, we first transformed our estimates into comparable forms. Specifically, the xenon treatment effect in Huster et al. (1989) can be expressed as $-k \log(\lambda) + treat1 = -4.20$ and similarly the argon treatment effect is $-k \log(\lambda) + treat2 = -3.63$, which are consistent with the reported estimates $(-4.20, -3.42)$ in the Table 2 (page 151) of Huster et al. (1989). The AIC and BIC of this model can be obtained from the S3 methods `AIC` and `BIC`.

```
AIC(clayton_wb)
```

```
1689.442
```

```
BIC(clayton_wb)
```

```
1705.858
```

After the model is fitted, Kendall's τ can be estimated through the function `tau_copula`.

```
tau_copula(eta = as.numeric(coef(clayton_wb)[["eta"]]), copula = "Clayton")
0.2368118
```

The fitted values (i.e., linear predictors and survival probabilities) can be extracted through the function `fitted`. As the model is a PH model, the linear predictors (`type` is “lp”) are the estimated log proportional hazards.

```
fit1 <- fitted(clayton_wb, type = "lp")
fit1[1:3, ]

id      lp1      lp2
5  0.000000000 0.005299655
14 0.005299655 0.000000000
16 -0.571449835 0.000000000
```

When `type` is “survival”, the fitted outputs are marginal (S_1, S_2) and joint (S_{12}) survival probabilities at the observed times (t_1, t_2).

```
fit2 <- fitted(clayton_wb, type = "survival")
fit2[1:3, ]

id   t1   t2      S1      S2      S12
5 46.23 46.23 0.5592967 0.5575724 0.3643588
14 42.50 31.30 0.5793467 0.6542323 0.4234880
16 42.27 42.27 0.7369175 0.5823995 0.4655204
```

Similarly, the `predict` function provides predictions for new observations with covariates. Its outputs can be either linear predictors or survival probabilities (at specified times). The following `newdata1` example contains two subjects under different treatments.

```
newdata1 <- data.frame(id = rep(1:2, each=2), ind = rep(c(1,2),2),
                       time = rep(40,4), treat = factor(c(0,1,0,2)))
newdata1

id ind time treat
1   1   40     0
1   2   40     1
2   1   40     0
2   2   40     2

predict(clayton_wb, newdata = newdata1, type = "lp")

id lp1      lp2
1   0 -0.571449835
2   0  0.005299655

predict(clayton_wb, newdata = newdata1, type = "survival")

id t1 t2      S1      S2      S12
1 40 40 0.5962669 0.7467754 0.4799705
2 40 40 0.5962669 0.5946309 0.4024998
```

Fitting copula models for bivariate interval-censored data

The bivariate interval-censored input dataset shall be a data frame including the covariates and five key input columns:

- id: the subject/cluster id,
- ind: the margin indicator (1 or 2),
- Left: the left bound of the observed interval,
- Right: the right bound of the observed interval (can take “Inf”),
- status: the censoring indicator (1 for left- or interval-censoring, 0 for right-censoring).

We use the AREDS (Age-Related Eye Disease Study) data as an example. The event of interest is the AMD (Age-related Macular Degeneration) disease, which is a leading cause of blindness in the developed world (Swaroop et al., 2009). It is known as a polygenic, progressive and neurodegenerative disorder. The AREDS study is a multi-center randomized clinical trial studying the development and progression of AMD, sponsored by the National Eye Institute (AREDS Group, 1999). Due to intermittent assessment times (every 6 months up to the first 6 years and every 1 year since after), the exact time when each eye progressed to late-AMD was only known to lie in a certain interval. As a result, the outcome data are bivariate interval-censored. The package includes a subset data of 629 Caucasian participants from AREDS who had at least one eye in moderate AMD stage at baseline. The data can be loaded by

```
data("AREDS", package = "CopulaCenR")
head(AREDS)
```

id	ind	Left	Right	status	SevScaleBL	ENROLLAGE	rs2284665
1	1	0.0	2.0	1	6	67.0	1
1	2	0.0	2.0	1	8	67.0	1
2	1	0.0	2.0	1	7	68.0	0
2	2	5.9	9.3	1	4	68.0	0
3	1	8.0	9.1	1	7	64.9	0
3	2	10.0	Inf	0	7	64.9	0

Out of these 629 subjects, 273 subjects developed late-AMD in both eyes during the study and the times to late-AMD were interval-censored; 138 subjects developed late-AMD in one eye (interval-censored) and did not develop late-AMD before the end of the study (right-censored); the rest 218 subjects were right-censored for late-AMD in both eyes.

There are three continuous covariates: **SevScaleBL** for baseline AMD severity score (a value between 1 and 8 with a higher value indicating more severe AMD), **ENROLLAGE** for baseline age and **rs2284665** for a genetic variant (0, 1, 2 for *GG*, *GT*, *TT*) that might be associated with AMD progression. The two clinical covariates **SevScaleBL** and **ENROLLAGE** are well-known risk factors of AMD. Thus, our primary interest is to find out whether the genetic variant **rs2284665** is significantly associated with AMD progression.

We fit a two-parameter copula semiparametric transformation model for the AREDS data through the function **ic_spTran_copula**. The arguments **l** and **u** are the range of event times, which need to be pre-specified by the user. In practice, **l** and **u** can be set as the minimum and maximum of observed times. The argument **m** corresponds to the degree of Bernstein polynomials (as shown in formula 5), with the default value $m = 3$. The argument **r** specifies the form of marginal transformation model (as shown in formula 6). In practice, the values of **m** and **r** can be chosen based on the smallest AIC for a list of fitted models with different values.

We now demonstrate how to fit a two-parameter copula semiparametric model to the AREDS data. We chose the range of event times as $l = 0$ and $u = 15$, use the default Bernstein polynomial degree as $m = 3$ and assume PO for the margins (i.e., $r = 3$).

```
library(CopulaCenR)
copula2_sp <- ic_spTran_copula(data = AREDS, copula = "Copula2",
                                 var_list = c("ENROLLAGE", "rs2284665", "SevScaleBL"),
                                 l = 0, u = 15, m = 3, r = 3)
summary(copula2_sp)

Copula: Copula2
Margin: semiparametric

      estimate       SE     stat   pvalue
ENROLLAGE 0.042610 0.012271 12.057 0.0005159 ***
rs2284665 0.397712 0.091180 19.026 1.290e-05 ***
SevScaleBL 0.722681 0.053258 184.132 < 2.2e-16 ***
alpha        0.930508 0.058714 251.167 < 2.2e-16 ***
kappa        0.974037 0.226081 18.562 1.645e-05 ***
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
(The Wald tests are testing whether each coefficient is 0)

Final llk: -2104.178
Convergence is completed successfully
```

From the output, the estimated odds ratio for the genetic variant *rs2284665* is $\exp(0.397712) = 1.49$ with a *p*-value 1.29×10^{-5} , implying it has a “harmful” effect for AMD patients by having more copies of its minor allele *T*. The AIC and BIC values are 4226.356 and 4266.353, respectively.

```
AIC(copula2_sp)
```

```
4226.356
```

```
BIC(copula2_sp)
```

```
4266.353
```

Also, the estimated Kendall’s τ is 0.38, suggesting moderate dependence in AMD progression between two eyes.

```
tau_copula(eta = as.numeric(coef(copula2_sp)[c("alpha","kappa")]),
            copula = "Copula2")
```

```
0.3851248
```

Furthermore, we can test the effect of *rs2284665* by the generalized score test. We first fit a null model without *rs2284665* and then test its effect using the function `score_copula`.

```
copula2_sp_null <- ic_spTran_copula(data = AREDS, copula = "Copula2",
                                       var_list = c("ENROLLAGE","SevScaleBL"),
                                       l = 0, u = 15, m = 3, r = 3)
score_copula(object = copula2_sp_null, var_score = "rs2284665")

      stat      pvalue
1.943163e+01 1.042661e-05
```

The LRT can also be performed by applying two nested models to the function `lrt_copula`.

```
lrt_copula(model1 = copula2_sp, model2 = copula2_sp_null)

      stat      pvalue
9.543119588 0.002007003
```

The following codes plot the 3D joint survival probabilities for the three subjects in `newdata2`, which have the same `SevScaleBL` = 3 in both eyes and `ENROLLAGE` = 60, but vary in the genotype of `rs2284665`. In the `plot` function, the argument `class` specifies the plot type, which can be one of “joint”, “conditional” and “marginal”. When `class` = “joint”, it generates a 3D interactive contour that can be manually rotated for the desired visualization. Figure 3 is a snapshot of 3D contours for the three subjects in `newdata2`.

```
newdata2 <- data.frame(id = rep(1:3, each=2), ind = rep(c(1,2),3),
                        SevScaleBL = rep(3,6), ENROLLAGE = rep(60,6),
                        rs2284665 = c(0,0,1,1,2,2))
newdata2
```

id	ind	SevScaleBL	ENROLLAGE	rs2284665
1	1	3	60	0
1	2	3	60	0
2	1	3	60	1
2	2	3	60	1
3	1	3	60	2
3	2	3	60	2

```
plot(x = copula2_sp, class = "joint", newdata = newdata2)
```

Similarly, the conditional survival probabilities (Figure 4) can be obtained for the left eyes from the same three subjects, given their right eyes (i.e., `cond_margin` = 2) had progressed (to late-AMD) at year 5 (i.e., `cond_time` = 5).

```
plot(x = copula2_sp, class = "conditional", newdata = newdata2,
      cond_margin = 2, cond_time = 5, ylim = c(0.25,1),
      ylab = "Conditional Survival Probability")
```

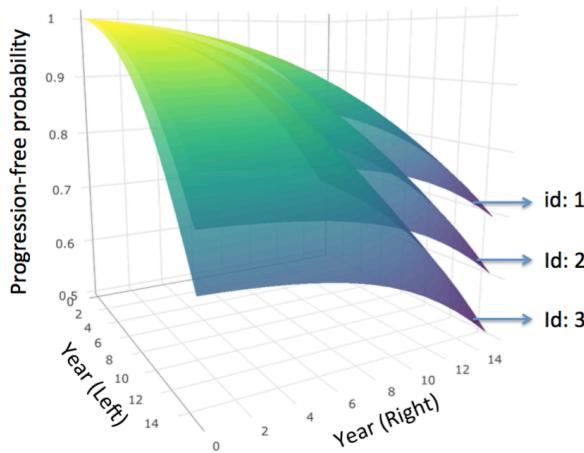


Figure 3: Estimated joint progression-free probability contours for subjects with different genotypes of *rs2284665* (age 60 and severity score 3 in both eyes).

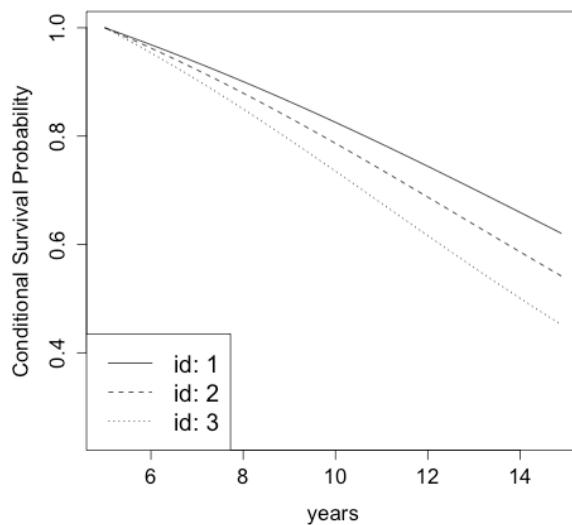


Figure 4: Estimated conditional progression-free probability of remaining years (after year 5) for the left eye, given the right eye has progressed by year 5, for subjects with different genotypes of *rs2284665* (age 60 and severity score 3 in both eyes).

Likewise, we can also obtain the eye-level marginal survival probabilities (i.e., `plot_margin = 1` for the left eyes) for the same three subjects, as illustrated in Figure 5.

```
plot(x = copula2_sp, class = "marginal", newdata = newdata2,
      plot_margin = 1, ylim = c(0.6,1), ylab = "Marginal Survival Probability")
```

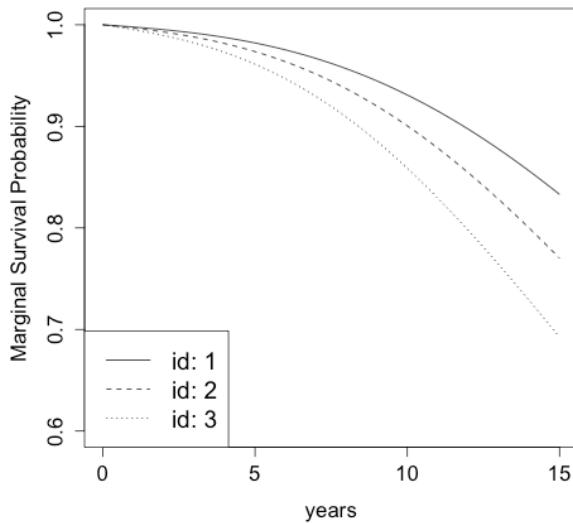


Figure 5: Estimated marginal progression-free probability for one eye from subjects with different genotypes of *rs2284665* (age 60 and severity score 3 in both eyes).

Summary

This paper presents the R package [CopulaCenR](#) for implementing copula-based regression models in bivariate censored data, including both bivariate right-censored data and bivariate interval-censored data. A variety of Archimedean copulas, including a flexible two-parameter copula, are built in the package to accommodate different dependence structures. Moreover, the package can fit various parametric and semiparametric regression models for the two margins within the copula function. In particular, a general semiparametric transformation model with PH and PO models being its special cases is implemented for the margins in this package. For parameter estimation, a novel two-step procedure is adopted to guarantee stable and fast computation. For the inference of covariate effects, all three likelihood-based tests are provided. Lastly, two real data examples are given to demonstrate the key features and capabilities of this package.

One future extension of this package is to allow multivariate copula functions for handling multivariate censored events. Another important research extension is to add goodness-of-fit tests, which is critical for choosing a proper copula model. However, there are limited works in testing copula models in bivariate censored data, especially in bivariate interval-censored data under the regression setting. The current literature (e.g., [Shih, 1998](#); [Andersen et al., 2010](#); [Emura et al., 2010](#); [Wang, 2010](#)) only focus on testing copulas in bivariate right-censored data without covariates. We are currently investigating these directions and plan to incorporate them in a future version of [CopulaCenR](#).

Acknowledgments

The Authors are grateful to Dr. Wei Chen for providing valuable suggestions about package development and the AREDS data analysis.

Bibliography

- K. Aas, C. Czado, A. Frigessi, and H. Bakken. Pair-copula constructions of multiple dependence. *Insurance: Mathematics and Economics*, 44(2):182 – 198, 2009. URL <https://doi.org/10.1016/j.insmatheco.2007.02.001>. [p]
- M. M. Ali, N. N. Mikhail, and M. S. Haq. A class of bivariate distributions including the bivariate logistic. *Journal of Multivariate Analysis*, 8(3):405–412, 1978. URL [https://doi.org/10.1016/0047-259X\(78\)90063-5](https://doi.org/10.1016/0047-259X(78)90063-5). [p]
- P. K. Andersen, C. T. Ekstrom, J. P. Klein, Y. Shu, and M.-J. Zhang. A class of goodness of fit tests for a copula based on bivariate right-censored data. *Biometrical Journal*, 47(6):815–824, 2010. URL <https://doi.org/10.1002/bimj.200410163>. [p]
- AREDS Group. The Age-Related Eye Disease Study (AREDS): Design implications. AREDS report no. 1. *Controlled Clinical Trials*, 20(6):573–600, 1999. URL [https://doi.org/10.1016/s0197-2456\(99\)00031-8](https://doi.org/10.1016/s0197-2456(99)00031-8). [p]
- N. E. Breslow. Discussion of the paper by D. R. Cox. *Journal of the Royal Statistical Society: Series B*, 34:216–217, 1972. URL <https://doi.org/10.2307/1403236>. [p]
- D. G. Clayton. A model for association in bivariate life tables and application in epidemiological studies of familial tendency in chronic disease incidence. *Biometrika*, 65(1):141–151, 1978. URL <https://doi.org/10.2307/2335289>. [p]
- D. Cox. Regression models and life-tables. *Journal of the Royal Statistical Society: Series B*, 34(2):187–220, 1972. URL <https://www.jstor.org/stable/2985181>. [p]
- D. R. Cox and D. V. Hinkley. *Theoretical Statistics*. Chapman and Hall/CRC, 1979. URL <https://doi.org/10.1201/b14832>. [p]
- M. C. Donohue and R. Xu. **phmm**: Proportional Hazards Mixed-effects Models, 2019. URL <https://CRAN.R-project.org/package=phmm>. R package version 0.7-11. [p]
- T. Emura. **Copula.surv**: Association Analysis of Bivariate Survival Data Based on Copulas, 2018. URL <https://CRAN.R-project.org/package=Copula.surv>. R package version 1.0. [p]
- T. Emura, C. W. Lin, and W. Wang. A goodness-of-fit test for archimedean copula models in the presence of right censoring. *Computational Statistics & Data Analysis*, 54(12):3033–3043, 2010. URL <https://doi.org/10.1016/j.csda.2010.03.013>. [p]
- M. J. Frank. On the simultaneous associativity of $f(x, y)$ and $x + y - f(x, y)$. *Aequationes Mathematicae*, 19(1):194–226, 1979. URL <https://doi.org/10.1007/BF02189866>. [p]
- E. J. Gumbel. Bivariate exponential distributions. *Journal of the American Statistical Association*, 55(292):698–707, 1960. URL <https://doi.org/10.2307/2281591>. [p]
- M. Hofert, I. Kojadinovic, M. Maechler, and J. Yan. **copula**: Multivariate Dependence with Copulas, 2018. URL <https://CRAN.R-project.org/package=copula>. R package version 0.999-19. [p]
- P. Hougaard. *Analysis of Multivariate Survival Data*. Springer-Verlag, New York, 2000. URL <https://doi.org/10.1002/sim.938>. [p]
- W. J. Huster, R. Brookmeyer, and S. G. Self. Modelling paired survival data with covariates. *Biometrics*, 45(1):145–156, 1989. URL <https://doi.org/10.2307/2532041>. [p]
- H. Joe. Parametric families of multivariate distributions with given margins. *Journal of Multivariate Analysis*, 46(2):262–282, 1993. URL <https://doi.org/10.1006/jmva.1993.1061>. [p]
- H. Joe. *Multivariate Models and Multivariate Dependence Concepts*. Chapman & Hall, London, 1997. URL <https://doi.org/10.1201/9780367803896>. [p]
- H. Joe. *Dependence modeling with copulas*. CRC press, 2014. URL <https://doi.org/10.1201/b17116>. [p]
- H. Joe and T. Hu. Multivariate distributions from mixtures of max-infinitely divisible distributions. *Journal of multivariate analysis*, 57(2):240–265, 1996. URL <https://doi.org/10.1006/jmva.1996.0032>. [p]

- I. Kojadinovic and J. Yan. Modeling multivariate distributions with continuous margins using the copula R package. *Journal of Statistical Software*, 34(9):1–20, 2010. URL <https://doi.org/10.18637/jss.v034.i09>. [p]
- G. Lindfield et al. *Microcomputers in Numerical Analysis*. Halsted Press, 1989. URL <https://doi.org/10.1111/S002557930001319X>. [p]
- G. Marra and R. Radice. Bivariate copula additive models for location, scale and shape. *Computational Statistics & Data Analysis*, 112:99–113, 2017. URL <https://doi.org/10.1016/j.csda.2017.03.004>. [p]
- G. Marra and R. Radice. Copula link-based additive models for right-censored event time data. *Journal of the American Statistical Association*, pages 1–20, 2019. URL <https://doi.org/10.1080/01621459.2019.1593178>. [p]
- G. Marra and R. Radice. **GJRM**: Generalised Joint Regression Modelling, 2020. URL <https://CRAN.R-project.org/package=GJRM>. R package version 0.2-2. [p]
- G. Marra, R. Radice, T. Bärnighausen, S. N. Wood, and M. E. McGovern. A simultaneous equation approach to estimating HIV prevalence with nonignorable missing responses. *Journal of the American Statistical Association*, 112(518):484–496, 2017. URL <https://doi.org/10.1080/01621459.2016.1224713>. [p]
- G. Masarotto and C. Varin. Gaussian copula regression in R. *Journal of Statistical Software*, 77(8):1–26, 2017. URL <https://doi.org/10.18637/jss.v077.i08>. [p]
- M. Munda, F. Rotolo, and C. Legrand. **parfm**: Parametric frailty models in R. *Journal of Statistical Software*, 51(11):1–20, 2012. URL <https://doi.org/10.18637/jss.v051.i11>. [p]
- T. Nagler and T. Vatter. **gamCopula**: Generalized Additive Models for Bivariate Conditional Dependence Structures and Vine Copulas, 2020. URL <https://CRAN.R-project.org/package=gamCopula>. R package version 0.0-7. [p]
- J. Nash. *Compact Numerical Methods for Computers: Linear Algebra and Function Minimisation*. Taylor & Francis, 1990. URL <https://doi.org/10.2307/3314683>. [p]
- R. B. Nelsen. *An Introduction to Copulas*. Springer-Verlag, New York, 2006. URL <https://doi.org/10.1007/0-387-28678-0>. [p]
- D. S. Nicole Kraemer. *Bivariate Copula Based Regression Models*, 2014. URL <https://cran.r-project.org/web/packages/CopulaRegression/>. R package version 0.1-5. [p]
- D. Oakes. A model for association in bivariate survival data. *Journal of the Royal Statistical Society: Series B*, 44(3):414–422, 1982. URL <https://www.jstor.org/stable/2345500>. [p]
- L. Prenen, R. Braekers, and L. Duchateau. Extending the Archimedean copula methodology to model multivariate survival data grouped in clusters of variable size. *Journal of the Royal Statistical Society: Series B*, 79(2):483–505, 2017a. URL <https://doi.org/10.1111/rssb.12174>. [p]
- L. Prenen, R. Braekers, L. Duchateau, and E. D. Troyer. **Sunclarco**: Survival Analysis using Copulas, 2017b. URL <https://CRAN.R-project.org/package=Sunclarco>. R package version 1.0.0. [p]
- R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2019. URL <http://www.R-project.org/>. [p]
- V. Rondeau, Y. Marzroui, and J. Gonzalez. **frailtypack**: An R package for the analysis of correlated survival data with frailty models using penalized likelihood estimation or parametrical estimation. *Journal of Statistical Software*, 47(4):1–28, 2012. URL <https://doi.org/10.18637/jss.v047.i04>. [p]
- U. Schepsmeier, J. Stoeber, E. C. Brechmann, B. Graeler, T. Nagler, T. Erhardt, C. Almeida, A. Min, C. Czado, M. Hofmann, et al. **VineCopula**: Statistical Inference of Vine Copulas, 2018. URL <https://CRAN.R-project.org/package=VineCopula>. R package version 2.1.8. [p]
- J. H. Shih. A goodness-of-fit test for association in a bivariate survival model. *Biometrika*, 85(1):189–200, 1998. URL <https://doi.org/10.1093/biomet/85.1.189>. [p]

- A. Sklar. Fonctions de répartition à n dimensions et leurs marges. *Publications de l'Institut de Statistique de l'Université de Paris*, 8:229–231, 1959. URL <https://doi.org/10.12691/ijefm-3-2-3>. [p]
- J. Sun. *The Statistical Analysis of Interval-Censored Failure Time Data*. Springer Science & Business Media, 2007. URL <https://doi.org/10.1007/0-387-37119-2>. [p]
- T. Sun and Y. Ding. Copula-based semiparametric regression method for bivariate data under general interval censoring. *Biostatistics*, 2019. URL <https://doi.org/10.1093/biostatistics/kxz032>. [p]
- T. Sun, Y. Liu, R. J. Cook, W. Chen, and Y. Ding. Copula-based score test for bivariate time-to-event data, with application to a genetic study of AMD progression. *Lifetime Data Analysis*, 25(3):546–568, 2019. URL <https://doi.org/10.1007/s10985-018-09459-5>. [p]
- A. Swaroop, E. Y. Chew, G. R. Abecasis, et al. Unraveling a multifactorial late-onset disease: from genetic susceptibility to disease mechanisms for Age-related Macular Degeneration. *Annual Review of Genomics and Human Genetics*, 10:19–43, 2009. URL <https://doi.org/10.1146/annurev.genom.9.081307.164350>. [p]
- T. M. Therneau. **coxme**: Mixed Effects Cox Models, 2018a. URL <https://CRAN.R-project.org/package=coxme>. R package version 2.2-10. [p]
- T. M. Therneau. **survival**: A Package for Survival Analysis in S, 2018b. URL <https://CRAN.R-project.org/package=survival>. R package version 2.43-3. [p]
- T. Vatter and V. Chavez-Demoulin. Generalized additive models for conditional dependence structures. *Journal of Multivariate Analysis*, 141:147–167, 2015. URL <https://doi.org/10.1016/j.jmva.2015.07.003>. [p]
- A. Wang. Goodness-of-fit tests for Archimedean copula models. *Statistica Sinica*, 20:441–453, 2010. URL <https://www.jstor.org/stable/24309000>. [p]
- L. J. Wei, D. Lin, and L. Weissfeld. Regression analysis of multivariate incomplete failure time data by modeling marginal distributions. *Journal of the American Statistical Association*, 84(408):1065–1073, 1989. URL <https://doi.org/10.2307/2290084>. [p]
- J. Yan. Enjoy the joy of copulas: With a package copula. *Journal of Statistical Software*, 21(4):1–21, 2007. URL <https://doi.org/10.18637/jss.v021.i04>. [p]
- Q. Zhou, T. Hu, and J. Sun. A sieve semiparametric maximum likelihood approach for regression analysis of bivariate interval-censored failure time data. *Journal of the American Statistical Association*, 112(518):664–672, 2017. URL <https://doi.org/10.1080/01621459.2016.1158113>. [p]

Tao Sun
School of Statistics
Renmin University of China
59 Zhongguancun Street
Beijing, China
Department of Biostatistics
University of Pittsburgh
Pittsburgh, U.S.A.
ORCID: 0000-0003-4447-3005
tao.sun@pitt.edu

Ying Ding
Department of Biostatistics
University of Pittsburgh
130 De Soto Street
Pittsburgh, U.S.A.
ORCID: 0000-0003-1352-1000
yingding@pitt.edu

mistr: A Computational Framework for Mixture and Composite Distributions

by Lukas Sablica and Kurt Hornik

Abstract Finite mixtures and composite distributions allow to model the probabilistic representation of data with more generality than simple distributions and are useful to consider in a wide range of applications. The R package **mistr** provides an extensible computational framework for creating, transforming, and evaluating these models, together with multiple methods for their visualization and description. In this paper we present the main computational framework of the package and illustrate its application. In addition, we provide and show functions for data modeling using two specific composite distributions as well as a numerical example where a composite distribution is estimated to describe the log-returns of selected stocks.

Introduction

During the history of financial mathematics mankind has developed many useful theories how to describe financial markets. While the models in asset pricing are generally covered by the central limit theorem (CLT) arguments, these arguments do not cover the tail behaviour and thus are usually not appropriate in the risk management with its focus on the tails of the distribution. A simple illustration might be the log-returns distribution of the German multinational software corporation SAP. Clearly, the tails are much heavier than in the case of normal distribution with the same mean and standard deviation. This behavior can be frequently found in a number of financial assets.

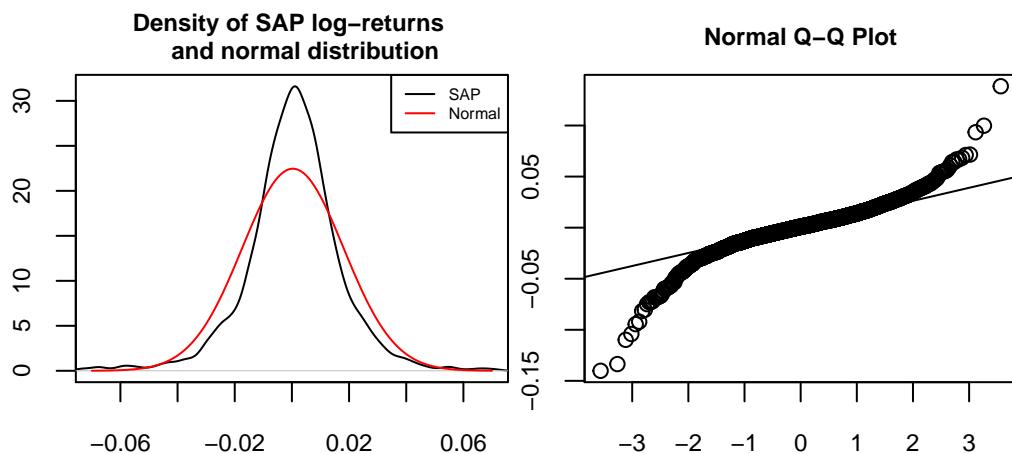


Figure 1: Left: Densities of the normal distribution and daily SAP log-returns from January 2007 to October 2017. Right: Quantile-quantile plot of the normal distribution against SAP log-returns.

A more interesting result can be seen from the quantile-quantile plot. While the normal distribution fails in the tails, it excels in the center. This suggests to use a more suitable distribution which can catch the fat tails presented above, yet follows a normal distribution in the center.

A simple answer to this idea is the concept of composite distributions (also known as spliced distributions) and mixture models, where one assumes that the distribution is a finite mixture of component distributions defined as

$$F(x) = \sum_{i=1}^n w_i F_i(x|B_i), \quad f(x) = \sum_{i=1}^n \frac{w_i}{F_i(B_i)} 1_{B_i}(x) f_i(x) \quad (1)$$

and

$$F(x) = \sum_{i=1}^n w_i F_i(x), \quad f(x) = \sum_{i=1}^n w_i f_i(x), \quad (2)$$

respectively, where w_1, w_2, \dots, w_n are positive weights that sum up to one, B_1, B_2, \dots, B_n are Borel sets giving a disjoint partition of the support, 1_{B_i} is the indicator function of a set B_i , and

F_1, F_2, \dots, F_n are the probability distributions over \mathbb{R} with $F_i(B_i) > 0$ for all $i = 1, 2, \dots, n$. Obviously, the composite models are a specific case of the mixture models, where the corresponding probability distribution functions are truncated to some disjoint support. For the presented data, a three-component composite model with the first and third component being some heavy-tailed distribution and its negative transform, respectively, is clearly something that might be appropriate. Note that this also suggests the need of a transformation framework that would be able to perform monotonic transformations on the random variables that represent these exceedance distributions. This innovation is motivated by the fact that even though most of the extreme value distributions belong to the location-scale family, without a decreasing transformation these distributions cannot be used for modeling of heavy left tails.

Moreover, composite models have gained a fair amount of attention in actuarial loss modeling. Most frequently one employs models composed of two components, where the first is based on the log-normal distribution and defined on the positive reals, and the second distribution is chosen according to the data set to model extreme measurements. Common choices for these tail-modeling distributions are, for instance, the generalized Pareto distribution or the Burr distribution. Such models have been proposed by various authors, for more details see [Nadarajah and Bakar \(2013\)](#), [Cooray and Ananda \(2005\)](#) and [Scollnik \(2007\)](#).

To offer a general framework for univariate distributions and for mixtures in general, package **mistr** is specifically designed to create such models, evaluate or even fit them. This article introduces **mistr** and illustrates with several examples how these distributions can be created and used.

Distributions in R

R currently employs the naming convention [prefix][name], where [name] corresponds to the name of the desired distribution and [prefix] is one of ‘p’, ‘d’, ‘q’ or ‘r’, indicating, respectively, the distribution, density and quantile functions, or random number generation. However, there are many of restrictions in this concept. What would be desirable is that one would be able to treat a distribution as a variable and so to be able to send it to a function or perform transformations on the random variable it represents.

Naturally, one way to do this is by using the object oriented system in R. To even improve this idea, one can use some favored dispatching mechanism, like S3 or S4, to let the computer decide how to handle the corresponding distribution correctly and which functions to use. In particular, the prefixes p, d, q, and r can still be just smartly evaluated as generic functions with appropriate methods. Moreover, with such a system we can add other useful calls and so take the distribution operations to the next level, such as monotonic transformation of a distribution. Additionally, once these objects containing all necessary information about the distributions are defined, they can be then reused for the purpose of the mixture and composite distributions.



This approach has already been used in the package **distr** ([Ruckdeschel et al., 2006](#)), which provides a conceptual treatment of distributions by means of S4 classes. A parent class *Distribution* allows to create objects and contains a slot for parameters as well as for the four methods mentioned above, `p()`, `d()`, `q()`, and `r()`. While **distr** provides several classes for distributions and finite mixtures in general, like many similar packages, to the best of our knowledge it does not contain any generating functions for the composite distributions. In particular, the only packages available for composite models are **CompLognormal** ([Nadarajah and Bakar, 2013](#)), **evmix** ([Hu and Scarrott, 2018](#)), **OpVar** ([Zou et al., 2018](#)), **ReIns** ([Reynkens and Verbeelen, 2018](#)) and **gendist** ([Bakar et al., 2016](#)), which do not offer a general framework for composite models with more than two components. Whereas **distr** provides an impressive functionality-rich and extensible framework implemented in S4, we decided after careful deliberation to realize **mistr** based on a self-contained, light-weight framework using S3. This offers to adjust the representations and settings of the distributions exactly to the required needs by either mixture or composite models.

The framework provided by package **mistr** currently supports all distributions that are included in the **stats** package and, in addition, it offers some extreme value distributions like generalized Pareto, Pareto, Frechet, and Burr. In case that the user would like to use a distribution that is not directly supported by the framework, a new distribution can be implemented in a very simple way. This procedure is documented in detail in the [Extensions vignette](#) of the package.

As was already mentioned, the objects in R that represent a distribution need to contain all the necessary information. This is, of course, for the purpose of evaluating the distribution function or quantile function at suitable points just the knowledge of the parameters. Since the framework we propose is beyond a simple distribution evaluation, the representation is a little bit more complex.

A simple random variable or a distribution is in our framework represented by the distribution family which the distribution follows, together with the proper parameters and some additional information that corresponds to the distribution. This information is either stored as a type of class or as attributes in a list. The framework then uses the S3 dispatching mechanism to use this information, and hence it works with the distribution. Such a representation allows to go beyond the simple p, d, q, and r evaluation and enables to define new and more complicated functions.

An example is the left-hand limit of the cumulative distribution function. It might not look of crucial importance to be able to evaluate $F(x-) = \mathbb{P}(X < x)$, but this function plays a huge role in the transformations and composite distributions. Of course, this function differs from the standard distribution function only if it is applied to a distribution with a positive probability mass in the point of interest. For this reason we simply characterize the (“pseudo”) support by the three parameters “from”, “to” and “by”. While the parameters “from” and “to” specify the range in which the distribution takes positive values, the parameter “by”, which is only a part of the lattice-valued distributions, describes the deterministic step in the support and for most known discrete distributions is equal to one, since they have support just on the integers. It might of course differ for some distributions, which have support only on even numbers, or some scaled distributions. It is essential that this parametrization allows to perfectly define the support of a distribution, and hence allows to do more complicated operations and calculations, e.g., the left-hand limit of the distribution function. In the case the user would like a distribution with no equally distanced outcomes, one can perform a non-linear transformation, which will be dealt with in the next chapter.

Combining distributions

Having the framework defined for the simple distributions, it is desired to combine these objects in order to define a larger class of distributions. The framework currently supports the mixture and composite models given in, respectively, (2) and (1).

Following the definition of the composite models in the univariate case, the interval representation of the truncation allows to use a sequence of breakpoints

$$-\infty = \beta_0 < \beta_1 \leq \beta_2 \leq \cdots \leq \beta_{n-1} < \beta_n = \infty$$

to fully characterize the partitions B_1, B_2, \dots, B_n . Note that if F_i is continuous, to ensure that the interval has positive probability we must take $\beta_{i-1} < \beta_i$.

This allows to define $\lambda_1 = 0$ and for all $i = 2, \dots, n$,

$$\lambda_i = \begin{cases} F_i(\beta_{i-1}) & \text{if } \beta_{i-1} \notin B_i, \\ F_i(\beta_{i-1}-) & \text{otherwise,} \end{cases}$$

where as before $F_i(\beta_{i-1}-)$ is the left-hand limit $\mathbb{P}(X_i < x)$, and for all $i = 1, 2, \dots, n-1$,

$$\rho_i = \begin{cases} F_i(\beta_i) & \text{if } \beta_i \in B_i, \\ F_i(\beta_i-) & \text{otherwise,} \end{cases}$$

with $\rho_n = 1$. Then for any $x \in B_i$

$$F_i((-\infty, x] \cap B_i) = \begin{cases} F_i(x) - F_i(\beta_{i-1}) & \text{if } \beta_{i-1} \notin B_i, \\ F_i(x) - F_i(\beta_{i-1}-) & \text{if } \beta_{i-1} \in B_i. \end{cases}$$

This means that for every $x \in B_i$ we can write the distribution as $F_i((-\infty, x] \cap B_i) = F_i(x) - \lambda_i$.

The straightforward implication of the above equations is that $\sup_{x \in B_i} F_i(x) = \rho_i$. Thus,

$$F_i(B_i) = \rho_i - \lambda_i.$$

Hence, if we define $p_i = \sum_{j:j \leq i} w_i$ the composite distribution satisfies

$$F(x) = p_{i-1} + w_i \frac{F_i(x) - \lambda_i}{F_i(B_i)} = p_{i-1} + w_i \frac{F_i(x) - \lambda_i}{\rho_i - \lambda_i}, \quad \forall x \in B_i$$

and, in addition,

$$p_{i-1} \leq F(x) \leq p_i, \quad \forall x \in B_i \quad \text{and} \quad \sup_{x \in B_i} F(x) = p_i.$$

If we take some $p \in (0, 1)$, then since $\sum_i p_i = 1$, there exists i such that $p_{i-1} < p < p_i$ or $p = p_i$. From this it follows that

$$F(x) \geq p \Leftrightarrow F_i(x) \geq \lambda_i + \frac{p - p_i}{w_i} (\rho_i - \lambda_i),$$

which implies

$$F^{-1}(p) = F_i^{-1}\left(\lambda_i + \frac{p - p_i}{w_i} (\rho_i - \lambda_i)\right), \quad \text{for } p_{i-1} < p \leq p_i.$$

Using the quantile transformation method and the above quantile function we can easily simulate from our composite distribution.

Therefore, to fully specify a composite distribution, in addition to the mixture specifications, one needs to set the values that correspond to the breakpoints, which split \mathbb{R} into disjoint partitions. Moreover, if at least one distribution is not absolutely continuous, it might be desired to specify to which adjacent interval should the breakpoint be included.

Computational framework

The objects representing a simple distribution can be created very easily. The creator functions follow a standard naming convention from R where the “dist” suffix is appended to the name of a distribution, and the parameters are entered as arguments. Thus, an object representing normal distribution with mean equal to 1 and standard deviation equal to 3 can be created as follows:

```
N <- normdist(mean = 1, sd = 3)
N

#> Distribution      Parameters
#>   Normal      mean = 1, sd = 3
```

Once the objects are created, they can be used for evaluation of various functions. The most commonly employed functions clearly will be the methods for `print()` already demonstrated, and the methods for the functions `p()`, `d()`, `q()` and `r()`. These can be easily evaluated as

```
d(N, c(1, 2, 3))
#> [1] 0.1329808 0.1257944 0.1064827

p(N, c(1, 2, 3))
#> [1] 0.5000000 0.6305587 0.7475075

q(N, c(0.1, 0.2, 0.3))
#> [1] -2.8446547 -1.5248637 -0.5732015

r(N, 3)
#> [1] 1.8144430 -2.6908086 0.2704776
```

Additional important functions provided by **mistr** are `plim()` and `qlim()`, which implement, respectively, the already mentioned left-hand limit of the cumulative distribution function $F(x-) = \mathbb{P}(X < x)$ and its pseudoinverse

$$Q(p+) = \inf \{x \in \mathbb{R} : p < \mathbb{P}(X \leq x)\}.$$

Note that the function `qlim()` plays a very important role when dealing with the transformations, and just as `plim()`, in the case of continuous distributions it simplifies to `q()`. Clearly if $-X$ has a positive probability mass at x , then

$$q = P(-X \leq x) = 1 - P(X < -x) \Rightarrow -Q_X(1 - q+) = x.$$

```
B <- binomdist(size = 12, prob = 0.3)
plim(B, c(-3, 0, 3, 12))

#> [1] 0.0000000 0.0000000 0.2528153 0.9999995

qlim(B, plim(B, c(0, 3, 7, 12)))

#> [1] 0 3 7 12
```

Adding transformation

Once the objects that represent a single distribution are created, we can use this representation to go beyond the scope of a simple distribution function evaluation. The knowledge of the distributions class and support range that is stored inside the object opens the doors for more complicated operations. One such an operation is the ability to perform monotone transformations of random variables.

The transformation framework currently allows for all standard monotone transformations like addition, subtraction, multiplication, division, logarithm, exponential and monotonic power transformations, where in case of binary operators a numeric value must be used as the second operand. Moreover, it is provided with the knowledge of invariant and direct transformations that correspond to the distributions it offers. This information is stored as a generic function that directly dispatches on the class of distribution family and not on the class *univdist* to prevent losing any information about the distribution. An example might be the exponential distribution where a multiplication with a positive scalar rather keeps the family and changes the parameters. On the other hand, a positive power transformation will directly create a Weibull distribution with appropriate parameters.

```
E <- expdist(2)

E * 2

#> Distribution      Parameters
#> Exponential      rate = 1

E^2

#> Distribution          Parameters
#>   Weibull      shape = 0.5, scale = 0.25
```

If the transformation is necessary and continuous on the support of the distribution, the transformation dispatches on the class *univdist*. For any untransformed distribution, this function will change the whole class and the class of the distribution family is removed since the random variable does not follow the distribution anymore. However, the information is stored for the case the distribution would need to untransform itself later. The function then builds an expression for the transformation and inverse transformation, along with a print expression and an expression for the derivative of the inverse transformation. Besides these list members, also a history member is stored, a list that records the information about the old transformations and becomes really handy when it comes to an inverse transformation of the previous one, or updating a transformation. A simple example of a transformation and update follows.

```
E2 <- E * -2
E3 <- E2 * 5
E3

#> Trafo Distribution Parameters
#> -10 * X Exponential    rate = 2
```

The next example uses the normal distribution that we created in the last chapter.

```
Norm_trafo <- (N - 1)^(1/3)
Norm_trafo

#> Trafo Distribution      Parameters
#> X^(1/3)      Normal mean = 0, sd = 3
```

Note that the $X - 1$ transformation is not displayed in the Trafo column as it is an invariant transformation that rather changed the parameter mean from 1 to 0.

The methods that evaluate the transformed distribution are called in the same fashion as the non-transformed distributions. Additionally, the new pseudo description of the support can be returned using *sudo_support()*, which gives two numeric values (“From” and “To”) describing the range of the support.

```
Binom_trafo <- -3 * log(B + 4)

q(Binom_trafo, c(0.05, 0.5, 0.95))
```

```
#> [1] -6.907755 -6.238325 -4.828314
plim(Binom_trafo, c(-6, -5, 0))
#> [1] 0.5074842 0.9149750 1.0000000
sudo_support(Binom_trafo)
#>      From      To
#> -8.317766 -4.158883
```

Visualization

In addition, the `plot()` and `autoplot()` generics can be called. These methods are offered for any distribution object in the `mistr` package and return the plot of PDF or PMF and CDF of a given object. The function uses the introduced `d()` and `p()` functions to evaluate the required values. While the `plot()` methods offer a plot constructed using base plotting, the `autoplot()` offers an alternative plot that is created using the `ggplot2` package (Wickham, 2016).

```
par(mai = c(0.4, 0.4, 0.2, 0.2))
plot(Norm_trafo, xlim1 = c(-2.5, 2.5), ylab1 = "", cex.axis = 0.75)
```

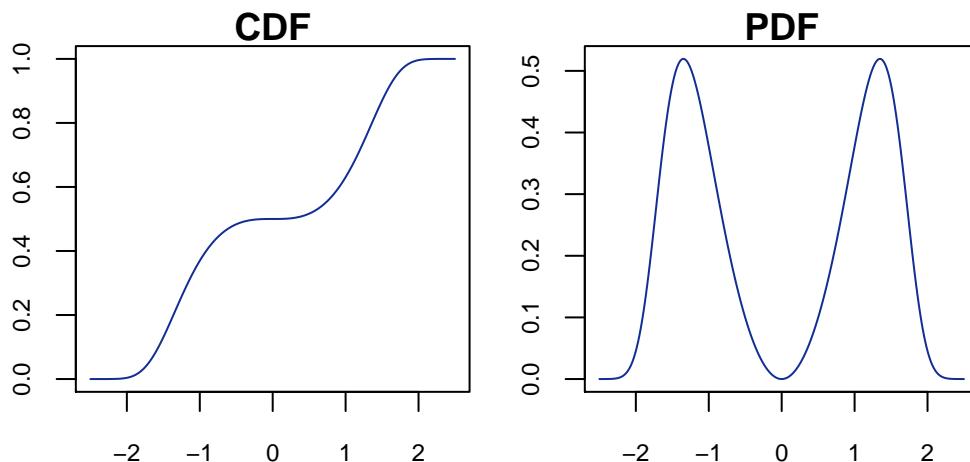


Figure 2: CDF (left) and PDF (right) plot of the `Norm_trafo` object using `plot()`.

```
library(ggplot2)
autoplot(Norm_trafo, xlim1 = c(-2.5, 2.5))
```

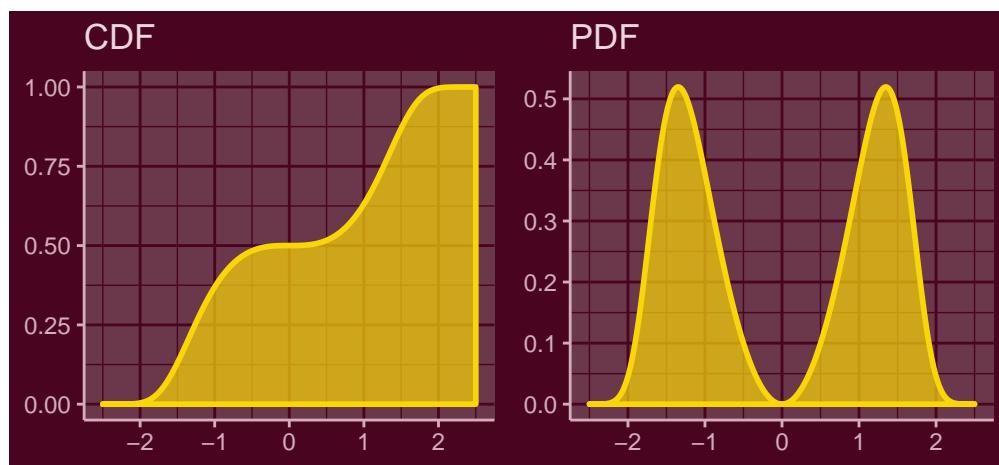


Figure 3: CDF (left) and PDF (right) plot of the `Norm_trafo` object using `autoplot()`.

Other plot functions offered for the `mistr` distribution objects are `QQplot()` and `QQplotgg()`. The methods for this functions graphically compare the empirical quantiles of two data sets, or

quantiles of two distribution objects, or quantiles of a distribution with the empirical quantiles of a sample. If quantiles of a continuous distribution are compared with a sample, a pointwise asymptotic confidence bound for this data is offered. This confidence “envelope” is based on the asymptotic results of the order statistics. For a distribution F as the number of observations n tends to infinity, the p^{th} sample quantile is asymptotically distributed as

$$X_{([np])} \sim AN \left(F^{-1}(p), \frac{p(1-p)}{n [f(F^{-1}(p))]^2} \right),$$

where $f(x)$ and $F^{-1}(p)$ are the density function and quantile function associated with $F(x)$, respectively. More details can be found on the [order statistics Wikipedia page](#) or in [Wilks \(1948\)](#), for example. For alternative bounds see [Almeida et al. \(2018\)](#).

```
QQplotgg(Norm_trafo, r(Norm_trafo, 1000), conf = 0.99, ylab = NULL, xlab = NULL)
```

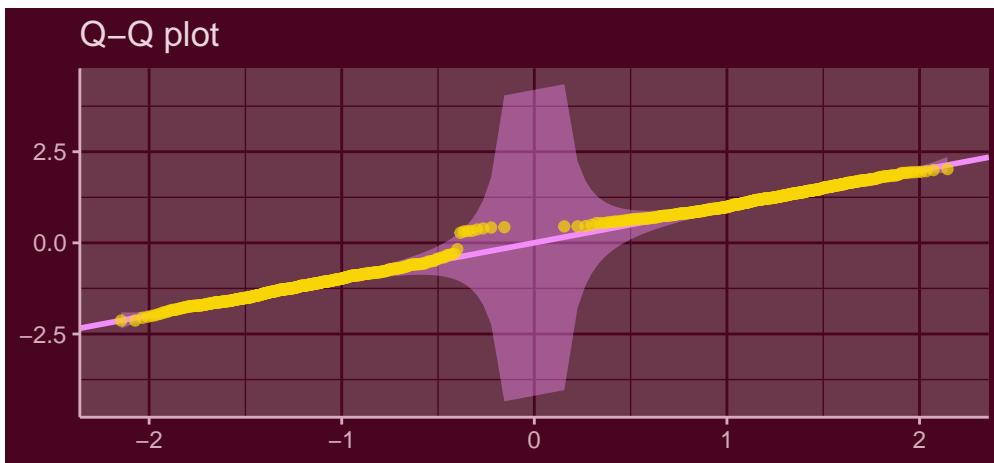


Figure 4: Q-Q plot of the `Norm_trafo` object against normally distributed sample with 99% confidence bound.

Combining objects

Mixtures

Mixture distributions are fully specified by the components $F_i(x)$ (i.e., the distributions) and by the weights w_i that correspond to these components. Function `mixdist()` allows to create mixtures by specifying these characterizations. This can be done in two ways. First, the user may specify the distribution names (names from `[prefix][name]` functions), the list of appropriate parameters of these distributions, and a sequence of weights. An example of such a call follows.

```
mixdist(c("norm", "unif"), list(c(2, 2), c(1, 5)), weights = c(0.5, 0.5))

#> Mixture distribution with:
#>
#>   Distribution      Parameters      Weight
#> 1     Normal      mean = 2, sd = 2        0.5
#> 2     Uniform    min = 1, max = 5        0.5
```

Another way is to use the objects that have already been defined. Since the parameters are already stored inside the object, all the function requires are the objects and the weights. This also allows to use transformed distributions from the last chapter or more complicated objects, which will be presented later. This means that the transformed normal and binomial distributions together with an exponential distribution can be reused for mixture distribution as:

```
M <- mixdist(Norm_trafo, Binom_trafo, expdist(0.5), weights = c(0.4, 0.2, 0.4))
```

The information about the mixture can be accessed via multiple S3 methods. The components can be extracted using square brackets `[]`, the weights can be obtained using `weights()`, and just as with standard distributions, the parameters are obtainable using `parameters()`.

Other interesting methods are those for `q` and `qlim` and `mixdist` objects. While finding the CDF and PDF of the mixture model is straightforward, an explicit general expression for quantile function of the mixture model is not available. However, it can be found numerically as a solution of a unit-root problem:

$$\sum_{i=1}^n w_i F_i(Q(p)) - p = 0.$$

What is more, one can show that the quantile of a mixture distribution $Q(p)$ can always be found within the range of its components quantiles, and hence

$$\min_{i \in \{1, \dots, n\}} Q_i(p) \leq Q(p) \leq \max_{i \in \{1, \dots, n\}} Q_i(p),$$

where $Q_i(\cdot)$ is the quantile function of the i -th component. This specifies the needed interval for the root finder that will then iteratively find the solution. Additionally, further problems are solved to return the correct values. To show how this algorithm works we perform a simple test and create a fully discrete mixture for which a decreasing transformation is applied. As the following plot reveals, all the values are calculated correctly.

```
DM <- mixdist(3 * binomdist(12, 0.4), -2*poisdist(2) + 12, weights=c(0.5, 0.5))
y <- c(0.05, 0.4, p(-DM, c(-5, -10, -15)), 0.95)
x <- q(-DM, y)
autoplot(-DM, which = "cdf", only_mix = TRUE, xlim1 = c(-37, 0)) +
  annotate("point", x, y, col = "white")
```

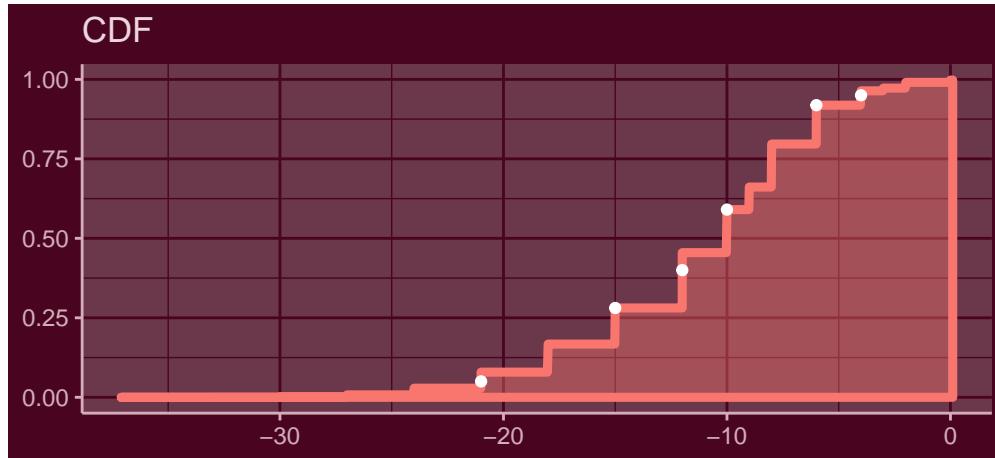


Figure 5: CDF plot of the `-DM` object with calculated quantiles annotated with white dots.

Finally, since inverse transform sampling is not efficient for mixture distributions, it can be replaced by first sampling according to the weights w_i and then, conditionally on that, by drawing from the selected component, similarly as in the `actuar` package (Dutang et al., 2008). This approach is implemented in the corresponding method of the `r()` function. This allows to draw from a mixture much faster than the inverse quantile transform method and can also be reused later for composite distributions. Besides the quantile function and other main functions for evaluation, one can call other generic functions that are designed for the class `mixdist`, e.g. `sudo_support()`.

```
sudo_support(M)

#> From    To
#> -Inf   Inf
```

Since the mixture models are in fact distributions, one can perform transformations of mixture random variables as well. It is easy to show that a transformation of a mixture random variable is the same as if we applied the same transformation to all of its components. In addition, since the support of the components is a subset of the mixture's support, if the framework allows to transform the mixture, then it does the components as well. Now using the mixture we created, we can perform a decreasing non-linear transformation. An example of `r()` and `autoplot()` follows.

```
M_trans <- -2 * (M)^(1/3)
r(M_trans, 4)
```

```
#> [1] 1.757693 2.247355 2.387137 -1.986538
autoplots(M_trans)
```

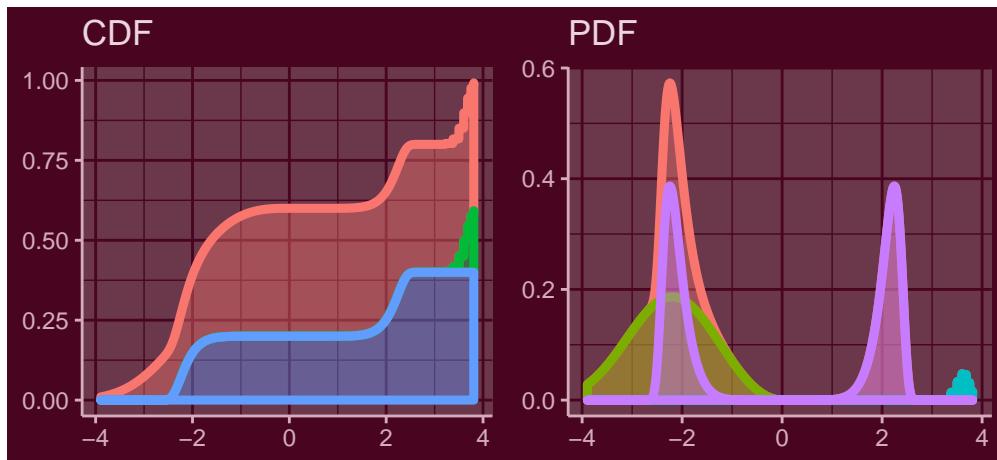


Figure 6: CDF (left) and PDF (right) plot of the `M_trans` object.

Composite distributions

Function `compdist()` creates composite distributions, which like mixture distributions can be done in two ways. Either one can directly use the objects or let the function create these objects by specifying the sequence of names and a list of parameters. In the following example, we will directly proceed with the first method where we define some objects inside the `compdist()` call to create a composite distribution. Besides these objects one needs to set the sequences of weights and breakpoints. Additionally, one may determine for each breakpoint to which partition should the breakpoint be included. This can be set by the argument `break.spec` with values ‘R’ or ‘L’, where ‘R’ and ‘L’ stand for right (i.e., include breakpoint to the interval on the right of the breakpoint) and left (i.e., include to the interval on the left), respectively. If this argument is not stated, the algorithm will by default set all intervals to be left-closed, i.e., right-open. This can be nicely seen from the following example where a linearly transformed Pareto distribution and a geometric distribution are combined with a normal distribution into a composite model.

```
C <- compdist(-paretodist(1, 1), normdist(0, 2), geomdist(0.3) + 2,
               weights = c(0.15, 0.7, 0.15), breakpoints = c(-3, 3),
               break.spec = c("L", "R"))
C
#> Composite distribution with:
#>
#>   Trafo Distribution      Parameters Weight Truncation
#> 1 -X      Pareto      scale = 1, shape = 1  0.15  (-Inf,-3]
#> 2 none    Normal      mean = 0, sd = 2  0.70  (-3,3)
#> 3 X + 2  Geometric    prob = 0.3  0.15  [3,Inf)
```

The argument `break.spec` is set to (“L”, “R”), and thus the breakpoint -3 belongs to the first partition while the second breakpoint is affiliated to the partition on the right. This can be observed from the print of the distribution, more precisely from the Truncation column, where the parentheses are printed according to this argument.

The package also permits to use the same breakpoint twice. This possibility allows to define a partition on a singleton, and hence to create a mass of probability. If this feature is used, the `break.spec` needs to be specified with “R” and “L”, for the first and second identical breakpoint, respectively, or not set at all. If the `break.spec` is not used, the source code will change `break.spec` such that this single point with probability mass is a closed set. This feature can become particularly useful when the user wants to create a distribution that is, for example, absolutely continuous on both the negative and positive reals and has positive mass at zero.

```
C2 <- compdist(-expdist(2), poisdist(), expdist(2),
                weights = c(0.25, 0.5, 0.25), breakpoints = c(0, 0))
C2
```

```
#> Composite distribution with:
#>
#> Trafo Distribution Parameters Weight Truncation
#> 1 -X Exponential rate = 2 0.25 (-Inf,0)
#> 2 none Poisson lambda = 1 0.50 [0,0]
#> 3 none Exponential rate = 2 0.25 (0,Inf)
```

Note that the distribution assigned to this singleton has to be a discrete distribution with support on that point, otherwise the interval will have zero probability.

As for any distribution, the framework offers many methods that can be used to obtain additional information or evaluate the composite distribution. One can extract the parameters, weights, or the support in the same manner as with mixture distributions. In addition, calling `breakpoints()` extracts the splicing points. Finally, there are methods for `plot()` and `autoplot()` where the components are shown by default, which again can be turned off using the `only_mix = TRUE` argument.

```
par(mai = c(0.4, 0.4, 0.2, 0.2))
plot(C, xlim1 = c(-15, 15), ylab1 = "", cex.axis = 0.75, mtext_cex = 0.75)
```

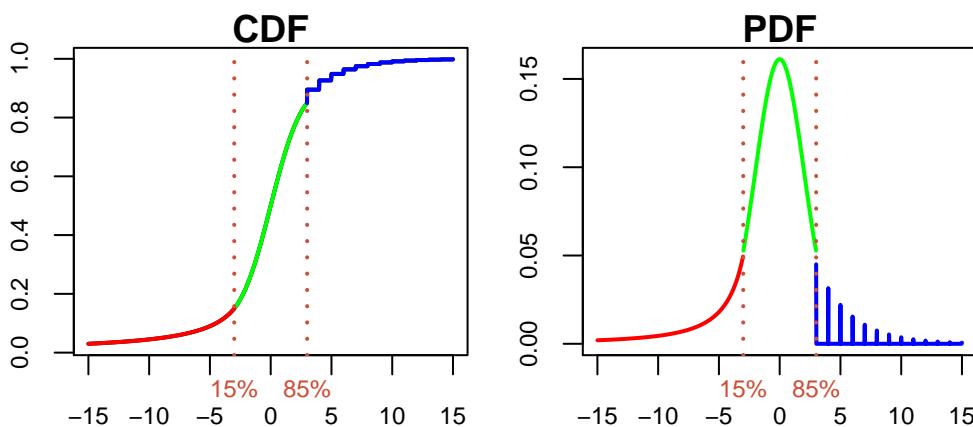


Figure 7: CDF (left) and PDF (right) plot of the `C` object using `plot()`.

```
autoplot(C2, text_ylim = 0.01)
```

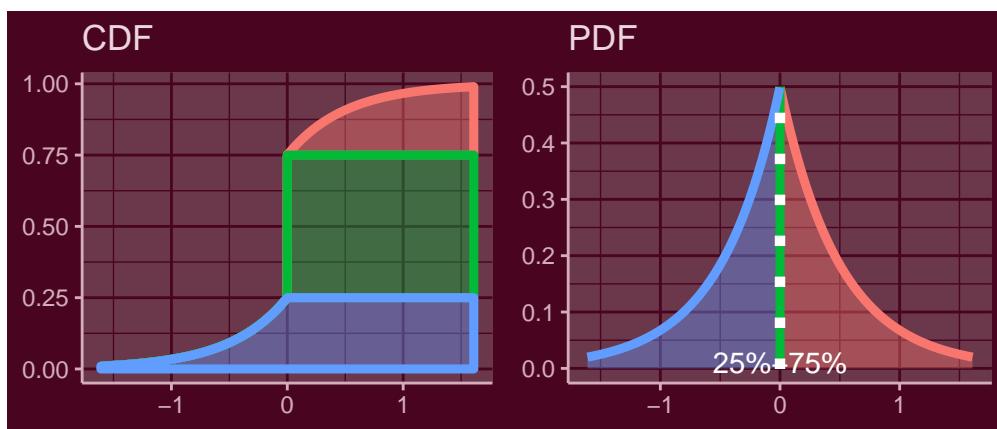


Figure 8: CDF (left) and PDF (right) plot of the `C2` object using `autoplot()`.

Analogously to the mixture distributions, the framework also supports the transformations of composite random variables. Thus, using the composite distribution we defined, we propose an example of a linear transformation.

```
C_trans <- -0.5 * (C + 7)
```

Even with such a distribution, the user still can evaluate all functions “of interest”. To support this, an example follows where the function `q()` and `r()` are used, and the functions `p()` and `d()` are represented graphically using the `autoplot()` method.

```

q(C_trans, c(0.075, 0.5, 0.7, 0.9))
#> [1] -5.500000 -3.500000 -2.833235 -1.250000
r(C_trans, 4)
#> [1] -4.635072 -3.161199 -5.500000 -4.817573
autoplot(C_trans, xlim1 = c(-10,5))

```

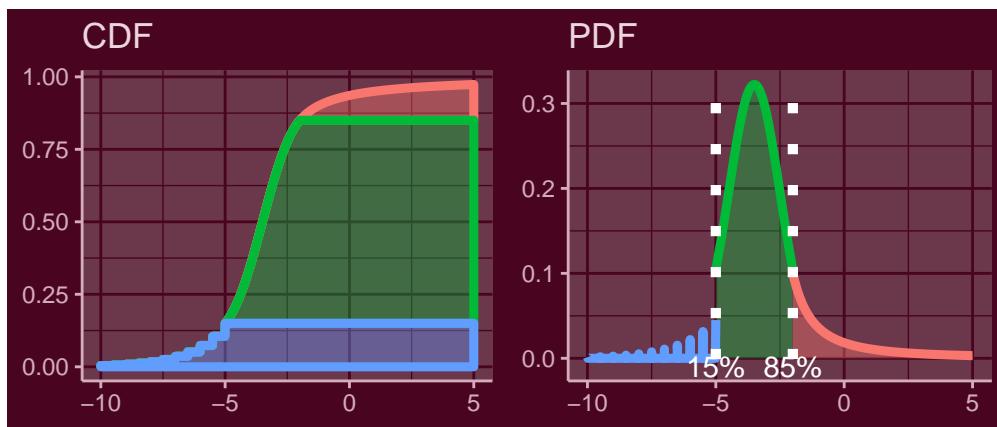


Figure 9: CDF (left) and PDF (right) plot of the `C_trans` object using `autoplot()`.

Combining mixture and composite distributions

A significant advantage of object oriented programming is that the dispatching mechanism automatically knows how to treat a given object. This allows to combine mixture and composite models into more complicated mixtures and composite distributions. Therefore, we can take the transformed mixture and the transformed composite distribution that we created in the last chapter to compose a composite distribution with these distributions as components. What is more, we can perform further transformations of such a distribution.

```

C3 <- compdist(M_trans - 3, C_trans, weights = c(0.5, 0.5), breakpoints = -4.5)
C3_trans <- -2 * C3 + 2

```

Thus, the object `C3_trans` is a transformed composite distribution that contains a transformed mixture and a transformed composite distribution, from which both additionally contain many transformed and untransformed distributions. Even in such complex models, the user may evaluate the most complicated functions like `p1m()` and `q1m()`. The functions `d()` and `p()` can be again best represented graphically, where both distributions can easily be recognized from previous chapters.

```
autoplot(C3_trans, xlim1 = c(0,20), text_ylim = 0.01, grey = TRUE)
```

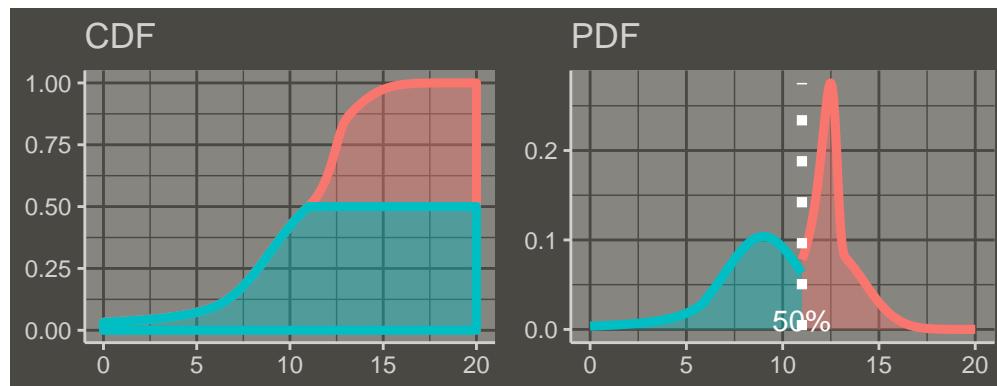


Figure 10: CDF (left) and PDF (right) plot of the `C3_trans` object.

```

plim(C3_trans, c(6, 10, 12))
#> [1] 0.09667553 0.42195189 0.62458021
qlim(C3_trans, c(0.3, 0.5, 0.7))
#> [1] 8.785363 11.000000 12.327907

```

As the `print()` output for such a hierarchical distribution does not contain lot of information, one can alternatively use `summary()` for which the package provides methods, which are particularly useful for the hierarchical distributions. The printed result of this call consists of all the necessary information, and much more as well. Nevertheless, since the result of `summary()` on the `C3_trans` object is two pages long, the demonstration is left to the reader.

To finish this chapter and to show that the user may go even further, we present an example where we combine the last object with another distribution from this chapter into a mixture distribution. The distribution is directly plotted using `autoplot()`.

```
autoplot(mixdist( C3_trans, C2 + 5, weights = c(0.7, 0.3)), xlim1 = c(0, 15))
```

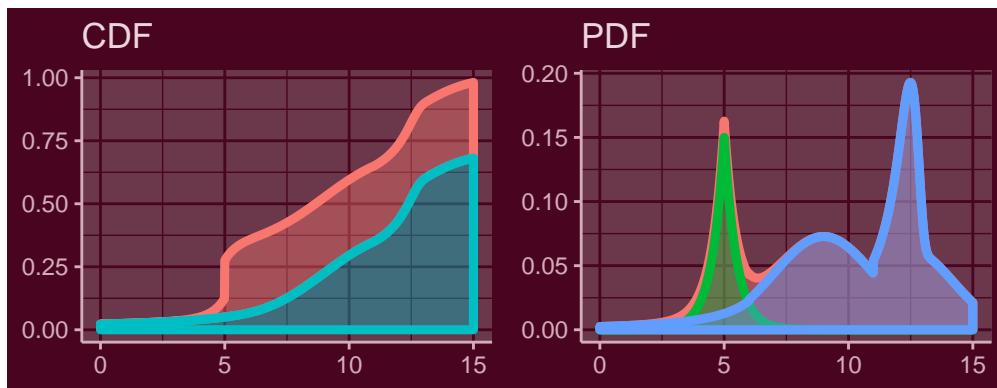


Figure 11: CDF (left) and PDF (right) plot of the mixture distribution with components `C3_trans` and `C2 + 5`.

Data modeling

While the previous chapters were aimed at showing the “muscles” (i.e., generality) of the framework, in this last chapter we will focus on examples using real data. In particular, we will present a simple fitting for two specific composite distributions.

As motivated in the introduction, the models in financial and actuarial mathematics suggest employing distributions which can capture the wide variety of behaviors in tails while still following the normal distribution in the center. In the two-tailed case, this suggests to use a three components composite distribution, where the first and third component will be used to model the extreme cases, i.e., the tails, and the second component will try to catch the center of the empirical distribution.

The first model offered by **mistr** is the Pareto-Normal-Pareto (PNP) model. This means that a $-X$ transformation of a Pareto random variable will be used for the left tail, normal distribution for the center and again Pareto for the right tail. From this it follows that the PDF of the model can be written as:

$$f(x) = \begin{cases} w_1 \frac{f_{-P}(x)}{F_{-P}(\beta_1)} & \text{if } -\infty < x < \beta_1, \\ w_2 \frac{f_N(x)}{F_N(\beta_2) - F_N(\beta_1)} & \text{if } \beta_1 \leq x < \beta_2, \\ w_3 \frac{f_P(x)}{1 - F_P(\beta_2)} & \text{if } \beta_2 \leq x < \infty, \end{cases}$$

where $f_P(x) = f_{-P}(-x)$ and $F_P(x) = 1 - (K/x)^\alpha$ are the density and distribution function of a Pareto distribution with $F_{-P}(x) = 1 - F_P(-x)$. $f_N(x)$ and $F_N(x)$ are the PDF and CDF of the normal distribution, respectively.

If we follow the properties of a Pareto distribution, the conditional probability distribution of a Pareto-distributed random variable, given that the event is greater than or equal to $\gamma > K$, is again a Pareto distribution with parameters γ and α . This means that the conditional distribution $f_P(x|K, \alpha)/(1 - F_P(\beta_2|K, \alpha)) = f_P(x|\beta_2, \alpha)$ if $\beta_2 > K$. On the other hand, if $\beta_2 < K$ the

distribution cannot be continuous as the support of a Pareto distribution starts at K . The same can be shown for the transformed distribution and hence we can rewrite the PDF as

$$f(x) = \begin{cases} w_1 f_{-P}(x|-\beta_1, \alpha_1) & \text{if } -\infty < x < \beta_1, \\ w_2 \frac{f_N(x|\mu, \sigma)}{F_N(\beta_2|\mu, \sigma) - F_N(\beta_1|\mu, \sigma)} & \text{if } \beta_1 \leq x < \beta_2, \\ w_3 f_P(x|\beta_2, \alpha_2) & \text{if } \beta_2 \leq x < \infty, \end{cases} \quad \text{where } \begin{cases} \beta_1 < 0 < \beta_2, \\ \alpha_1, \alpha_2 > 0. \end{cases}$$

The condition $\beta_1 < 0 < \beta_2$ follows from the fact that the scale parameter has to be positive. Thus, such a model can be fully used only with demeaned data sample or with data with a mean close to zero. This is of course not a problem for stock returns, which are the aim of this chapter. What is more, one can show that the density is continuous if for the shape parameters it holds that

$$\alpha_1 = -\beta_1 \frac{w_2 f_N(\beta_1|\mu, \sigma)}{w_1 (F_N(\beta_2|\mu, \sigma) - F_N(\beta_1|\mu, \sigma))},$$

$$\alpha_2 = \beta_2 \frac{w_2 f_N(\beta_2|\mu, \sigma)}{w_3 (F_N(\beta_2|\mu, \sigma) - F_N(\beta_1|\mu, \sigma))}.$$

This condition is not only natural when modelling the real data, it additionally solves the possible instability effects at estimation once observations come to lie close to breakpoints. Due to the fact that a composite distribution can be represented as a mixture of truncated distributions that are truncated to a disjoint support, the weight of each component can be estimated as the proportion of points that correspond to each of the truncated regions. Obviously, this condition ensures that the empirical and estimated CDF match on each of the breakpoints. Thus, conditionally on the fact that the breakpoints are known, similarly as in [Reynkens et al. \(2017\)](#), the weights can be computed as

$$w_1 = \frac{\sum_{i=1}^n 1_{\{x_i < \beta_1\}}}{n}, \quad w_2 = \frac{\sum_{i=1}^n 1_{\{\beta_1 \leq x_i < \beta_2\}}}{n}, \quad w_3 = \frac{\sum_{i=1}^n 1_{\{\beta_2 \leq x_i\}}}{n},$$

where $1_{\{\cdot\}}$ is the indicator function and x_i is the i-th data value. These conditions decrease the number of parameters from 11 to 4 and imply the density function of the form:

$$f(x|\beta_1, \beta_2, \mu, \sigma).$$

This model is offered by the function `PNP_fit()` which takes the data and a named vector of starting values with names `break1`, `break2`, `mean`, and `sd` and returns a list of class `comp_fit`. Other arguments are passed to the optimizer which then maximizes the likelihood of the above specified model. The optimization is handled by the `mle2()` function from `bbmle` package ([Ben Bolker and R Development Core Team, 2017](#)). To demonstrate this, we will take the same data we used in the introduction to fit a PNP model with the default starting values.

```
PNP_model <- PNP_fit(stocks$SAP)

PNP_model

#> Fitted composite Pareto-Normal-Pareto distribution:
#>
#> Breakpoints: -0.019304 0.020518
#> Weights: 0.092443 0.82135 0.086207
#>
#> Parameters:
#>   scale1    shape1      mean       sd   scale2    shape2
#> 0.019304 1.773598 0.000961 0.012950 0.020518 2.198590
#>
#> Log-likelihood: 7400.117, Average log-likelihood: 2.7146
```

If the fitted object is printed, the function prints all the parameters together with the log-likelihood that was achieved by the optimization. In addition, the average log-likelihood is printed, which is just the log-likelihood divided by the size of the data set. The user can extract parameters using `parameters()`, weights using `weights()`, and breakpoints using `breakpoints()`. The `distribution()` function can be used to extract the distribution with fitted parameters that can be used for evaluation.

Finally, again `plot()` and `autoplot()` methods are offered, which give a Q-Q plot of the fitted distribution and the data, and the CDF and PDF plot of the fitted distribution, which overlap with the empirical CDF and PDF of the data set. Again, the `which` argument can extract the proposed plots separately (i.e., `which = "pdf"`). Other arguments are passed to the the plot calls.

```
plot(PNP_model, ylab1 = "", ylab2 = "")
```

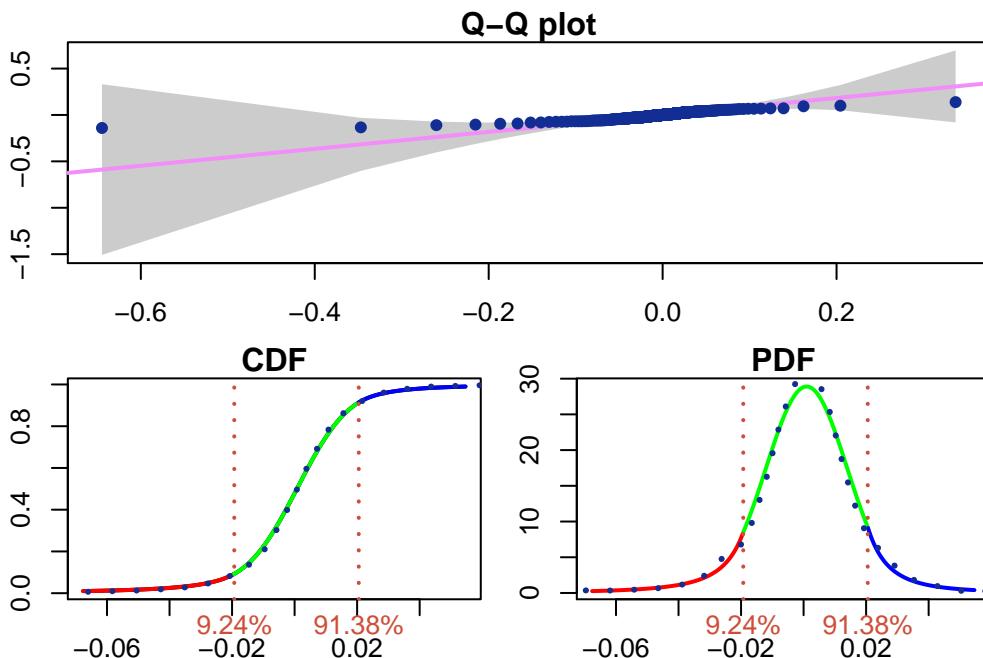


Figure 12: Q-Q (up), CDF (bottom left) and PDF (bottom right) plot of the fitted PNP model overlapped with empirical results indicated by blue dots using `plot()`.

The plots indicate an overall nice fit where all quantiles are in the confidence bound.

The second model offered is a similar distribution to the previous one, except we will replace the Pareto distributions by the generalized Pareto distributions (GPD)

$$F_{GPD}(x) = \begin{cases} 1 - \left(1 + \xi \frac{x-\theta}{\gamma}\right)^{-1/\xi} & \text{if } \xi \neq 0, \\ 1 - \exp\left(-\frac{x-\theta}{\gamma}\right) & \text{if } \xi = 0. \end{cases}$$

This means that the PDF of this model can be written as:

$$f(x) = \begin{cases} w_1 \frac{f_{-GPD}(x)}{F_{-GPD}(\beta_1)} & \text{if } -\infty < x < \beta_1, \\ w_2 \frac{f_N(x)}{F_N(\beta_2) - F_N(\beta_1)} & \text{if } \beta_1 \leq x < \beta_2, \\ w_3 \frac{f_{GPD}(x)}{1 - F_{GPD}(\beta_2)} & \text{if } \beta_2 \leq x < \infty. \end{cases}$$

The same way as in the PNP model, the scale parameters can be eliminated by the continuity conditions, weights by the above mentioned condition and in addition, under current settings and the continuity conditions, the value of the conditional GPD distribution depends on the location parameter only through the conditions $-\beta_1 \geq \theta_1$ and $\beta_2 \geq \theta_2$. This suggests to choose without any loss in the model $-\beta_1 = \theta_1$ and $\beta_2 = \theta_2$. This PDF is then fully characterized by

$$f(x|\beta_1, \beta_2, \mu, \sigma, \xi_1, \xi_2),$$

where the only restriction on the parameters is $-\infty < \beta_1 < \beta_2 < \infty$.

These conditions decrease the number of parameters from 13 to 6. What is more, the function `GNG_fit()` contains the argument `break_fix`, which fixes the breakpoints from the vector of starting values, and so decreases the number of parameters to 4 if `TRUE` is assigned. In this case, the breakpoints are fixed and weights are computed before the optimization. The function `GNG_fit()` takes the data, the named vector of starting values with names `break1`, `break2`, `mean`, `sd`, `shape1` and `shape2`, the `break_fix` argument and the argument `midd`, which is by default set to be equal to the mean of the data. The `midd` value is used to split \mathbb{R} into two sub-intervals and then the first breakpoint is optimized on the left of the `midd` value and the second breakpoint on the right.

The function returns a list of class `comp_fit`. The results can be then extracted, printed or visualized in the same way as the results of `PNP_fit()`.

```
GNG_model <- GNG_fit(stocks$SAP)
```

```
GNG_model

#> Fitted composite GPD-Normal-GPD distribution:
#>
#> Breakpoints: -0.019414 0.019353
#> Weights: 0.091343 0.812546 0.096112
#>
#> Parameters:
#>   loc1    scale1    shape1      mean       sd     loc2    scale2    shape2
#> 0.019414 0.013439 0.150141 0.000907 0.011696 0.019353 0.010842 0.096832
#>
#> Log-likelihood: 7423.245, Average log-likelihood: 2.7231

autoplot(GNG_model)
```

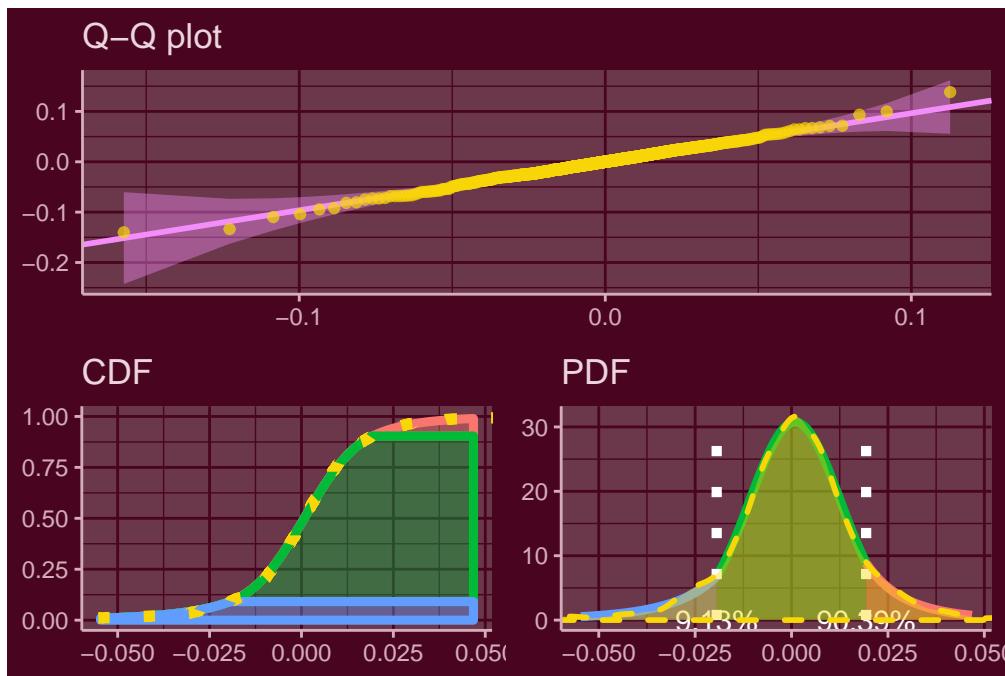


Figure 13: Q-Q (up), CDF (bottom left) and PDF (bottom right) plot of the fitted GNG model overlapped with empirical results indicated by yellow dots and dashes using `autoplot()`.

The log-likelihood has increased to 7423.2 with the average of 2.723 per data-point. In this model, the generalized Pareto distribution explains the first 9.1% from the left tail and the last 9.6% from the right tail. Since the GPD generalizes the Pareto distribution, the higher likelihood is a reasonable result. Moreover, the QQ-plot suggests an almost perfect fit.

The result of these estimations is a proper continuous parametric set-up that describes the distribution of the data. What is more, the distribution has been fitted as a whole with respect to the continuity conditions. This means that the tails take into account the whole distribution, which allows to calculate the risk measures with an even higher precision as when only the tails are modeled.

Risk measures

Package **mistr** provides a function `risk()` which can be used for rapid calculations of point estimates of prescribed quantiles, expected shortfalls and expectiles (in the following table denoted as VaR, ES and Exp, respectively). For more details on these measures see [McNeil et al. \(2015\)](#). As input parameters this function needs the output of the function `PNP_fit()` or `GNG_fit()` and a vector of the desired levels. As an example we illustrate this function on our fitted object.

```
risk(GNG_model, c(0.02, 0.05, 0.07, 0.1, 0.2, 0.3))
```

```
#>   level      VaR       ES       Exp
#> 1  0.02 0.042341368 0.06220519 0.032240507
#> 2  0.05 0.027889909 0.04520065 0.021949976
#> 3  0.07 0.023062791 0.03952074 0.018526202
#> 4  0.10 0.018245509 0.03380624 0.015091608
#> 5  0.20 0.010642738 0.02386181 0.008851640
#> 6  0.30 0.006167236 0.01867190 0.005168659
```

These results can be also visualized if arguments `plot` or `ggplot` are set to `TRUE`.

Summary

We introduced a new extensible framework for mixture and composite distributions in R via the `mistr` package. It offers creation and manipulation of simple distributions which can be combined into more complicated distributions to offer more generality in the current data modeling. Furthermore, the framework provides multiple methods specifically designed to describe and visualize the distributions or functions capable of fitting the two pre-defined composite distributions introduced in the last chapter.

The package is additionally equipped with the possibility to extend the current list of known distributions and monotone transformations by letting the user add these in a very simple way. This procedure is documented in detail in the [Extensions vignette](#). Finally, we will keep adding multiple extensions and distributions to extend the generality even more.

Bibliography

- A. Almeida, A. Loy, and H. Hofmann. ggplot2 Compatible Quantile-Quantile Plots in R. *The R Journal*, 10(2):248–261, 2018. URL <https://doi.org/10.32614/RJ-2018-051>. [p]
- S. Bakar, S. Nadarajah, Z. Kamarul Adzhar, and I. Mohamed. Gendist: An r package for generated probability distribution models. *P L o S One*, 11(6), 6 2016. ISSN 1932-6203. URL <https://doi.org/10.1371/journal.pone.0156537>. [p]
- Ben Bolker and R Development Core Team. *bbmle: Tools for General Maximum Likelihood Estimation*, 2017. URL <https://CRAN.R-project.org/package=bbmle>. R package version 1.0.20. [p]
- K. Cooray and M. M. Ananda. Modeling actuarial data with a composite lognormal-pareto model. *Scandinavian Actuarial Journal*, 2005(5):321–334, 2005. URL <https://doi.org/10.1080/03461230510009763>. [p]
- C. Dutang, V. Goulet, and M. Pigeon. actuuar: An r package for actuarial science. *Journal of Statistical Software, Articles*, 25(7):1–37, 2008. ISSN 1548-7660. URL <https://doi.org/10.18637/jss.v025.i07>. [p]
- Y. Hu and C. Scarrott. evmix: An r package for extreme value mixture modeling, threshold estimation and boundary corrected kernel density estimation. *Journal of Statistical Software, Articles*, 84(5):1–27, 2018. ISSN 1548-7660. URL <https://doi.org/10.18637/jss.v084.i05>. [p]
- A. J. McNeil, R. Frey, and P. Embrechts. *Quantitative Risk Management: Concepts, Techniques and Tools*. Princeton University Press, Princeton, NJ, USA, 2015. ISBN 0691166277, 9780691166278. [p]
- S. Nadarajah and S. A. A. Bakar. CompLognormal: An R Package for Composite Lognormal Distributions. *The R Journal*, 5(2):97–103, 2013. URL <https://doi.org/10.32614/RJ-2013-030>. [p]
- T. Reynkens and R. Verbelen. ReIns: Functions from "Reinsurance: Actuarial and Statistical Aspects", 2018. URL <https://CRAN.R-project.org/package=ReIns>. R package version 1.0.8. [p]
- T. Reynkens, R. Verbelen, J. Beirlant, and K. Antonio. Modelling censored losses using splicing: A global fit strategy with mixed erlang and extreme value distributions. *Insurance: Mathematics and Economics*, 77:65–77, 2017. URL <https://doi.org/10.1016/j.insmatheco.2017.08.005>. [p]

- P. Ruckdeschel, M. Kohl, T. Stabla, and F. Camphausen. S4 classes for distributions. *R News*, 6(2):2–6, May 2006. [p]
- D. P. M. Scollnik. On composite lognormal-pareto models. *Scandinavian Actuarial Journal*, 2007(1):20–33, 2007. URL <https://doi.org/10.1080/03461230601110447>. [p]
- H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. ISBN 978-3-319-24277-4. URL <https://ggplot2.tidyverse.org>. [p]
- S. S. Wilks. Order statistics. *Bulletin of the American Mathematical Society*, 54(1):6–50, 1948. [p]
- C. Zou, M. Pfeuffer, M. Fischer, K. Dehler, N. Derfuss, B. Graswald, L. Moestel, J. Wang, and L. Wicht. *OpVaR: Statistical Methods for Modeling Operational Risk*, 2018. URL <https://CRAN.R-project.org/package=OpVaR>. R package version 1.0.5. [p]

Lukas Sablica

Institute for Statistics and Mathematics
Vienna University of Economics and Business
Austria
<https://www.wu.ac.at/en/statmath>
ORCID: 0000-0001-9166-4563
lsablica@wu.ac.at

Kurt Hornik

Institute for Statistics and Mathematics
Vienna University of Economics and Business
Austria
<https://www.wu.ac.at/en/statmath>
ORCID: 0000-0003-4198-9911
Kurt.Hornik@wu.ac.at

difNLR: Generalized Logistic Regression Models for DIF and DDF Detection

by Adéla Hladká, Patrícia Martinková

Abstract Differential item functioning (DIF) and differential distractor functioning (DDF) are important topics in psychometrics, pointing to potential unfairness in items with respect to minorities or different social groups. Various methods have been proposed to detect these issues. The **difNLR** R package extends DIF methods currently provided in other packages by offering approaches based on generalized logistic regression models that account for possible guessing or inattention, and by providing methods to detect DIF and DDF among ordinal and nominal data. In the current paper, we describe implementation of the main functions of the **difNLR** package, from data generation, through the model fitting and hypothesis testing, to graphical representation of the results. Finally, we provide a real data example to bring the concepts together.

Introduction

Differential item functioning (DIF) is a phenomenon studied in the field of psychometrics which occurs when two subjects with the same ability or knowledge but from different social groups have a different probability of answering a specific test item correctly. DIF may signal bias and therefore potential unfairness of the measurement, so, DIF analysis should be used routinely in development and validation of educational and psychological tests (Martinková et al., 2017).

In recent decades, various methods for DIF detection have been proposed by many different authors and examples of their (non-exhaustive) overviews can be found in Penfield and Camilli (2006) or Magis et al. (2010). Generally, DIF detection methods can be classified into two groups – those based on item response theory (IRT) models and those based on other approaches (referenced here as non-IRT). IRT models assume that ability is latent and needs to be estimated, whereas non-IRT methods use the total test score (or its standardization) for an approximation of ability level. IRT models can be found more conceptually and computationally challenging, while non-IRT methods are in general easier to implement and interpret.

Another distinction of DIF detection methods can be made with respect to the nature of the DIF that can be detected. Some methods, such as the Mantel-Haenszel test (Mantel and Haenszel, 1959) or IRT methods using the Rasch model (Rasch, 1993), can only detect so-called uniform DIF, a situation whereby an item is consistently advantageous to one group across all levels of ability. Other methods, for instance logistic regression (Swaminathan and Rogers, 1990) or methods using a 2 parameter logistic (2PL) IRT model, can also detect a non-uniform DIF, a situation when an item is advantageous to one group for some parts of ability levels while for other parts of ability levels the other group is in a more favourable position. DIF can be described as a difference in item parameters between the two groups. Uniform DIF can be then viewed as the difference in item difficulties (location of the inflection point in the item characteristic curve) whereas non-uniform DIF also comprises a difference in item discrimination (slope at the inflection point in the item characteristic curve). Most methods for DIF detection, with the important exception of methods based on 3PL and 4PL IRT models, are limited to testing differences in difficulty and discrimination and do not account for the possibility that an item can be guessed without necessary knowledge or incorrectly answered due to inattention, let alone the possibility to test for differences in related parameters.

There are several packages on the Comprehensive R Archive Network (CRAN, <https://CRAN.R-project.org/>) which implement various methods for DIF detection. The **difR** package (Magis et al., 2010) includes a number of DIF detection methods among dichotomous items, inclusive of non-IRT methods as well as those based on IRT models. The **DIFlasso** (Schauberger, 2017) and **DIFboost** (Schauberger, 2016) packages implement penalty approach and boosting techniques in the IRT Rasch model. The **GDINA** package (Ma and de la Torre, 2019) offers the implementation of generalized deterministic inputs, noisy, and gate model. The **mirt** package (Chalmers, 2012) provides DIF detection based on various IRT models including those for dichotomous as well as ordinal or nominal items. The **lordif** package (Choi et al., 2016) brings an iterative method based on a mixture of ordinal logistic regression and an IRT approach for DIF detection. Finally, the **psychotree** package (Strobl et al., 2015) offers a method build on model-based recursive partitioning to detect latent groups of subjects exhibiting uniform DIF under the Rasch model. Mentioned

packages do not allow for detecting differences in guessing and inattention, do not offer possibility of various matching criteria, and/or they assume IRT models which may become computationally demanding.

A potential gap in DIF methodology can be filled by generalizations of logistic regression models implemented within the **difNLR** package described here. These nonlinear extensions include estimations of pseudo-guessing parameters as proposed by Drabinová and Martinková (2017). However, the **difNLR** package goes further by introducing a wide range of nonlinear regression models which cover both guessing and the possibility of inattention and allow for the testing of group differences in these parameters or their combinations. Moreover, in the case of ordinal data from rating scales or items allowing for partial credit, the **difNLR** package is able to detect differential use of the scale among groups by adjacent category logit or cumulative logit models. Finally, in the case of nominal data from multiple choice items, the package offers a multinomial model to test for so-called differential distractor functioning (DDF, Green et al., 1989), a situation when respondents with the same ability or knowledge but from different social groups are attracted differently by offered distractors (incorrect option choices). In addition, the **difNLR** package provides features such as item purification (see e.g., Lord, 1980) or corrections for multiple comparisons (see e.g., Kim and Oshima, 2013).

This paper gives a detailed description of the **difNLR** package with functional examples, from data generation, through the model fitting, to interpretation of the results and their graphical representation, while separate parts are dedicated to features and troubleshooting. To bring the concepts together, we conclude by illustrative real data example connecting the usage of the main functions of the package.

The main advantage of non-IRT DIF detection methods implemented within the **difNLR** package is their flexibility with respect to the issue of guessing or inattention, while they retain pleasant properties such as low ratio of convergence issues when compared to IRT models (Drabinová and Martinková, 2017). In addition to that, the **difNLR** provides DIF and DDF detection methods among ordinal and nominal data and thus offers a wide range of techniques to deal with an important topic of detecting potentially unfair items. For users who are new to R, interactive implementation of **difNLR** functions within the **ShinyItemAnalysis** package (Martinková and Drabinová, 2018) with toy datasets may be helpful.

Generalized logistic models for DIF and DDF detection

The class of generalized logistic models described here includes nonlinear regression models for DIF detection in dichotomously scored items, cumulative logit and adjacent category logit models for DIF detection in ordinal items, and a multinomial model for DDF detection in the case when items are nominal (e.g., multiple-choice).

Nonlinear regression models for binary items

Nonlinear regression models for DIF detection among binary items are extensions of the logistic regression method (Swaminathan and Rogers, 1990). These extensions account for the possibility that an item can be guessed, in other words correctly answered without possessing the necessary knowledge, i.e., lower asymptote of the item characteristic curve can be larger than zero (Drabinová and Martinková, 2017). Similarly, these models account for a situation when an item is incorrectly answered by a person with high ability, e.g., due to inattention or lack of time, i.e., upper asymptote of the item characteristic curve can be lower than one.

The probability of a correct answer on item i by person p is then given by

$$P(Y_{ip} = 1|X_p, G_p) = c_{iG_p} + (d_{iG_p} - c_{iG_p}) \frac{e^{a_{iG_p}(X_p - b_{iG_p})}}{1 + e^{a_{iG_p}(X_p - b_{iG_p})}}, \quad (1)$$

where X_p is a variable describing observed knowledge or ability of person p and G_p stands for their membership to social group ($G_p = 1$ for focal group, usually seen as the disadvantaged one, $G_p = 0$ for the reference group). Terms $a_{iG_p}, b_{iG_p} \in \mathbb{R}$, and $c_{iG_p}, d_{iG_p} \in [0, 1]$ represent discrimination, difficulty, probability of guessing, and a parameter related to the probability of inattention in item i , while they can differ for the reference and the focal group (denoted by the index G_p). Further, we use parametrization $a_{iG_p} = a_i + a_{iDIF}G_p$, where a_{iDIF} is a difference between the focal and the reference group in discrimination (analogously for other parameters). In other words, $a_{i0} = a_i$ for the reference group and $a_{i1} = a_i + a_{iDIF}$ for the focal group. Thus, there are eight parameters for each item in total. For simplicity of the formulas, we stick with the notation of $a_{iG_p}, b_{iG_p}, c_{iG_p}$,

and d_{iG_p} .

The class of models determined by equation (1) contains a wide range of methods for DIF detection. For instance, with $d_{iG_p} = 1$, we get a nonlinear model for the detection of DIF allowing for differential guessing in groups as presented by Drabinová and Martinková (2017). Assuming moreover $c_{iG_p} = 0$, one can obtain a classical logistic regression model for the detection of uniform and non-uniform DIF as proposed by Swaminathan and Rogers (1990).

In contrast to IRT models, model (1) assumes that knowledge X_p is represented by the total test score or its standardized form (Z-score) and thus the described method belongs to the class of non-IRT approaches. As such, model (1) can be seen as a proxy to the 4PL IRT model for DIF detection. While estimation of the asymptote parameters is notoriously challenging in IRT models, it was shown that generalized logistic models require a smaller sample size to be fitted while they keep pleasant properties in terms of power and rejection rates (Drabinová and Martinková, 2017).

The **difNLR** package offers two approaches to estimate parameters of model (1). The first option is the nonlinear least squares method (Dennis et al., 1981; Ritz and Streibig, 2008), that is, minimization of the residual sums of squares (RSS) for item i with respect to item parameters:

$$\text{RSS}(a_{ig_p}, b_{ig_p}, c_{ig_p}, d_{ig_p}) = \sum_{p=1}^n \left\{ y_{ip} - c_{ig_p} - \frac{(d_{ig_p} - c_{ig_p})}{1 + e^{-a_{ig_p}(x_p - b_{ig_p})}} \right\}^2,$$

where n denotes number of respondents, y_{ip} is the response of respondent p to the item i , x_p is their (standardized) test score, and g_p is their group membership. The second option is the maximum likelihood method. Likelihood function of item i

$$L(a_{ig_p}, b_{ig_p}, c_{ig_p}, d_{ig_p}) = \prod_{p=1}^n \left\{ c_{ig_p} + \frac{d_{ig_p} - c_{ig_p}}{1 + e^{-a_{ig_p}(x_p - b_{ig_p})}} \right\}^{y_{ip}} \left\{ 1 - c_{ig_p} - \frac{d_{ig_p} - c_{ig_p}}{1 + e^{-a_{ig_p}(x_p - b_{ig_p})}} \right\}^{1-y_{ip}}$$

is maximized with respect to item parameters.

Regression models for ordinal and nominal items

The logistic regression procedure which estimates the probability of the correct answer can be extended to estimate the probability of partial credit scores or option choices. When the responses are ordinal, DIF detection can be performed using the cumulative logit regression model (see e.g., Agresti, 2010, Chapter 3). For $K + 1$ outcome categories, i.e., $Y \in \{0, 1, \dots, K\}$, the cumulative probability is

$$P(Y_{ip} \geq k | X_p, G_p) = \frac{e^{a_{iG_p}(X_p - b_{ikG_p})}}{1 + e^{a_{iG_p}(X_p - b_{ikG_p})}}, \quad (2)$$

for $k = 1, \dots, K$, where $P(Y_{ip} \geq 0 | X_p, G_p) = 1$ and $b_{ikG_p} = b_{ik} + b_{iDIF}G_p$. The parameter b_{iDIF} can be interpreted as the difference in difficulty of item i between the reference and the focal group. The parameter b_{ik} can be seen as category k specific difficulty of item i which is the same for both groups. The category probability is then given by

$$P(Y_{ip} = k | X_p, G_p) = P(Y_{ip} \geq k | X_p, G_p) - P(Y_{ip} \geq k + 1 | X_p, G_p),$$

for categories $k = 0, \dots, K - 1$ while $P(Y_{ip} = K | X_p, G_p) = P(Y_{ip} \geq K | X_p, G_p)$. Again, knowledge is represented by observed (standardized) total test score X_p and therefore model (2) can be seen as a proxy to a graded response IRT model (Samejima, 1969).

Another approach for DIF detection in ordinal data is the adjacent category logit model (see e.g., Agresti, 2010, Chapter 4), which for $K + 1$ outcome categories is given by

$$\log \frac{P(Y_{ip} = k | X_p, G_p)}{P(Y_{ip} = k - 1 | X_p, G_p)} = a_{iG_p}(X_p - b_{ikG_p}),$$

where $k = 1, \dots, K$ and $b_{ikG_p} = b_{ik} + b_{iDIF}G_p$. The category probability takes the following form:

$$P(Y_{ip} = k | X_p, G_p) = \frac{e^{\sum_{l=0}^k a_{iG_p}(X_p - b_{ilG_p})}}{\sum_{j=0}^K e^{\sum_{l=0}^j a_{iG_p}(X_p - b_{ilG_p})}}, \quad (3)$$

where $k = 1, \dots, K$ and parameters for $k = 0$ are set to zero, i.e., $a_{iG_p}(X_p - b_{i0G_p}) = 0$ and hence

$P(Y_{ip} = 0|X_p, G_p) = \frac{1}{\sum_{j=0}^K e^{\sum_{l=0}^j a_{ilG_p}(X_p - b_{ilG_p})}}$. As such, an adjacent category logit model (3) can be seen as a proxy to the rating scale IRT model (Andrich, 1978). Both models, cumulative logit and adjacent category logit, are estimated by iteratively re-weighted least squares.

When responses are nominal (e.g., multiple choice), DDF detection can be performed with the multinomial model. Considering that $k = 0, \dots, K$ possible option choices are offered, with $k = 0$ being the correct answer and other ones distractors, the probability of choosing distractor k is given by

$$P(Y_{ip} = k|X_p, G_p) = \frac{e^{a_{ikG_p}(X_p - b_{ikG_p})}}{\sum_{l=0}^K e^{a_{ilG_p}(X_p - b_{ilG_p})}}, \quad (4)$$

while for the correct answer $k = 0$ the parameters are set to zero, i.e., $a_{i0G_p}(X_p - b_{i0G_p}) = 0$ and thus $P(Y_{ip} = 0|X_p, G_p) = \frac{1}{\sum_{l=0}^K e^{a_{ilG_p}(X_p - b_{ilG_p})}}$. In contrast to ordinal models (2) and (3), discrimination a_{ikG_p} and difficulty b_{ikG_p} are category-specific and they may vary between groups. The parameters are estimated via neural networks.

Implementation in examples

The **difNLR** package provides implementation of the methods to detect DIF and DDF based on generalized logistic models. Specifically, a nonlinear regression model (1), cumulative logit model (2), adjacent category model (3), and multinomial model (4) are available within functions **difNLR()**, **difORD()**, and **ddfMLR()** (see Table 1). All three functions were designed to correspond to one of the most widely used R packages for DIF detection – **difR** (Magis et al., 2010).

Function	Description
difNLR()	Performs DIF detection procedure for dichotomous data based on a nonlinear regression model (generalized logistic regression) and either likelihood-ratio or F test of the submodel.
difORD()	Performs DIF detection procedure for ordinal data based on either an adjacent category logit model or a cumulative logit model and likelihood ratio test of the submodel.
ddfMLR()	Performs DDF detection procedure for nominal data based on a multinomial log-linear regression model and likelihood ratio test of the submodel.

Table 1: DIF and DDF detection methods available in the **difNLR** package.

DIF detection

In this part, we discuss implementation and usage of the **difNLR()** function which offers a wide range of methods for DIF detection among dichotomous data based on a generalized logistic regression model (1). The full syntax of the **difNLR()** function is

```
difNLR(
  Data, group, focal.name, model, constraints,
  type = "all", method = "nls", match = "zscore",
  anchor = NULL, purify = FALSE, nrIter = 10,
  test = "LR", alpha = 0.05, p.adjust.method = "none",
  start, initboot = TRUE, nrBo = 20
)
```

To detect DIF using the **difNLR()** function, the user always needs to provide four pieces of information: 1. the binary data set, 2. the group membership vector, 3. the indication of the focal group, and 4. the model.

Data. **Data** is a **matrix** or a **data.frame** with rows representing dichotomously scored respondents' answers (1 correct, 0 incorrect) and columns which correspond to the items. In addition, **Data** may contain the vector of group membership. If so, the **group** is a column identifier of the **Data**. Otherwise, the **group** must be a dichotomous vector of the same length as the number of rows (respondents) in **Data**. The name of the focal group is specified in **focal.name** argument.

Data generation. To run a simulation study or to create an illustrative example, the **difNLR** package contains a data generator **genNLR()**, which can be used to generate dichotomous, ordinal, or nominal data. The type of items to be generated can be specified via **itemtype** argument: **itemtype = "dich"** for dichotomous items, **"ordinal"** for ordinal items, and **"nominal"** for nominal items.

For the generation of dichotomous items, discrimination and difficulty parameters need to be specified within **a** and **b** arguments in the form of matrices with two columns. The first column stands for the reference group and the second one for the focal group. Each row of matrices corresponds to one item. Additionally, one can provide guessing and inattention parameters via arguments **c** and **d** in the same way as for discriminations and difficulties. By default, values of guessing parameters are set to 0 in both groups, and the values of inattention parameters to 1 in both groups.

Distribution of the underlying latent trait is considered to be Gaussian. The user can specify its mean and standard deviation via arguments **mu** and **sigma** respectively. By default, mean is 0 and standard deviation is 1 and they are the same for both groups.

Furthermore, the user needs to provide a sample size (**N**) and the ratio of respondents in the reference and focal group (**ratio**). The latent trait for both groups is then generated and together with item parameters is used to generate item data. Output of the **genNLR()** function is a **data.frame** with items represented by columns and responses to them represented by rows. The last column is a group indicator, where 0 stands for a focal group and 1 indicates a reference group.

To illustrate generation of dichotomously scored items and to exemplify basic DIF detection with a **difNLR()** function, we create an example dataset. We choose discrimination *a*, difficulty *b*, guessing *c*, and inattention *d* parameters for 15 items. Parameters are then set the same for both groups.

```
# discrimination
a <- matrix(rep(c(1.00, 1.12, 1.45, 1.25, 1.32, 1.38, 1.44, 0.89, 1.15,
                 1.30, 1.29, 1.46, 1.16, 1.26, 0.98), 2), ncol = 2)
# difficulty
b <- matrix(rep(c(1.34, 0.06, 1.62, 0.24, -1.45, -0.10, 1.76, 1.96, -1.53,
                 -0.44, -1.67, 1.91, 1.62, 1.79, -0.21), 2), ncol = 2)
# guessing
c <- matrix(rep(c(0.00, 0.00, 0.00, 0.00, 0.00, 0.17, 0.18, 0.05, 0.10,
                 0.11, 0.15, 0.20, 0.21, 0.23, 0.24), 2), ncol = 2)
# inattention
d <- matrix(rep(c(1.00, 1.00, 1.00, 0.92, 0.87, 1.00, 1.00, 0.88, 0.93,
                 0.94, 0.81, 0.98, 0.87, 0.96, 0.85), 2), ncol = 2)
```

For items 5, 8, 11, and 15, we introduce DIF caused by various sources: In item 5, DIF is caused by a difference in difficulty; in item 8 by discrimination; in item 11, the reference and focal groups differ in inattention, and in item 15 in guessing.

```
b[5, 2] <- b[5, 2] + 1
a[8, 2] <- a[8, 2] + 1
d[11, 2] <- 1
c[15, 2] <- 0
```

We generate dichotomous data with 500 observations in the reference group and 500 in the focal group. We assume that an underlying latent trait comes from a standard normal distribution for both groups (default setting). The output is a **data.frame** where the first 15 columns are dichotomously scored answers of 1,000 respondents and the last column is a group membership variable.

```
set.seed(42)
df <- genNLR(N = 1000, a = a, b = b, c = c, d = d)
head(df[, c(1:5, 16)])
  Item1 Item2 Item3 Item4 Item5 group
1      0     1     1     1     1     0
2      0     1     1     0     1     0
3      0     1     0     0     1     0
4      1     1     1     0     1     0
5      1     1     0     1     1     0
6      0     1     0     0     1     0

DataDIF <- df[, 1:15]
groupDIF <- df[, 16]
```

Model. The last necessary input of the `difNLR()` function is specification of the model to be estimated. This can be made by `model` argument. There are several predefined models, all of them based on the 4PL model stated in equation (1) (see Table 2).

Model annotation	Description
"4PL"	4PL model
"4PLcdg", "4PLc"	4PL model with an inattention parameter set equal for the two groups
"4PLcgd", "4PLd"	4PL model with a guessing parameter set equal for the two groups
"4PLcgdg"	4PL model with a guessing and an inattention parameters set equal for the two groups
"3PLd"	3PL model with an inattention parameter and $c = 0$
"3PLc", "3PL"	3PL model with a guessing parameter and $d = 1$
"3PLdg"	3PL model with an inattention parameter set equal for the two groups
"3PLcg"	3PL model with a guessing parameter set equal for the two groups
"2PL"	Logistic regression model, i.e. $c = 0$ and $d = 1$
"1PL"	1PL model with a discrimination parameter set equal for the two groups
"Rasch"	1PL model with a discrimination parameter fixed on value of 1 for the two groups

Table 2: Predefined models for the `model` argument in the `difNLR()` function.

We are now able to perform the basic DIF detection with a 4PL model for all the items on a generated example dataset `DataDIF`.

```
(fit1 <- difNLR(DataDIF, groupDIF, focal.name = 1, model = "4PL"))
Detection of all types of differential item functioning
using generalized logistic regression model
```

```
Generalized logistic regression likelihood ratio chi-square statistics
based on 4PL model
```

```
Parameters were estimated with nonlinear least squares
```

```
Item purification was not applied
No p-value adjustment for multiple comparisons
```

	Chisq-value	P-value
Item1	6.2044	0.1844
Item2	0.2802	0.9911
Item3	2.7038	0.6086
Item4	5.8271	0.2124
Item5	48.0052	0.0000 ***
Item6	7.2060	0.1254
Item7	3.2390	0.5187
Item8	16.8991	0.0020 **
Item9	2.1595	0.7064
Item10	4.6866	0.3210
Item11	69.5328	0.0000 ***
Item12	8.1931	0.0848 .
Item13	2.5850	0.6295
Item14	2.9478	0.5666
Item15	20.6589	0.0004 ***

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

```
Detection thresholds: 9.4877 (significance level: 0.05)
```

```
Items detected as DIF items:
Item5
Item8
Item11
```

Item15

The output returns values of the test statistics for DIF detection, corresponding p-values, and set of items which are detected as functioning differently. All items (5, 8, 11, and 15) are correctly identified.

Estimates of parameters can be viewed with `coef()` method. Method `coef()` returns a list of parameters, which can be simplified to a matrix by setting `simplify = TRUE`. Each row then corresponds to one item and columns indicate parameters of the estimated model.

```
round(coef(fit1, simplify = TRUE), 3)
      a     b     c     d   aDif   bDif   cDif   dDif
Item1 1.484 1.294 0.049 1.000  0.000  0.000  0.000  0.000
Item2 1.176 0.153 0.000 1.000  0.000  0.000  0.000  0.000
Item3 1.281 1.766 0.001 1.000  0.000  0.000  0.000  0.000
Item4 1.450 0.421 0.000 1.000  0.000  0.000  0.000  0.000
Item5 1.965 -1.147 0.000 0.868 -0.408  0.769  0.023 -0.006
Item6 1.458 -0.527 0.000 0.954  0.000  0.000  0.000  0.000
Item7 0.888 1.392 0.000 1.000  0.000  0.000  0.000  0.000
Item8 1.162 1.407 0.000 0.866 -0.117  0.974  0.007  0.134
Item9 1.482 -1.337 0.000 0.928  0.000  0.000  0.000  0.000
Item10 1.375 -0.570 0.007 0.967  0.000  0.000  0.000  0.000
Item11 1.071 -1.027 0.000 0.969  1.173 -0.499  0.000  0.011
Item12 1.051 1.560 0.080 1.000  0.000  0.000  0.000  0.000
Item13 1.009 1.348 0.084 1.000  0.000  0.000  0.000  0.000
Item14 1.093 1.659 0.141 1.000  0.000  0.000  0.000  0.000
Item15 0.875 -0.565 0.000 0.945  0.205  0.348  0.000 -0.142
```

The user can also print standard errors of the estimates using an option `SE = TRUE`. For example, estimated difference in difficulty between the reference and the focal groups in item 5 is 0.769 with standard error of 0.483.

```
round(coef(fit1, SE = TRUE)[[5]], 3)
      a     b     c     d   aDif   bDif   cDif   dDif
estimate 1.965 -1.147 0.000 0.868 -0.408  0.769  0.023 -0.006
SE       0.844  0.404 0.307 0.044  1.045  0.483  0.345  0.093
```

The `difNLR()` function provides a visual representation of the item characteristic curves using the `ggplot2` package (Wickham, 2016) and its graphical environment. Curves are always based on results of a DIF detection procedure – when an item displays DIF, two curves are plotted, one for the reference and one for the focal group. Curves are accompanied by points representing empirical probabilities, i.e., proportions of correct answers with respect to the ability level and group membership. Size of the points is determined by the number of respondents at this ability level. Characteristic curves may simply be rendered with method `plot()` and by specifying items to be plotted. We show here characteristic curves for DIF items only (Figure 1).

```
plot(fit1, item = fit1$DIFitems)
```

Besides predefined models (see Table 2), all parameters of the model can be further constrained using argument `constraints` specifying which parameters should be set equally for the two groups. For example, choice "ac" in 4PL model means that discrimination parameter *a* and pseudo-guessing parameter *c* are set equally for the two groups while the remaining parameters (*b* and *d*) are not. In addition, both arguments `model` and `constraints` are item-specific, meaning that a single value for all items can be introduced as well as a vector specifying the setting for each item. While the model specification can be challenging, this offers a wide range of models for DIF detection which goes hand in hand with the complexity of the offered method.

Furthermore, via `type` argument one can specify which type of DIF to test. Default option `type = "all"` allows one to test the difference in any parameter which is not constrained to be the same for both groups. Uniform DIF (difference in difficulty *b* only) can be tested by setting `type = "udif"`, while nonuniform DIF (difference also in discrimination *a*) by setting `type = "nudif"`. With the argument `type = "both"`, the differences in both parameters (*a* and *b*) are tested. Moreover, to identify DIF in more detail, one can determine in which parameters the difference should be tested. The argument `type` is also item-specific.

```
# item-specific model
model <- c("1PL", rep("2PL", 2), rep("3PL", 2), rep("3PLd", 2), rep("4PL", 8))
fit2 <- difNLR(DataDIF, groupDIF, focal.name = 1, model = model, type = "all")
fit2$DIFitems
```

```
[1] 5 8 11 15
# item-specific type
type <- rep("all", 15)
type[5] <- "b"; type[8] <- "a"; type[11] <- "c"; type[15] <- "d"
fit3 <- difNLR(DataDIF, groupDIF, focal.name = 1, model = model, type = type)
fit3$DIFitems
[1] 5
# item-specific constraints
constraints <- rep(NA, 15)
constraints[5] <- "ac"; constraints[8] <- "bcd";
constraints[11] <- "abd"; constraints[15] <- "abc"
fit4 <- difNLR(DataDIF, groupDIF, focal.name = 1, model = model,
                 constraints = constraints, type = type)
fit4$DIFitems
[1] 5 8 11 15
```

In `fit2` we allowed different models for items. In `fit3`, when items were intended to function differently, we tested only the difference in those parameters which were selected to be a source of DIF when we generated data, while using the same item-specific models as for `fit2`. Finally, in items which were intended to function differently we constrained all other parameters to be the same for both groups in `fit4`. As expected, models `fit2` and `fit4` correctly identified all DIF items, while `fit3` detected only item 5.

The `difNLR()` function offers two techniques to estimate parameters of a generalized logistic regression model (1). With a default option `method = "nls"`, nonlinear least square estimation is applied using a `nls()` function from the `stats` package. With an option `method = "likelihood"`, the maximum likelihood method is used via an `optim()` function again from the `stats` package. Moreover, with an argument `test`, the user can specify what test of a submodel should be used to analyze DIF. The default option is the likelihood-ratio test.

Fit of selected models can be examined using information criteria, specifically Akaike's criterion (AIC, Akaike, 1974) and Schwarz's Bayesian criterion (BIC, Schwarz et al., 1978).

```
df <- data.frame(AIC = c(AIC(fit2), AIC(fit3), AIC(fit4)),
                  BIC = c(BIC(fit2), BIC(fit3), BIC(fit4)),
                  Fit = paste0("fit", rep(2:4, each = 15)),
                  Item = as.factor(rep(1:15, 3)))
ggplot(df, aes(x = Item, y = AIC, col = Fit)) +
  geom_point(size = 3) +
  scale_color_manual(values = c("#b94685", "#ffbe33", "#61b8ea"))
ggplot(df, aes(x = Item, y = BIC, col = Fit)) +
  geom_point(size = 3) +
  scale_color_manual(values = c("#b94685", "#ffbe33", "#61b8ea"))
```

While there is, not surprisingly, no difference between information criteria of the three models for non-DIF items, a distinction may be observed in DIF items. AIC suggests that model `fit3` fits best to items 8 and 11 and model `fit4` to items 5 and 15, while BIC indicates that for item 8 model `fit4` is the most suitable. However the differences are small (Figure 2). Fit measures can also be displayed for specific items.

```
logLik(fit3, item = 8)
'log Lik.' -312.7227 (df=7)
logLik(fit4, item = 8)
'log Lik.' -316.4998 (df=5)
```

Fitted values and residuals can be standardly calculated with methods `fitted()` and `residuals()`, again for all items or for those specified via `item` argument. This also holds for predicted values and method `predict()`. Predictions for any new respondents can be obtained by `group` and `match` arguments representing group membership and the value of matching criterion (e.g., standardized total score) of the new respondent. For example, with `fit1` in item 5, new respondents with average performance (`match = 0`) have approximately a 22% lower probability of a correct answer if they come from a focal rather than reference group.

```
predict(fit1, item = 5, group = c(0, 1), match = 0)
  item match group      prob
1 Item5     0     0 0.7851739
2 Item5     0     1 0.5624883
```

This can also be observed when comparing item characteristic curves for the reference and the focal group in item 5 (see upper left Figure 1).

DIF detection among ordinal data

Here we show implementation and usage of the `difORD()` function which offers two models – cumulative logit model (2) and adjacent category logit model (3) to detect DIF among ordinal data. The full syntax of the `difORD()` function is

```
difORD(
  Data, group, focal.name, model = "adjacent",
  type = "both", match = "zscore",
  anchor = NULL, purify = FALSE, nrIter = 10, p.adjust.method = "none",
  parametrization = "irt", alpha = 0.05
)
```

To detect DIF among ordinal data using the `difORD()` function, the user needs to provide four pieces of information: 1. the ordinal data set, 2. the group membership vector, 3. the indication of the focal group, and 4. the model to be fitted.

Data. `Data` takes a similar format as used for the `difNLR()` function, however, rows represent ordinally scored respondents' answers instead of dichotomous. Specifications of `group` and `focal.name` remain the same.

Data generation. Data generator `genNLR()` is able to generate ordinal data using an adjacent category logit model (3) by setting `itemtype = "ordinal"`. For polytomous items (ordinal or nominal), `a` and `b` have the form of matrices as well as for dichotomous data but each column now represents parameters of partial scores (or distractors). For example, to generate an item with 4 partial scores (i.e., 0-3), the user needs to provide 3 sets of discrimination and difficulty parameters. The parameters for minimal partial scores (i.e., 0; or correct answer in the case of nominal data) do not need to be specified because their probabilities are calculated as a complement to the sums of the partial scores probabilities. Guessing and inattention parameters are disregarded.

To illustrate usage of the `difORD()` function, we created an example ordinal dataset with 5 items, each scored with a range of 0-4. We first generated discrimination parameters a and difficulties b from a uniform distribution for partial scores $k = 1, \dots, 4$ for each item. In an adjacent category logit model (3), parameter b_{ik} corresponds to an ability level for which the response categories k and $k - 1$ intersect in item i . For this reason and to create well-functioning items, parameters b_{ik} are sorted so that $b_{ik} < b_{ik+1}$. The parameters are set the same for both the reference and focal group.

```
set.seed(42)
# discrimination
a <- matrix(rep(runif(5, 0.25, 1), 8), ncol = 8)
# difficulty
b <- t(sapply(1:5, function(i) rep(sort(runif(4, -1, 1)), 2)))
```

For the first two items we introduce uniform and non-uniform DIF respectively.

```
b[1, 5:8] <- b[1, 5:8] + 0.1
a[2, 5:8] <- a[2, 5:8] - 0.2
```

Using parameters `a` and `b` of an adjacent category logit model, we generated ordinal data with a total sample size of 1,000 (500 observations per group). The first 5 columns of dataset `DataORD` represent ordinally scored items, while the last column represents a group membership variable.

```
DataORD <- genNLR(N = 1000, itemtype = "ordinal", a = a, b = b)
summary(DataORD)
```

Item1	Item2	Item3	Item4	Item5	group
0:488	0:376	0:417	0:530	0:556	Min. :0.0
1:229	1:237	1:331	1:226	1:253	1st Qu.:0.0
2:150	2:195	2:170	2:129	2:123	Median :0.5
3: 93	3:114	3: 71	3: 83	3: 47	Mean :0.5
4: 40	4: 78	4: 11	4: 32	4: 21	3rd Qu.:1.0
					Max. :1.0

Model. The last input of the `difORD()` function which needs to be specified is `model`. It offers two possibilities. With an option `model = "cumulative"` a cumulative logit model (2) is fitted, while with an option `model = "adjacent"` (default) DIF detection is performed using an adjacent category logit model (3). The parameters for both models are estimated via `vgam()` function from the **VGAM** package (Yee, 2010).

DIF detection with cumulative logit model. In this part we exemplify usage of the `difORD()` function to fit a cumulative logit model for DIF detection among ordinal data. The `group` argument is introduced here by specifying the name of the group membership variable in `DataORD` dataset, i.e., `group = "group"`. Knowledge is represented by observed standardized total score, i.e., standardized sum of all item scores.

```
(fit5 <- difORD(DataORD, group = "group", focal.name = 1, model = "cumulative"))
Detection of both types of Differential Item
Functioning for ordinal data using cumulative logit
regression model

Likelihood-ratio Chi-square statistics

Item purification was not applied
No p-value adjustment for multiple comparisons

      Chisq-value P-value
Item1  7.4263    0.0244 *
Item2 13.4267    0.0012 **
Item3  0.6805    0.7116
Item4  5.6662    0.0588 .
Item5  2.7916    0.2476

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

Items detected as DIF items:
Item1
Item2
```

Output provides test statistics for the likelihood ratio test, corresponding p-values, and the set of items which were detected as functioning differently. Items 1 and 2 are correctly identified as DIF items.

Similarly as for the `difNLR()` function, characteristic curves can be imaged with the `plot()` method. Besides characteristic curves, the method `plot()` for the cumulative logit model also offers the plot of cumulative probabilities. This can be achieved using `plot.type = "cumulative"`, while with `plot.type = "category"` characteristic curves are shown. The plot of cumulative probabilities shows only 4 partial scores and does not show the cumulative probability of $P(Y_{ip} \geq 0)$ since it is always equal to 1. Note that category probability of the highest score corresponds to its cumulative probability (Figure 3).

```
plot(fit5, item = "Item1", plot.type = "cumulative")
plot(fit5, item = "Item1", plot.type = "category")
```

Similarly to the `difNLR()` function, `difORD()` offers fit measures provided by `AIC()`, `BIC()`, and `logLik()` S3 methods, and method `coef()` to print the estimated parameters of the fitted model.

DIF detection with adjacent logit model. We illustrate here the fitting of an adjacent category logit model for DIF detection using the `difORD()` function. The `group` argument is now introduced by specifying the identifier of a group membership variable in `Data` (i.e., `group = 6`).

```
(fit6 <- difORD(DataORD, group = 6, focal.name = 1, model = "adjacent"))
Detection of both types of Differential Item
Functioning for ordinal data using adjacent category
logit model

Likelihood-ratio Chi-square statistics

Item purification was not applied
No p-value adjustment for multiple comparisons
```

```

  Chisq-value P-value
Item1 8.9024      0.0117 *
Item2 12.9198     0.0016 **
Item3 1.0313      0.5971
Item4 4.3545      0.1134
Item5 2.3809      0.3041

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

```

```

Items detected as DIF items:
Item1
Item2

```

Output again provides test statistics for the likelihood ratio test, corresponding p-values, and the set of items which were detected as functioning differently. Items 1 and 2 are correctly identified as DIF items. Characteristic curves can again be rendered using the `plot()` method (Figure 4).

```
plot(fit6, item = fit6$DIFitems)
```

Function `difORD()` offers the possibility to specify parametrization of regression coefficients with an argument `parametrization`. By default, IRT parametrization as stated in (2) and (3) is utilized using `parametrization = "irt"`, but also classical intercept-slope parametrization (`parametrization = "classic"`) with the effect of group membership and its interaction with matching criterion as covariates may be applied, i.e., $b_{0kG_p} + b_{1G_p}X_p$ instead of $a_{iG_p}(X_p - b_{ikG_p})$). The DIF detection is the same as with IRT parametrization, the only difference can be found in parameter estimates:

```

fit6a <- difORD(DataORD, group = 6, focal.name = 1, model = "adjacent",
                  parametrization = "classic")
# coefficients with IRT parametrization
round(coef(fit6)[[1]], 3)
  b1    b2    b3    b4      a  bDIF1  bDIF2  bDIF3  bDIF4   aDIF
  0.013  0.603  1.500  2.500  1.776 -0.042 -0.121 -0.240 -0.374  0.273
# coefficients with classical intercept-slope parametrization
round(coef(fit6a)[[1]], 3)
(Intercept):1 (Intercept):2 (Intercept):3 (Intercept):4      x  group  x:group
  -0.023       -1.070      -2.664      -4.441     1.776   0.082   0.273

```

Note that estimated discrimination for the reference group (parameter `a`) corresponds to the effect of matching criterion `x`, and in both cases their value is 1.776 for item 1. The same holds for the difference in discrimination and the effect of interaction between the matching criterion and group membership.

DDF detection among nominal data

Function `ddfMLR()` offers DDF detection among nominal data with the multinomial model (4). Here we illustrate its implementation and usage on a generated example. The full syntax of the `ddfMLR()` function is:

```

ddfMLR(
  Data, group, focal.name, key,
  type = "both", match = "zscore",
  anchor = NULL, purify = FALSE, nrIter = 10, p.adjust.method = "none",
  parametrization = "irt", alpha = 0.05
)

```

To detect DDF among nominal data using the `ddfMLR()` function, the user needs to provide four pieces of information: 1. the unscored data set, 2. the key of correct answers, 3. the group membership vector, and 4. the indication of the focal group. The parameters are estimated via `multinom()` function from the `nnet` package (Venables and Ripley, 2002).

Data. The format of `Data` argument is similar to previously described functions. However, rows here represent respondents' unscored answers (e.g., in ABCD format). The `group` and `focal.name` is specified as in `difNLR()` or `difORD()` functions.

Data generation. Data generator `genNLR()` can be used to generate nominal data using a multinomial model (4) by setting `itemtype = "nominal"`. Specification of arguments `a` and `b` is the same as for ordinal items, however, it now represents parameters for distractors (incorrect answers).

To create an illustrative example dataset of nominal data, we must first generate discrimination a and difficulty b parameters from a uniform distribution for distractors of 10 items. The parameters are set the same for the reference and the focal group. For the first 5 items, we consider only two distractors (i.e., three item choices in total). For the last 5 items, we consider three distractors (i.e., four item choices in total).

```
set.seed(42)
# discrimination
a <- matrix(rep(runif(30, -2, -0.5), 2), ncol = 6)
a[1:5, c(3, 6)] <- NA
# difficulty
b <- matrix(rep(runif(30, -3, 1), 2), ncol = 6)
b[1:5, c(3, 6)] <- NA
```

For item 1, we introduce DDF by difference in discrimination and for item 6 by difference in difficulty.

```
a[1, 4] <- a[1, 1] - 1; a[1, 5] <- a[1, 2] + 1
b[6, 4] <- b[6, 1] - 1; b[6, 5] <- b[6, 2] - 1.5
```

Finally, we generate nominal data with 500 observations in each group, i.e. 1,000 in total. The first 10 columns of the generated dataset `DataDDF` represent the unscored answers of respondents and the last column describes a group membership variable.

```
DataDDF <- genNLR(N = 1000, itemtype = "nominal", a = a, b = b)
head(DataDDF)

  Item1 Item2 Item3 Item4 Item5 Item6 Item7 Item8 Item9 Item10 group
1     B     B     C     A     C     B     B     D     B     B     0
2     C     A     B     A     C     C     B     B     C     C     0
3     B     C     C     B     C     C     B     C     B     D     0
4     B     A     C     A     C     B     A     B     B     B     0
5     B     B     C     B     C     B     A     C     A     B     0
6     B     A     A     A     A     B     B     A     A     A     0
```

The correct answers in the generated dataset are denoted by A for each item; the key is hence a vector of As with a length of 10.

Now we have all the necessary inputs to fit a multinomial model (4) using the `ddfMLR()` function. The `group` argument is introduced here by specifying the name of group membership variable in `Data` (i.e., `group = "group"`). For the generated data, the total score is calculated as number of correct answers (i.e., number of As on a given row) and the matching criterion is then its standardized value (Z-score).

```
(fit7 <- ddfMLR(DataDDF, group = "group", focal.name = 1, key = rep("A", 10)))
Detection of both types of Differential Distractor
Functioning using multinomial log-linear regression model
```

Likelihood-ratio chi-square statistics

```
Item purification was not applied
No p-value adjustment for multiple comparisons
```

	Chisq-value	P-value
Item1	29.5508	0.0000 ***
Item2	1.1136	0.8921
Item3	1.0362	0.9043
Item4	4.1345	0.3881
Item5	7.4608	0.1134
Item6	47.0701	0.0000 ***
Item7	1.3285	0.9701
Item8	2.3629	0.8835
Item9	10.4472	0.1070
Item10	3.5602	0.7359

```

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

Items detected as DDF items:
Item1
Item6

```

The output again summarizes the statistics of likelihood ratio test of a submodel, corresponding p-values, and the set of items identified as DDF. As expected, items 1 and 6 are detected as DDF.

Their characteristic curves can be displayed with a `plot()` method while the name of the reference and focal group can be modified via `group.names` argument (Figure 5). This option is also available for functions `difNLR()` and `difORD()` and their plotting methods.

```
plot(fit7, item = fit7$DDFitems, group.names = c("Group 1", "Group 2"))
```

Similarly as for `difNLR()` and `difORD()`, item fit measures are offered via `AIC()`, `BIC()`, and `logLik()` S3 methods. Parameter estimates can be obtained using the method `coef()`.

Further features

The `difNLR` covers user-friendly features that are common in standard DIF software – various matching criteria, anchor items, item purification, and p-value adjustments. These features are available for all main functions – `difNLR()`, `difORD()`, and `ddfMLR()`.

Matching criterion. The models covered in the `difNLR` package are extensions of the logistic regression model described by Swaminathan and Rogers (1990) who considered the total test score as an observed ability X_p . While this approach is well rooted in the psychometric research, note that it may lead to contradictions, e.g. when a nonzero item score is predicted for a respondent with a zero total test score. In the `difNLR` package, the default observed ability considered in all three main functions is the standardized total score. However, this estimate can be changed using the `match` argument. Besides default option "zscore" (standardized total score), it can also be the total test score (`match = "score"`) or any numeric vector of the same length as the number of respondents. It is hence possible to use, for instance, latent trait estimates provided by some IRT models, or to use a pre-test score instead of the total score of the current test to be examined and analyze DIF in the context of item validity.

Anchor items and item purification. Including DIF items into the calculation of matching criterion can lead to a potential bias and misidentification of DIF and non-DIF items. With an argument `anchor`, one can specify which items are supposed to be used for the calculation of matching criterion.

In the following examples, we go back to our generated dichotomous dataset `DataDIF` and the `difNLR()` function. For illustration, we take only items 1-6 and we apply some features with the 4PL model. Matching criterion is now calculated as a total test score based on items 1-6. Similar examples can also be illustrated with functions `difORD()` and `ddfMLR()`.

We start with not specifying the anchor items. This indicates that any item can be considered as DIF one.

```

fit8a <- difNLR(DataDIF[, 1:6], groupDIF, focal.name = 1, match = "score",
                  model = "4PL", type = "all")
fit8a$DIFitems
[1] 5 6

```

Initial fit `fit8a` detected items 5 a 6 as functioning differently. Now we can set all items excluding these two as the anchors.

```

fit8b <- difNLR(DataDIF[, 1:6], groupDIF, focal.name = 1, match = "score",
                  model = "4PL", type = "all", anchor = 1:4)
fit8b$DIFitems
[1] 5

```

With a test score based only on DIF-free items 1-4 (i.e., excluding potentially unfair items 5 and 6 detected in previous run from calculation of the total score), we detected only item 5 as functioning differently. We could again fit the model excluding only item 5 from calculation of matching criterion.

The process of including and omitting DIF and potentially unfair items could be demanding and time consuming. However, this process can be applied iteratively and automatically. This procedure is called item purification (Lord, 1980; Marco, 1977) and it has been shown that it can improve DIF detection. Item purification can be accessed with a `purify` argument. This can only be done when the matching criterion is either the total score or Z-score. The maximal number of iterations is determined by the `nrIter` argument, where the default value is 10.

```
fit9 <- difNLR(DataDIF[, 1:6], groupDIF, focal.name = 1, match = "score",
                 model = "4PL", type = "all", purify = TRUE)
```

Item purification was run with 2 iterations plus one initial step. The process of including and excluding items into the calculation of matching criterion can be found in the `difPur` element of the output.

```
fit9$difPur
  Item1 Item2 Item3 Item4 Item5 Item6
Step0    0     0     0     0     1     1
Step1    0     0     0     0     1     0
Step2    0     0     0     0     1     0
```

In the initial step, items 5 and 6 were identified as DIF as it was shown with `fit8a`. The matching criterion was then calculated as the sum of the correct answers in items 1-4 as demonstrated by `fit8b`. In the next step, only item 5 was identified as DIF and the matching criterion was based on items 1-4 and 6. The result of the DIF detection procedure was the same in the next step and the item purification process thus ended.

Multiple comparison corrections. As the DIF detection procedure is done item by item, corrections for multiple comparisons may be considered (see Kim and Oshima, 2013). For example, applying Holm's adjustment (Holm, 1979) results in item 5 being detected as DIF.

```
fit10 <- difNLR(DataDIF[, 1:6], groupDIF, focal.name = 1, match = "score",
                  model = "4PL", type = "all", p.adjust.method = "holm")
fit10$DIFitems
[1] 5
```

And of course, item purification and multiple comparison corrections can be combined in a way that the p-value adjustment is applied for a final run of the item purification.

```
fit11 <- difNLR(DataDIF[, 1:6], groupDIF, focal.name = 1, match = "score",
                  model = "4PL", type = "all", p.adjust.method = "holm",
                  purify = TRUE)
fit11$DIFitems
[1] 5
```

While all three approaches correctly identify item 5 as a DIF item, the significance level varies:

```
round(fit9$pval, 3)
[1] 0.144 0.974 0.244 0.507 0.000 0.126
round(fit10$adj.pval, 3)
[1] 1.000 1.000 1.000 0.747 0.000 0.137
round(fit11$adj.pval, 3)
[1] 0.629 1.000 0.733 1.000 0.000 0.629
```

Troubleshooting

In this section, we focus on several issues which can be encountered when fitting generalized logistic regression models and using the features offered in the `difNLR` package.

Convergence issues. First, there is no guarantee that the estimation process in the `difNLR()` function will always end successfully. For instance, in the case of a small sample size, convergence issues may appear.

The easiest way to fix such issues is to specify different starting values. Various starting values can be applied via a `start` argument as a list with named numeric vectors as its elements. Each element needs to include values for the parameters `a`, `b`, `c`, and `d` of the reference group and the differences between reference and focal groups denoted by `aDif`, `bDif`, `cDif`, and `dDif`. However,

there is no need to determine initial values manually. In the instance of convergence issues, the initial values are by default automatically re-calculated based on bootstrapped samples and applied only to models that failed to converge. This is also performed when starting values were initially introduced via a `start` argument. This feature can be turned off by setting `initboot = FALSE`. In such a case, no estimates are obtained for items that failed to converge. To demonstrate described situations, we now use a sample of our original simulated data set.

```
# sampled data
set.seed(42)
sam <- sample(1:1000, 420)
# using re-calculation of starting values
fit12a <- difNLR(DataDIF[sam, ], groupDIF[sam], focal.name = 1, model = "4PL",
                    type = "all")
Starting values were calculated based on bootstraped samples.

# turn off option of re-calculating starting values
fit12b <- difNLR(DataDIF[sam, ], groupDIF[sam], focal.name = 1, model = "4PL",
                    type = "all", initboot = FALSE)
Warning message:
Convergence failure in item 3
Convergence failure in item 14
```

With an option `initboot = TRUE` in `fit12a`, starting values were re-calculated and no convergence issue occurred. When setting `initboot = FALSE` in `fit12b` we observed convergence failures in items 3 and 14.

The re-calculation process is by default performed up to twenty times, but the number of runs can be increased via the `nrBo` argument.

Another option is to apply the maximum likelihood method instead of nonlinear least squares to estimate parameters.

```
fit13 <- difNLR(DataDIF[sam, ], groupDIF[sam], focal.name = 1, model = "4PL",
                    type = "all", method = "likelihood")
```

There is no convergence issue in `fit13` using the maximum likelihood estimation in contrast to `fit12b` and nonlinear least squares option.

Item purification. Issues may also occur when applying an item purification process. Although this is rare in practice, there is no guarantee that the process will end successfully. This can be observed, for instance, when we use `DataDIF` with the first 12 items only.

```
fit14 <- difNLR(DataDIF[, 1:12], groupDIF, focal.name = 1, model = "4PL",
                    type = "all", purify = TRUE)
Warning message:
Item purification process not converged after 10 iterations.
Results are based on the last iteration of the item purification.
```

The maximum number of item purification iterations can be increased via the `nrIter` argument. However, in our example this would not necessarily lead to success as the process was not able to decide whether or not to include item 1 in the calculation of matching criterion.

```
fit14$difPur
      Item1 Item2 Item3 Item4 Item5 Item6 Item7 Item8 Item9 Item10 Item11 Item12
Step0    0    0    0    0    1    0    0    1    0    0    1    0
Step1    1    0    0    0    1    0    0    1    0    0    1    0
Step2    0    0    0    0    1    0    0    1    0    0    1    0
Step3    1    0    0    0    1    0    0    1    0    0    1    0
Step4    0    0    0    0    1    0    0    1    0    0    1    0
Step5    1    0    0    0    1    0    0    1    0    0    1    0
Step6    0    0    0    0    1    0    0    1    0    0    1    0
Step7    1    0    0    0    1    0    0    1    0    0    1    0
Step8    0    0    0    0    1    0    0    1    0    0    1    0
Step9    1    0    0    0    1    0    0    1    0    0    1    0
Step10   0    0    0    0    1    0    0    1    0    0    1    0
```

In this context, we advise considering such items as DIF to be on the safe side. As a general rule, any suspicious item should be reviewed by content experts. Not every DIF item is necessarily

unfair, however even in such a case, understanding the reasons behind DIF may inform educators and help provide the best assessment and learning experience to all individuals involved.

Real data example

To illustrate the work with the **difNLR** package, we present a real data example from a study exploring the effect of student tracking on gains in learning competence (Martinková et al., 2020). The **LearningToLearn** dataset presented here is available in the **ShinyItemAnalysis** package (Martinková and Hladká, 2020). It contains dichotomously scored responses of 782 students in total; 391 from a basic school (BS) track and 391 from a selective academic school (AS) track, whereas each student answered exactly the same test questions in Grade 6 and Grade 9. The sample was created from a larger dataset of 2,917 + 1,322 students using propensity score matching with the aim of obtaining similar baseline achievement and socio-economic characteristics of the students in both tracks. In addition, the matching algorithm required an exact match of the LtL total score in Grade 6 (i.e., exactly the same total score value in each matched pair of BS and AS students).

We demonstrate the three main functions of the **difNLR** package using items from the most difficult subscale 6, called Mathematical concepts, which consists of 8 multiple-choice items (6A, 6B, ..., 6H) with four response options. We first test for DIF in Grade 6 using the **difNLR()** function to provide a detailed analysis of between-group differences on the baseline. Then, to get a more detailed insight into the differences in student gains after three years of education in two different tracks, we perform an analysis of the differential item functioning in change (DIF-C; Martinková et al., 2020) using student item responses in Grade 9 and then also using the nominal and ordinal data of item changes from Grade 6 to Grade 9.

To identify items functioning differently on the baseline, we use a reduced dataset **LtL6_gr6** which includes only student responses to eight subscale 6 items collected in Grade 6 and a group membership variable **track**. We further use a standardized total test score achieved in Grade 6 as an estimate of the students' ability.

```
data("LearningToLearn", package = "ShinyItemAnalysis")
# dichotomous items for Grade 6
LtL6_gr6 <- LearningToLearn[, c("track", paste0("Item6", LETTERS[1:8], "_6"))]
head(LtL6_gr6)
  track Item6A_6 Item6B_6 Item6C_6 Item6D_6 Item6E_6 Item6F_6 Item6G_6 Item6H_6
1 3453    AS      0      0      0      0      0      0      0      1
2 3456    AS      0      0      0      1      1      1      0      0
3 3458    AS      0      0      0      0      0      1      0      1
4 3464    AS      0      0      0      1      0      0      0      1
5 3474    AS      0      1      0      0      1      1      0      1
6 3491    AS      0      0      1      0      0      1      0      0
# standardized total score achieved in Grade 6
zscore6 <- LearningToLearn$score_6
```

Martinková et al. (2020) hypothesized that, on the baseline, DIF found in this difficult subscale may possibly be caused by differences in guessing. To test this hypothesis, we can use the 3PL generalized logistic model which allows for detecting group differences in item difficulty, item discrimination, as well as the probability of guessing.

```
fitex1 <- difNLR(Data = LtL6_gr6, group = "track", focal.name = "AS", model = "3PL",
                    match = zsore6)
fitex1$DIFitems
[1] 8
```

Using the 3PL model, the eighth item (i.e., item 6H) was identified as DIF, while this DIF may have been caused by difference in any of the three item parameters.

In this eighth item (ninth column in the **LtL_gr6** dataset), we now test a more specific hypothesis that DIF is caused by differences in guessing. This can be done by adding **type = "c"**.

```
fitex2 <- difNLR(Data = LtL6_gr6[, c(1, 9)], group = "track", focal.name = "AS",
                    model = "3PL", type = "c", match = zsore6)
fitex2$DIFitems
[1] 1
```

When specifically analyzing differences in the pseudo-guessing parameter c , we again detected a significant DIF in this item (6H). According to the model, for BS the estimated probability of guessing

was close to 0.25, which is to be expected in multiple-choice items providing four response options. In contrast, for AS the probability of guessing was close to 0 (Figure 6).

```
plot(fitex2, item = fitex2$DIFitems)
```

We further perform DIF-C analysis to get a more detailed insight into the differences between AS and BS students in item gains. In contrast to DIF analysis, we use pre-test achievement in DIF-C analysis as the matching criterion while testing for group differences in the post-test, or in changes from pre-test to post-test.

In what follows, we demonstrate use of the `difNLR()`, `difORD()`, and `ddfMLR()` functions to provide evidence of DIF-C as described above. We first create a dataset `LtL6_gr9` which consists of student responses in Grade 9 to subscale 6 items and the track variable.

```
# dichotomous items for Grade 9
LtL6_gr9 <- LearningToLearn[, c("track", paste0("Item6", LETTERS[1:8], "_9"))]
head(LtL6_gr9)
  track Item6A_9 Item6B_9 Item6C_9 Item6D_9 Item6E_9 Item6F_9 Item6G_9 Item6H_9
1 3453    AS      1      0      0      1      0      1      0      1
2 3456    AS      1      1      1      1      1      1      1      1
3 3458    AS      1      1      0      1      1      1      0      0
4 3464    AS      0      0      0      0      1      0      0      1
5 3474    AS      1      1      0      1      0      1      0      0
6 3491    AS      1      1      0      1      1      1      0      1
```

We then use the standardized total score achieved in Grade 6 as an estimate of baseline ability to explore the functioning of items in Grade 9 and the possible differences in their parameters between BS and AS tracks. We next use the 3PL generalized logistic regression model and the `difNLR()` function, as in the previous example.

```
fitex3 <- difNLR(Data = LtL6_gr9, group = "track", focal.name = "AS", model = "3PL",
  match = zscore6)
fitex3$DIFitems
[1] 1 2
```

The first two easier items of the subscale (i.e., items 6A and 6B) were identified as functioning differently in Grade 9 when accounting for the standardized total score in Grade 6. The differences between tracks can be illustrated by comparing the probabilities of answering item 6A correctly in Grade 9 by students with various baseline abilities: A low-performing student, average-performing student, and student performing above average, specifically, students with standardized total scores in Grade 6 equal to -1, 0, and 1. Method `predict()` also offers calculation of confidence intervals using delta method approach. Note that applying delta method in this case can result in confidence intervals slightly exceeding probability bounds 0 and 1.

```
predict(
  fitex3,
  match = rep(c(-1, 0, 1), 2),
  group = rep(c(0, 1), each = 3),
  item = 1,
  interval = "confidence"
)
   item match group     prob lwr.conf upr.conf
1 Item6A_9    -1    0 0.6785773 0.6188773 0.7382773
2 Item6A_9     0    0 0.7773050 0.7269066 0.8277034
3 Item6A_9     1    0 0.8781427 0.8114574 0.9448280
4 Item6A_9    -1    1 0.7802955 0.7187001 0.8418909
5 Item6A_9     0    1 0.8431045 0.7869886 0.8992204
6 Item6A_9     1    1 0.9290780 0.8549489 1.0032071
```

A low-performing student in Grade 6 from BS track has around 10% lower probability of answering item 6A correctly in Grade 9 than a student with the same baseline ability level but from AS track (68%, 95% CI = [62, 74] vs. 78% [72, 84]). Similarly, the probability is 78% [73, 83] vs. 84% [79, 90] for average-performing students and 88% [81, 94] vs. 93% [85, 100] for those performing above average in Grade 6. Note that confidence intervals may overlap even in case when differential item functioning is significant among the two groups.

A second method for testing DIF-C proposed in Martinková et al. (2020) is based upon a multinomial model (4). To demonstrate the usage of `ddfMLR()` we use a restricted dataset of variables combining information about responses in Grades 6 and 9 into four response patterns: "00"

indicating that the student did not answer correctly in either of the grades (did not improve); "10" meaning that the student answered correctly in Grade 6 but incorrectly in Grade 9 (deteriorated); "01" standing for a situation where the student answered incorrectly in Grade 6 but correctly in Grade 9 (improved); "11" describing the case where the student answered correctly in both grades (did not deteriorate). These variables denoted as "`changes`" can be extracted directly from the `LearningToLearn` dataset.

```
# nominal data for changes between 6th and 9th grade
Ltl6_change <- LearningToLearn[, c("track", paste0("Item6", LETTERS[1:8], "_changes"))]
summary(Ltl6_change[, 1:4])
track   Item6A_changes Item6B_changes Item6C_changes
BS:391    00:113        00:186        00:465
AS:391    10: 33         10: 33        10: 36
          01:431        01:414        01:252
          11:205        11:149        11: 29
```

We then fit a multinomial model (4) with the `ddfMLR()` function on `Ltl6_change` data, again using the standardized total score achieved in Grade 6 as an estimate of a student's baseline ability, and by using the vector of patterns "11" as the argument `key`.

```
fitex4 <- ddfMLR(Data = Ltl6_change, group = "track", focal.name = "AS",
                    key = rep("11", 8), match = zscore6)
fitex4$DDFItems
[1] 2 5
```

Using the multinomial model, we identified items 2 and 5 (i.e., items 6B and 6E) as DIF-C items. In both items, the AS track was favoured in patterns "01" and "11", meaning that AS students had a higher probability of improving (pattern "01") or not deteriorating (pattern "11") than students with the same baseline standardized total score from the BS track. On the contrary, students with the same baseline standardized total score had a higher probability of not improving (pattern "00") or even deteriorating (pattern "10") in the BS track than in the AS track (Figure 7).

```
plot(fitex4, item = fitex4$DDFItems)
```

Finally, we introduce a third method for testing DIF-C using ordinal regression models. For this purpose, we create an ordinal dataset combining information about student responses in Grade 6 and Grade 9 into three categories: Score "0" means that the student deteriorated, i.e., answered correctly in Grade 6 but incorrectly in Grade 9; score "1" indicates no change in the accuracy of the answer, i.e., either correct or incorrect in both grades; and the best score "2" stands for improvement, i.e., the student did not answer correctly in Grade 6 but answered correctly in Grade 9.

```
# ordinal data for change between Grade 6 and 9
Ltl6_change_ord <- data.frame(
  track = Ltl6_change$track,
  sapply(Ltl6_change[, -1],
        function(x) as.factor(ifelse(x == "10", 0, ifelse(x == "01", 2, 1))))
)
summary(Ltl6_change_ord[, 1:4])
track   Item6A_changes Item6B_changes Item6C_changes
BS:391    0: 33         0: 33        0: 36
AS:391    1:318        1:335        1:494
          2:431        2:414        2:252
```

We then fit an adjacent category logit model (2), as we are mainly interested in the baseline ability needed to move from one ordinal category into the next one.

```
fitex5 <- difORD(Data = Ltl6_change_ord, group = "track", focal.name = "AS",
                    model = "adjacent", match = zscore6)
fitex5$DIFItems
[1] 2 4 5
```

Using the ordinal model, we identified items 2, 4, and 5 (i.e., items 6B, 6D, and 6E) as functioning differently. In all the items, AS track was favoured in the improvement category "2", while BS students had a higher probability of deteriorating (category "0") and no change (category "1") when compared to AS students with the same baseline ability. The probability of belonging to category "2" was decreasing with an increasing baseline ability in items 6B and 6D, which is reasonable since the students with a higher baseline ability were more likely to have already answered these items correctly while in Grade 6. On the contrary, the probability of falling into the deteriorating

category "0" was slightly increasing with baseline ability in these items, which can be explained as a tendency for over-thinking or inattention in very well-performing students. Trends were different in item 6E, where category probabilities seemed to be stable for all baseline ability levels, however, differences between the BS and AS track remained the same as for items 6B and 6D, favouring the AS track in the improving category "2" (Figure 8).

```
plot(fit5, item = fit5$DIFitems)
```

All three approaches, using functions `difNLR()`, `difORD()`, and `ddfMLR()` jointly confirmed DIF-C in item 6B. Conclusions regarding other items were somewhat ambiguous, nevertheless, this is to be expected given the fact that each approach used different data (binary, nominal, and ordinal) which contained slightly different information.

Similar analysis can be conducted using an interactive environment of the **ShinyItemAnalysis** application (Martinková and Drabinová, 2018) which offers complex psychometric analysis including some functionalities implemented via `difNLR` package (see Figure 9). The datasets used in the real data example are included in the application; moreover, the user may upload and analyze interactively their own datasets as well.

Summary

This article introduced the R package **difNLR** version 1.3.5 for DIF and DDF detection with extensions of a logistic regression model. The release version of the **difNLR** package is hosted on CRAN and the newest development version on GitHub, which can be accessed by `devtools::install_github("adelahladka/difNLR")`. The current paper offered a description of the implementation of its three main functions `difNLR()`, `difORD()`, and `ddfMLR()` using simulated examples as well as a real data example with a longitudinal dataset `LearningToLearn` from the **ShinyItemAnalysis** package.

While the **difR** package already offers a classical logistic regression model for DIF detection via its function `difLogistic()`, the **difNLR** offers a wide range of methods based on extensions of the original model. The package is user-friendly since its model-fitting functions were designed to behave like the analogous functions in the **difR** package. Hence users who are already proficient in DIF detection among dichotomous items using R will find it easy to follow DIF and DDF detection via **difNLR**. In addition, the **difNLR** package offers various S3 methods, including graphical representation of characteristic curves using the `ggplot2` environment, various fit measures, extraction of fitted values and residuals as well as a prediction for a generalized logistic model.

Described models are also accessible via shiny application **ShinyItemAnalysis** which provides not only an interactive environment with a number of toy datasets including the real dataset analyzed here, but also a selected R code which can serve as a springboard for those new to R.

Future plans for the **difNLR** package include extensions of its functionality. In the `difNLR()` function, this comprises more options for estimation algorithms, as well as fine-tuning of the starting values to prevent convergence issues and to improve accuracy of the nonlinear models. We also plan to implement methods to extract fitted values and residuals for the `difORD()` and `ddfMLR()` functions together with their prediction via `predict()` method. Further, we would like to offer confidence and prediction intervals and their graphical representation for all three main functions. Finally, considered extensions include generalizations for longitudinal data with more time points.

In summary, the **difNLR** package provides various methods for DIF and DDF detection and can thus serve as a complex tool for detection of between-group differences in various contexts of educational and psychological tests and their items.

Acknowledgments

We gratefully thank Jon Kern, Eva Potužníková, Michal Kulich, and anonymous reviewers for helpful comments to previous versions of this manuscript. We also thank Jan Netík for computational assistance. The work was supported by the long-term strategic development financing of the Institute of Computer Science (Czech Republic RVO 67985807).

Bibliography

- A. Agresti. *Analysis of ordinal categorical data*. John Wiley & Sons, second edition, 2010. URL <https://doi.org/10.1002/9780470594001>. [p]

- H. Akaike. A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19(6):716–723, 1974. URL <https://doi.org/10.1109/TAC.1974.1100705>. [p]
- D. Andrich. A rating formulation for ordered response categories. *Psychometrika*, 43(4):561–573, 1978. URL <https://doi.org/10.1007/BF02293814>. [p]
- R. P. Chalmers. mirt: A multidimensional item response theory package for the R environment. *Journal of Statistical Software*, 48(6):1–29, 2012. URL <https://doi.org/10.18637/jss.v048.i06>. [p]
- S. W. Choi, with contributions from Laura E. Gibbons, and P. K. Crane. lordif: Logistic ordinal regression differential item functioning using IRT, 2016. URL <https://CRAN.R-project.org/package=lordif>. R package version 0.3-3. [p]
- J. E. J. Dennis, D. M. Gay, and R. E. Welsch. An adaptive nonlinear least-squares algorithm. *Transactions on Mathematical Software*, 7(3):348–368, 1981. URL <https://doi.org/10.1145/355958.355965>. [p]
- A. Drabinová and P. Martinková. Detection of differential item functioning with nonlinear regression: A non-IRT approach accounting for guessing. *Journal of Educational Measurement*, 54(4):498–517, 2017. URL <https://doi.org/10.1111/jedm.12158>. [p]
- B. F. Green, C. R. Crone, and V. G. Folk. A method for studying differential distractor functioning. *Journal of Educational Measurement*, 26(2):147–160, 1989. URL <https://doi.org/10.1111/j.1745-3984.1989.tb00325.x>. [p]
- S. Holm. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6:65–70, 1979. [p]
- J. Kim and T. Oshima. Effect of multiple testing adjustment in differential item functioning detection. *Educational and Psychological Measurement*, 73(3):458–470, 2013. URL <https://doi.org/10.1177/0013164412467033>. [p]
- F. M. Lord. *Applications of item response theory to practical testing problems*. Routledge, New York, NY, first edition, 1980. [p]
- W. Ma and J. de la Torre. GDINA: The generalized DINA model framework, 2019. URL <https://CRAN.R-project.org/package=GDINA>. R package version 2.7. [p]
- D. Magis, S. Beland, F. Tuerlinckx, and P. De Boeck. A general framework and an R package for the detection of dichotomous differential item functioning. *Behavior Research Methods*, 42:847–862, 2010. URL <https://doi.org/10.3758/BRM.42.3.847>. [p]
- N. Mantel and W. Haenszel. Statistical aspects of the analysis of data from retrospective studies. *Journal of the National Cancer Institute*, 22(4):719–748, 1959. URL <https://doi.org/10.1093/jnci/22.4.719>. [p]
- G. L. Marco. Item characteristic curve solutions to three intractable testing problems. *Journal of Educational Measurement*, 14(2):139–160, 1977. URL <https://doi.org/10.1111/j.1745-3984.1977.tb00033.x>. [p]
- P. Martinková, A. Drabinová, Y.-L. Liaw, E. A. Sanders, J. L. McFarland, and R. M. Price. Checking equity: Why differential item functioning analysis should be a routine part of developing conceptual assessments. *CBE-Life Sciences Education*, 16(2):rm2, 2017. URL <https://doi.org/10.1187/cbe.16-10-0307>. [p]
- P. Martinková, A. Hladká, and E. Potužníková. Is academic tracking related to gains in learning competence? Using propensity score matching and differential item change functioning analysis for better understanding of tracking implications. *Learning and Instruction*, 66:101286, 2020. URL <https://doi.org/10.1016/j.learninstruc.2019.101286>. [p]
- P. Martinková and A. Drabinová. ShinyItemAnalysis for teaching psychometrics and to enforce routine analysis of educational tests. *The R Journal*, 10(2):503–515, 2018. URL <https://doi.org/10.32614/RJ-2018-074>. [p]
- P. Martinková and A. Hladká. ShinyItemAnalysis: Test and item analysis via shiny, 2020. URL <https://CRAN.R-project.org/package=ShinyItemAnalysis>. R package version 1.3.3. [p]

- R. D. Penfield and G. Camilli. Differential item functioning and item bias. In C. R. Rao and S. Sinharay, editors, *Psychometrics*, volume 26 of *Handbook of Statistics*, pages 125–167. Elsevier, 2006. URL [https://doi.org/10.1016/S0169-7161\(06\)26005-X](https://doi.org/10.1016/S0169-7161(06)26005-X). [p]
- G. Rasch. *Probabilistic models for some intelligence and attainment tests*. MESA Press, 1993. [p]
- C. Ritz and J. C. Streibig. *Nonlinear regression with R*. Springer, New York, NY, 2008. URL <https://doi.org/10.1007/978-0-387-09616-2>. [p]
- F. Samejima. Estimation of latent ability using a response pattern of graded scores. *Psychometrika*, 34(Suppl 1), 1969. URL <https://doi.org/10.1007/BF03372160>. [p]
- G. Schauberger. *DIFboost: Detection of differential item functioning (DIF) in Rasch models by boosting techniques*, 2016. URL <https://CRAN.R-project.org/package=DIFboost>. R package version 0.2. [p]
- G. Schauberger. *DIFlasso: A penalty approach to differential item functioning in Rasch models*, 2017. URL <https://CRAN.R-project.org/package=DIFlasso>. R package version 1.0-3. [p]
- G. Schwarz et al. Estimating the dimension of a model. *The Annals of Statistics*, 6(2):461–464, 1978. URL <https://doi.org/10.1214/aos/1176344136>. [p]
- C. Strobl, J. Kopf, and A. Zeileis. Rasch trees: A new method for detecting differential item functioning in the Rasch model. *Psychometrika*, 80(2):289–316, 2015. URL <https://doi.org/10.1007/s11336-013-9388-3>. [p]
- H. Swaminathan and H. J. Rogers. Detecting differential item functioning using logistic regression procedures. *Journal of Educational Measurement*, 27(4):361–370, 1990. URL <https://doi.org/10.1111/j.1745-3984.1990.tb00754.x>. [p]
- 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>. [p]
- H. Wickham. *ggplot2: Elegant graphics for data analysis*. Springer-Verlag New York, second edition, 2016. URL <https://ggplot2.tidyverse.org>. [p]
- T. Yee. The VGAM package for categorical data analysis. *Journal of Statistical Software*, 32(10):1–34, 2010. URL <https://doi.org/10.18637/jss.v032.i10>. [p]

Adéla Hladká

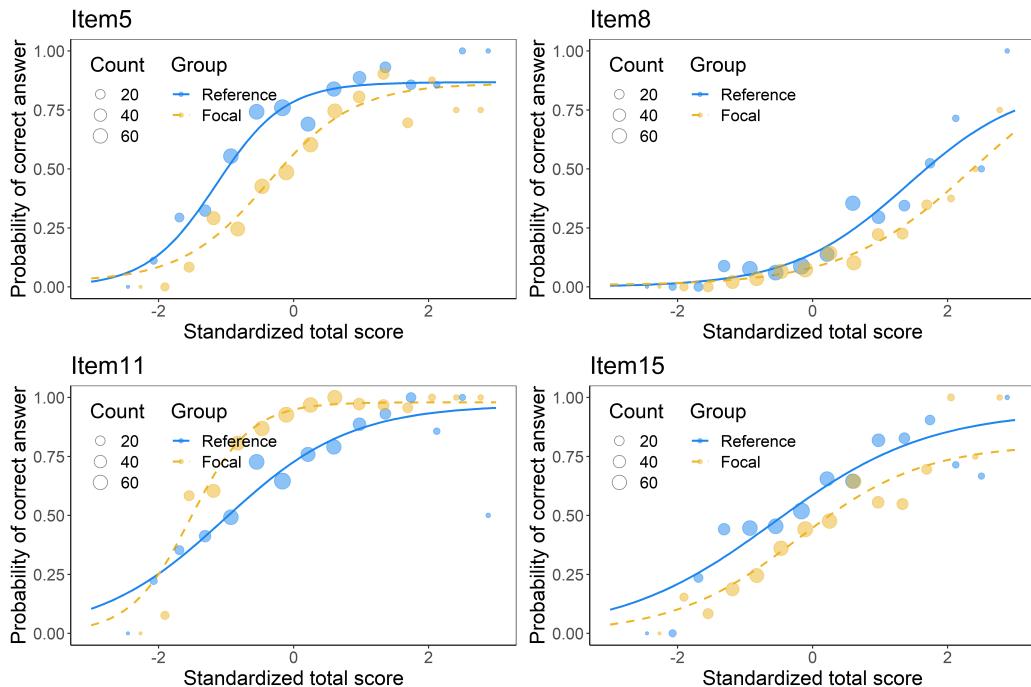
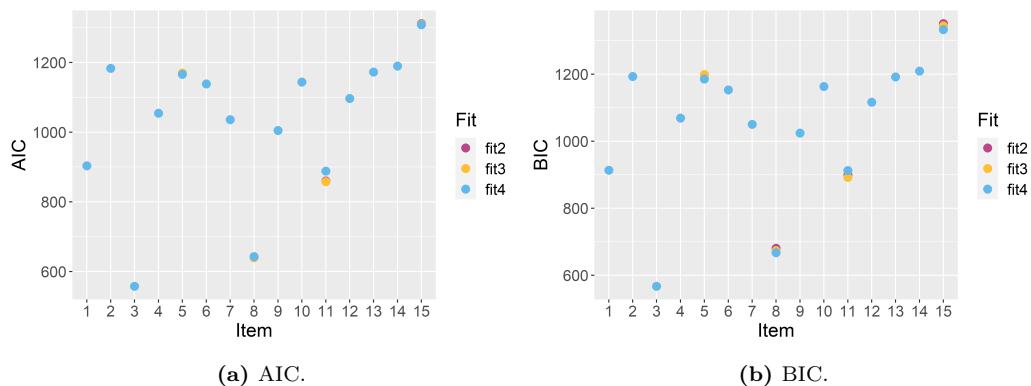
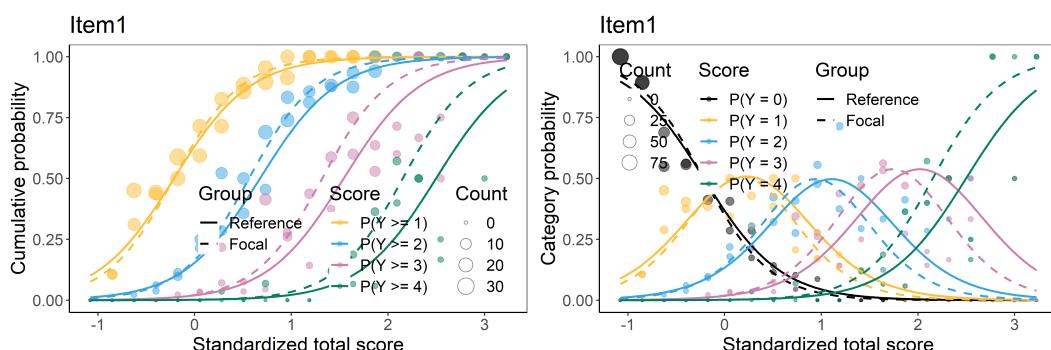
Department of Statistical Modelling, Institute of Computer Science of the Czech Academy of Sciences
Pod Vodárenskou věží 271/2, Prague, 182 07
and

Department of Probability and Mathematical Statistics, Faculty of Mathematics and Physics, Charles University
Sokolovská 83, Prague, 186 75
Czech Republic
ORCID: 0000-0002-9112-1208
hladka@cs.cas.cz

Patrícia Martinková

Department of Statistical Modelling, Institute of Computer Science of the Czech Academy of Sciences
Pod Vodárenskou věží 271/2, Prague, 182 07
and

Institute for Research and Development of Education, Faculty of Education, Charles University
Myslíkova 7, Prague, 110 00
Czech Republic
ORCID: 0000-0003-4754-8543
martinkova@cs.cas.cz

**Figure 1:** Characteristic curves of DIF items.**Figure 2:** Information criteria for item models.**Figure 3:** Cumulative probabilities and characteristic curves of item 1 with a cumulative logit model.

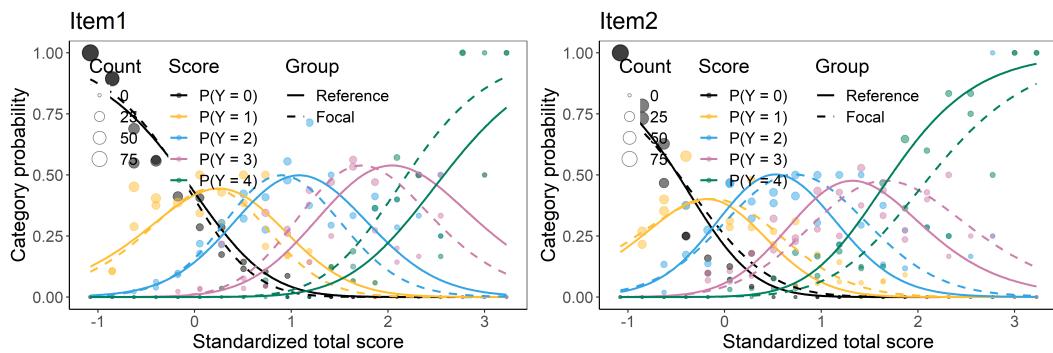


Figure 4: Characteristic curves of DIF items with an adjacent category logit model.

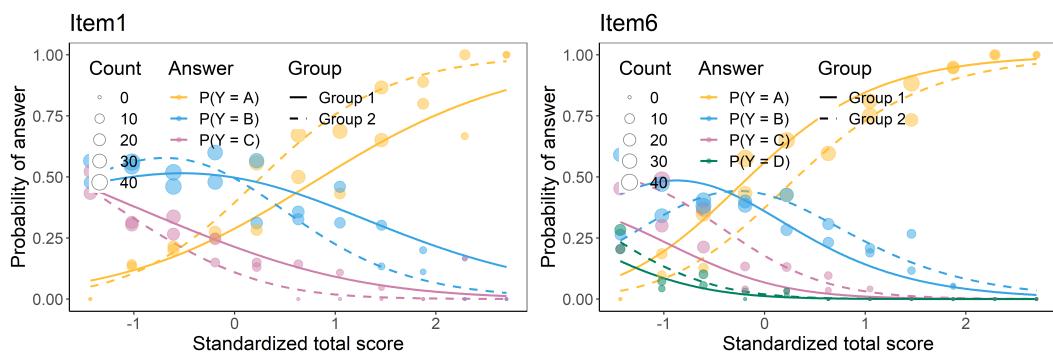


Figure 5: Characteristic curves of DDF items with a multinomial model.

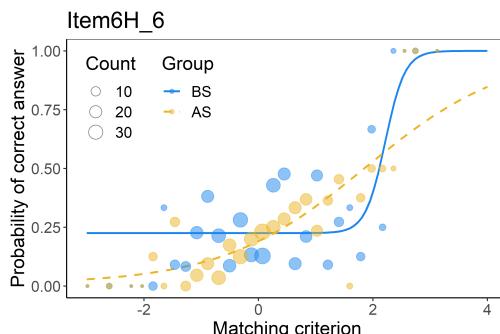


Figure 6: Characteristic curves of item 6H with DIF caused by differences in guessing between tracks in Grade 6.

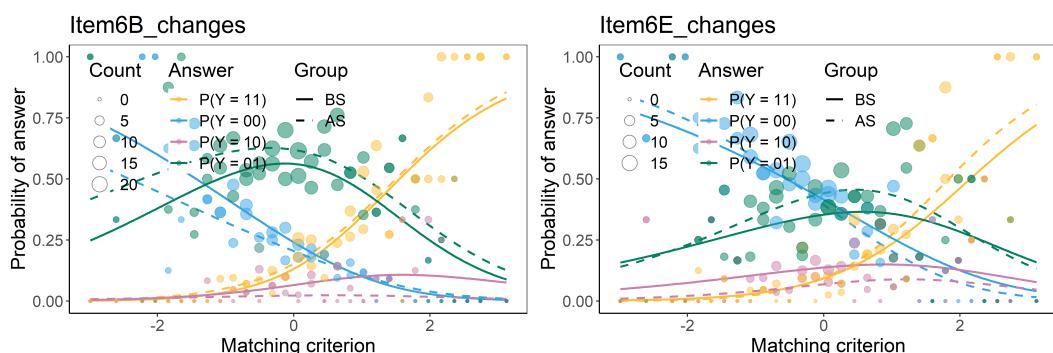


Figure 7: Characteristic curves of DIF-C items 6B and 6E using nominal data and multinomial model.

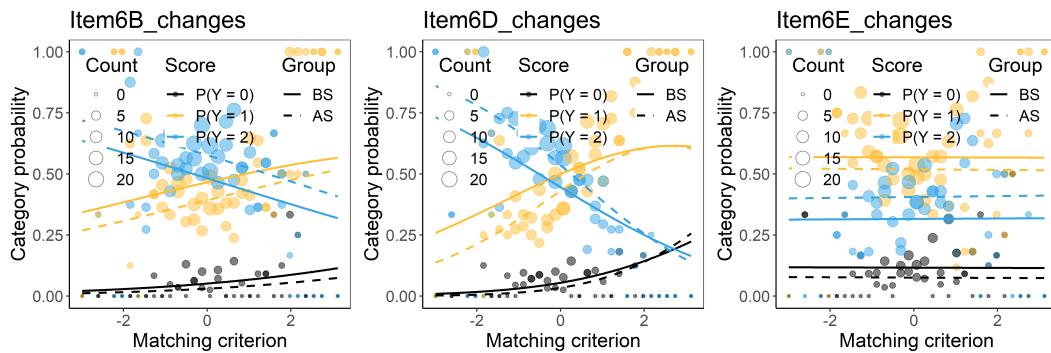


Figure 8: Characteristic curves of DIF-C items 6B, 6D, and 6E using ordinal data and adjacent category logit model.

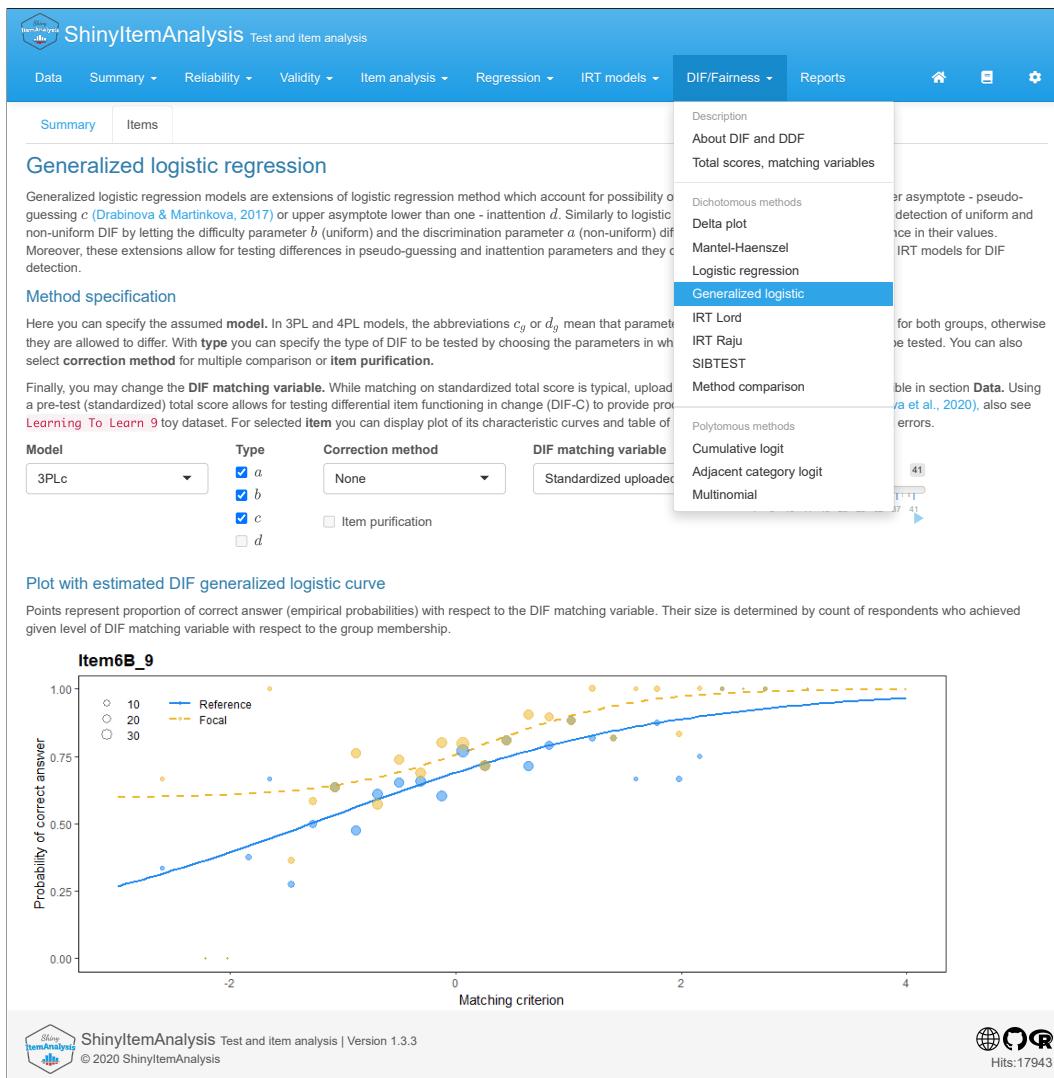


Figure 9: Generalized logistic model for DIF detection using ShinyItemAnalysis.

BayesMallows: An R Package for the Bayesian Mallows Model

by Øystein Sørensen, Marta Crispino, Qinghua Liu, and Valeria Vitelli

Abstract **BayesMallows** is an R package for analyzing preference data in the form of rankings with the Mallows rank model, and its finite mixture extension, in a Bayesian framework. The model is grounded on the idea that the probability density of an observed ranking decreases exponentially with the distance to the location parameter. It is the first Bayesian implementation that allows wide choices of distances, and it works well with a large amount of items to be ranked. **BayesMallows** handles non-standard data: partial rankings and pairwise comparisons, even in cases including non-transitive preference patterns. The Bayesian paradigm allows coherent quantification of posterior uncertainties of estimates of any quantity of interest. These posteriors are fully available to the user, and the package comes with convenient tools for summarizing and visualizing the posterior distributions.

Introduction

Preference data are usually collected when individuals are asked to rank a set of items according to a certain preference criterion. The booming of internet-related activities and applications in recent years has led to a rapid increase of the amount of data that have ranks as their natural scale, however often in the form of partial or indirect rankings (pairwise preferences, ratings, clicks). The amount of readily available software handling preference data has consequently increased consistently in the last decade or so, but not many packages are flexible enough to handle all types of data mentioned above. The typical tasks when analyzing preference data are rank aggregation, classification or clustering, and prediction, where the latter task refers to the estimation of the individual rankings of the assessors when completely or partially missing. These tasks can be addressed either via model-based inference or via heuristic machine learning algorithms, with or without uncertainty quantification. However, very few methods allow handling diverse data types, combining several inferential tasks with proper propagation of the uncertainties, while also providing individualized predictions. Our proposal goes exactly in this direction, thus making the scopes of **BayesMallows** broad when it comes to data handling and individual-level inference.

The R package **BayesMallows** is the first software conceived to answer the needs mentioned above in a unified framework: it implements full Bayesian inference for ranking data, and performs all of the tasks above in the framework of the Bayesian Mallows model (BMM) (Mallows, 1957; Vitelli et al., 2018). More specifically, **BayesMallows** allows for data in the forms of complete rankings, partial rankings, as well as pairwise comparisons, including the case where some comparisons are inconsistent. In these situations, it provides all Bayesian inferential tools for rank modeling with the BMM: it performs rank aggregation (estimation of a consensus ranking of the items), it can cluster together the assessors providing similar preferences (estimating both cluster specific model parameters, and individual cluster assignments, with uncertainty), it performs data augmentation for estimating the latent assessor-specific full ranking of the items in all missing data situations (partial rankings, pairwise preferences). The latter in particular, i.e., the possibility of predicting individual preferences for unranked items, enables the model to be used as a probabilistic recommender system. **BayesMallows** also enlarges the pool of distances that can be used in the Mallows model, and it supports the rank distances most used in the literature: Spearman's footrule (henceforth footrule), Spearman's rank correlation (henceforth Spearman), Cayley, Hamming, Kendall, and Ulam distances (we refer to Diaconis, 1988; Marden, 1995, for details on these). Finally, **BayesMallows** implements the Iterative Proportional Fitting Procedure (IPFP) algorithm for computing the partition function for the Mallows model (MM) (Mukherjee, 2016) and the importance sampling algorithm described in Vitelli et al. (2018). In addition to being used in the MM, these functions may be of interest in their own right.

Comparing with other available inferential software, we notice that not many packages allow for such flexibility, very few in combination with full Bayesian inference, and none when using the MM as outlined in Section 4 below. Often machine learning approaches focus on either rank aggregation (i.e., consensus estimation), or individual rank prediction, while **BayesMallows** handles both. Since the BMM is fully Bayesian, all posterior quantities of interest are automatically available from **BayesMallows** for the first time for the MM. In addition, the package also has tools for visualizing posterior distributions, and hence, posterior quantities as well as their associated uncertainties. Uncertainty quantification is often critical in real applications: for recommender systems, the model should not spam the users with very uncertain recommendations; when performing subtype iden-

tification for cancer patients, a very uncertain cluster assignment might have serious consequences for the clinical practice, for example in treatment choice. The package also works well with a fairly large number of items, thanks to computational approximations and efficient programming. In conclusion, **BayesMallows** provides the first fully probabilistic inferential tool for the MM with many different distances. It is flexible in the types of data it handles, and computationally efficient. We therefore think that this package will gain popularity, and prove its usefulness in many practical situations, many of which we probably cannot foresee now.

The paper is organized as follows. The BMM for ranking data is briefly reviewed in Section 2, as well as its model extensions to different data types and to mixtures. Section 3 includes details on the implementation of the inferential procedure. For a thorough discussion of both the model and its implementation we refer interested readers to Vitelli et al. (2018). An overview of existing R packages implementing the Mallows model (MM) is given in Section 4. The use of the **BayesMallows** package is presented, in the form of three case studies, in Sections 5, 6, and 7. Section 8 concludes the paper, also discussing model extensions that will come with new releases of the package.

Background: the Bayesian Mallows model for rankings

In this section we give the background for understanding the functions in the **BayesMallows** package. More details can be found in Vitelli et al. (2018) and Liu et al. (2019). The section is organized as follows: we first clarify the notations that we will use throughout the paper (Section 2.1). We then briefly describe the BMM for complete ranking data (Section 2.2), also focusing on the relevance of the choice of distance (Section 2.3). The last two sections focus on model extensions: partial and pairwise data (Section 2.4), non-transitive pairwise comparisons (Section 2.5), and mixtures (Section 2.6).

Notation

Let us denote with $\mathcal{A} = \{A_1, \dots, A_n\}$ the finite set of labeled items to be ranked, and with \mathcal{P}_n the space of n -dimensional permutations. A complete ranking of n items is then a mapping $\mathbf{R} : \mathcal{A} \rightarrow \mathcal{P}_n$ that attributes a rank $R_i \in \{1, \dots, n\}$ to each item, according to some criterion. We here denote a generic complete ranking by $\mathbf{R} = (R_1, \dots, R_n)$, where R_i is the rank assigned to item A_i . Given a pair of items $\{A_i, A_k\}$, we denote a pairwise preference between them as $(A_i \prec A_k)$, meaning that item A_i is preferred to item A_k . Note the intimate relation that exists between a ranking and pairwise preferences. Given a full ranking $\mathbf{R} \in \mathcal{P}_n$, it is immediate to evince all the possible $n(n - 1)/2$ pairwise preferences between the items taken in pairs, since the item in the pair having the lower rank is the preferred one:

$$(A_{t_1} \prec A_{t_2}) \iff R_{t_1} < R_{t_2}, \quad t_1, t_2 = 1, \dots, n, \quad t_1 \neq t_2.$$

Conversely, obtaining a full ranking from a set of pairwise preferences is not straightforward. Pairwise preference data are typically incomplete, meaning that not all pairwise preferences necessary to determine each individual ranking are present. They can contain non-transitive patterns, that is, one or more pairwise preferences contradict what is implied by other pairwise preferences. In this package we can handle partial and possibly non-transitive pairwise preferences.

The BMM for Complete Rankings

The MM for ranking data (Mallows, 1957) specifies the probability density for a ranking $\mathbf{r} \in \mathcal{P}_n$ as follows

$$P(\mathbf{r}|\alpha, \rho) = \frac{1}{Z_n(\alpha)} \exp \left[-\frac{\alpha}{n} d(\mathbf{r}, \rho) \right] 1_{\mathcal{P}_n}(\mathbf{r}) \quad (1)$$

where $\rho \in \mathcal{P}_n$ is the location parameter representing the consensus ranking, $\alpha \geq 0$ is the scale parameter (precision), $Z_n(\alpha)$ is the normalizing function (or partition function), $d(\cdot, \cdot)$ is a right-invariant distance among rankings (Diaconis, 1988), and $1_{\mathcal{P}_n}(\mathbf{r})$ is an indicator function for the set \mathcal{P}_n which equals one when $\mathbf{r} \in \mathcal{P}_n$ and zero otherwise.

In the complete data case, N assessors have provided complete rankings of the n items in \mathcal{A} according to some criterion, yielding the permutation $\mathbf{R}_j = (R_{1j}, \dots, R_{nj})$ for assessor j , $j = 1, \dots, N$. The likelihood of the N observed rankings $\mathbf{R}_1, \dots, \mathbf{R}_N$, assumed conditionally

independent given α and ρ , is

$$P(\mathbf{R}_1, \dots, \mathbf{R}_N | \alpha, \rho) = \frac{1}{Z_n(\alpha)^N} \exp \left[-\frac{\alpha}{n} \sum_{j=1}^N d(\mathbf{R}_j, \rho) \right] \prod_{j=1}^N 1_{\mathcal{P}_n}(\mathbf{R}_j). \quad (2)$$

According to the BMM introduced in Vitelli et al. (2018), prior distributions have to be elicited on every parameter of interest. A truncated exponential prior distribution was specified for α

$$\pi(\alpha | \lambda) = \frac{\lambda \exp(-\lambda\alpha) 1_{[0, \alpha_{\max}]}(\alpha)}{1 - \exp(-\lambda\alpha_{\max})}, \quad (3)$$

where λ is a rate parameter, small enough to ensure good prior dispersion, and α_{\max} is a cutoff, large enough to cover reasonable α values. A uniform prior $\pi(\rho)$ on \mathcal{P}_n was assumed for ρ . It follows that the posterior distribution for α and ρ is

$$P(\alpha, \rho | \mathbf{R}_1, \dots, \mathbf{R}_N) \propto \frac{1_{\mathcal{P}_n}(\rho)}{Z_n(\alpha)^N} \exp \left[-\frac{\alpha}{n} \sum_{j=1}^N d(\mathbf{R}_j, \rho) - \lambda\alpha \right] 1_{[0, \alpha_{\max}]}(\alpha). \quad (4)$$

Inference on the model parameters is based on a Metropolis-Hastings (M-H) Markov Chain Monte Carlo (MCMC) algorithm, described in Vitelli et al. (2018). Some details relevant for a correct use of this package are also given in Section 3.1.

Distance measures and partition function

The partition function $Z_n(\alpha)$ in (1), (2) and (4) does not depend on the latent consensus ranking ρ when the distance $d(\cdot, \cdot)$ is right-invariant, meaning that it is unaffected by a relabelling of the items (Diaconis, 1988). Right-invariant distances play an important role in the MM, and **BayesMallows** only handles right-invariant distances. The choice of distance affects the model fit to the data and the results of the analysis, and is crucial also because of its role in the partition function computation. Some right-invariant distances allow for analytical computation of the partition function, and for this reason they have become quite popular. In particular, the MM with Kendall (Meilă and Chen, 2010; Lu and Boutilier, 2014), Hamming (Irurozki et al., 2014) and Cayley (Irurozki et al., 2016b) distances have a closed form of $Z_n(\alpha)$ due to Fligner and Verducci (1986). There are however important and natural right-invariant distances for which the computation of the partition function is NP-hard, in particular the footrule (l_1) and the Spearman (l_2) distances. For precise definitions of all distances involved in the MM we refer to Marden (1995). **BayesMallows** handles the footrule, Spearman, Cayley, Hamming, Kendall, and Ulam distances.

Partial rankings and transitive pairwise comparisons

When complete rankings of all items are not readily available, the BMM can still be used by applying data augmentation techniques. Partial rankings can occur because ranks are missing at random, because the assessors have only ranked their top- k items, or because they have been presented with a subset of items. In more complex situations, data do not include ranks at all, but the assessors have only compared pairs of items and given a preference between the two. The Bayesian data augmentation scheme can still be used to handle such pairwise comparison data, thus providing a solution that is fully integrated into the Bayesian inferential framework. The following paragraphs provide a summary of Sections 4.1 and 4.2 of Vitelli et al. (2018), which we refer to for details.

Let us start by considering the case of top- k rankings. Suppose that each assessor j has chosen a set of preferred items $\mathcal{A}_j \subseteq \mathcal{A}$, which were given ranks from 1 to $n_j = |\mathcal{A}_j|$. Now $R_{ij} \in \{1, \dots, n_j\}$ if $A_i \in \mathcal{A}_j$, while for $A_i \in \mathcal{A}_j^c$, R_{ij} is unknown, except for the constraint $R_{ij} > n_j$, $j = 1, \dots, N$. The augmented data vectors $\tilde{\mathbf{R}}_1, \dots, \tilde{\mathbf{R}}_N$ are introduced in the model to include the missing ranks, which are estimated as latent parameters. Let $\mathcal{S}_j = \{\tilde{\mathbf{R}}_j \in \mathcal{P}_n : \tilde{R}_{ij} = R_{ij} \text{ if } A_i \in \mathcal{A}_j\}$, $j = 1, \dots, N$ be the set of possible augmented random vectors, including the ranks of the observed top- n_j items together with the unobserved ranks, which are assigned a uniform prior on the permutations of $\{n_j + 1, \dots, n\}$. The goal is to sample from the posterior distribution

$$P(\alpha, \rho | \mathbf{R}_1, \dots, \mathbf{R}_N) = \sum_{\tilde{\mathbf{R}}_1 \in \mathcal{S}_1} \dots \sum_{\tilde{\mathbf{R}}_N \in \mathcal{S}_N} P(\alpha, \rho, \tilde{\mathbf{R}}_1, \dots, \tilde{\mathbf{R}}_N | \mathbf{R}_1, \dots, \mathbf{R}_N). \quad (5)$$

The augmentation scheme amounts to alternating between sampling α and ρ given the current values

of the augmented ranks using the posterior given in (4), and sampling the augmented ranks given the current values of α and ρ . For the latter task, once α , ρ and the observed ranks $\mathbf{R}_1, \dots, \mathbf{R}_N$ are fixed, one can see that $\tilde{\mathbf{R}}_1, \dots, \tilde{\mathbf{R}}_N$ are conditionally independent, and that each $\tilde{\mathbf{R}}_j$ only depends on the corresponding \mathbf{R}_j . As a consequence, the update of new augmented vectors is performed independently, for each $j = 1, \dots, N$.

The above procedure can also handle more general situations where missing rankings are not necessarily the bottom ones, and where each assessor is asked to provide the mutual ranking of some possibly random subset $\mathcal{A}_j \subseteq \mathcal{A}$ consisting of $n_j \leq n$ items. Note that the only difference from the previous formulation is that each latent rank vector $\tilde{\mathbf{R}}_j$ takes values in the set

$$\mathcal{S}_j = \left\{ \tilde{\mathbf{R}}_j \in \mathcal{P}_n : (R_{i_1j} < R_{i_2j}) \wedge (A_{i_1}, A_{i_2} \in \mathcal{A}_j) \Rightarrow \tilde{R}_{i_1j} < \tilde{R}_{i_2j} \right\}.$$

Also in this case the prior for $\tilde{\mathbf{R}}_j$ is assumed uniform on \mathcal{S}_j .

In the case of pairwise comparison data, let us call \mathcal{B}_j the set of all pairwise preferences stated by assessor j , and let \mathcal{A}_j be the set of items appearing at least once in \mathcal{B}_j . Note that the items in \mathcal{A}_j are not necessarily fixed to a given rank but may only be given some partial ordering. For the time being, we assume that the observed pairwise orderings in \mathcal{B}_j are transitive, i.e., mutually compatible, and define by $\text{tc}(\mathcal{B}_j)$ the transitive closure of \mathcal{B}_j , which contains all pairwise orderings of the elements in \mathcal{A}_j induced by \mathcal{B}_j . The model formulation remains the same as in the case of partial rankings, with the prior for the augmented data vectors $\tilde{\mathbf{R}}_1, \dots, \tilde{\mathbf{R}}_N$ being uniform on the set \mathcal{S}_j of rankings that are compatible with the observed data.

Non-transitive pairwise comparisons

It can happen in real applications that individual pairwise comparison data are non-transitive, that is, they may contain a pattern of the form $x \prec y$, $y \prec z$ but $z \prec x$. This is typically the case of data collected from internet user activities, when the pool of items is very large: non-transitive patterns can arise for instance due to assessors' inattentiveness, uncertainty in their preferences, and actual confusion, even when one specific criterion for ranking is used. Another frequent situation is when the number of items is not very large, but the items are perceived as very similar by the assessors. This setting is discussed in Crispino et al. (2019), where the model for transitive pairwise comparisons of Section 2.4 is generalized to handle situations where non-transitivities in the data occur. Note that the kind of non-transitivity that is considered in Crispino et al. (2019) considers only the individual level preferences. A different type of non-transitivity, which we do not consider here, arises when aggregating preferences across assessors, as under Condorcet (Marquis of Condorcet, 1785) or Borda (de Borda, 1781) voting rules.

The key ingredient of this generalization consists of adding one layer of latent variables to the model hierarchy, accounting for the fact that assessors can make mistakes. The main assumption is that the assessor makes pairwise comparisons based on her latent full rankings $\tilde{\mathbf{R}}$. A mistake is defined as an inconsistency between one of the assessor's pairwise comparisons and $\tilde{\mathbf{R}}$. Suppose each assessor $j = 1, \dots, N$ has assessed M_j pairwise comparisons, collected in the set \mathcal{B}_j , and assume the existence of latent ranking vectors $\tilde{\mathbf{R}}_j$, $j = 1, \dots, N$. Differently from Section 2.4, since \mathcal{B}_j is allowed to contain non-transitive pairwise preferences, the transitive closure of \mathcal{B}_j is not defined, and the posterior density (5) cannot be evaluated. In this case, the posterior takes the form,

$$\begin{aligned} P(\alpha, \rho | \mathcal{B}_1, \dots, \mathcal{B}_N) &= \sum_{\tilde{\mathbf{R}}_1 \in \mathcal{P}_n} \dots \sum_{\tilde{\mathbf{R}}_N \in \mathcal{P}_n} P(\alpha, \rho, \tilde{\mathbf{R}}_1, \dots, \tilde{\mathbf{R}}_N | \mathcal{B}_1, \dots, \mathcal{B}_N) = \\ &= \sum_{\tilde{\mathbf{R}}_1 \in \mathcal{P}_n} \dots \sum_{\tilde{\mathbf{R}}_N \in \mathcal{P}_n} P(\alpha, \rho | \tilde{\mathbf{R}}_1, \dots, \tilde{\mathbf{R}}_N) P(\tilde{\mathbf{R}}_1, \dots, \tilde{\mathbf{R}}_N | \mathcal{B}_1, \dots, \mathcal{B}_N) \end{aligned} \quad (6)$$

where the term $P(\tilde{\mathbf{R}}_1, \dots, \tilde{\mathbf{R}}_N | \mathcal{B}_1, \dots, \mathcal{B}_N)$ models the presence of mistakes in the data, while in the case of transitive pair comparisons it was implicitly assumed equal to 1 if each augmented ranking $\tilde{\mathbf{R}}_j$ was compatible with the partial information contained in \mathcal{B}_j , and 0 otherwise.

Two models for (6) are considered in Crispino et al. (2019): the Bernoulli model, which accounts for random mistakes, and the Logistic model, which lets the probability of making a mistake depend on the similarity of the items being compared. The Bernoulli model states that:

$$P(\tilde{\mathbf{R}}_1, \dots, \tilde{\mathbf{R}}_N | \theta, \mathcal{B}_1, \dots, \mathcal{B}_N) \propto \theta^M (1 - \theta)^{\sum_j M_j - M}, \quad \theta \in [0, 0.5] \quad (7)$$

where M counts the number of times the observed preferences contradict what is implied by the ranking $\tilde{\mathbf{R}}_j$, M_j is the number of pairwise comparisons reported by assessor j , and the parameter θ is the probability of making a mistake in a single pairwise preference. θ is *a priori* assigned a truncated

Beta distribution on the interval $[0, 0.5]$ with given hyper-parameters κ_1 and κ_2 , conjugate to the Bernoulli model (7). The Logistic model is a generalization of (7) where, instead of assigning a constant value θ to the probability of making a mistake, it depends on the distance between the ranks of the two items under comparison. In Crispino et al. (2019) the Logistic model gave results very similar to the Bernoulli model, and currently only the Bernoulli model is available in **BayesMallows**. The sampling scheme is similar to the one used for the case of transitive pairwise preferences, apart from an additional step for updating θ , and the augmentation scheme for $\tilde{\mathbf{R}}_j$, which is slightly different. We refer to Crispino et al. (2019) for details.

Clustering

The assumption, implicit in the discussion so far, that there exists a unique consensus ranking shared by all assessors is unrealistic in most real applications: the BMM thus handles the case where the rankings provided by all assessors can be modeled as a finite mixture of MMs. In the following brief discussion we assume that the data consist of complete rankings, but **BayesMallows** can fit a mixture based on any kinds of preference data described so far. See Section 4.3 of Vitelli et al. (2018) for details.

Let $z_1, \dots, z_N \in \{1, \dots, C\}$ assign each assessor to one of C clusters, and let the rankings within each cluster $c \in \{1, \dots, C\}$ be described by an MM with parameters α_c and ρ_c . The likelihood of the observed rankings $\mathbf{R}_1, \dots, \mathbf{R}_N$ is given by

$$P\left(\mathbf{R}_1, \dots, \mathbf{R}_N \mid \{\rho_c, \alpha_c\}_{c=1, \dots, C}, \{z_j\}_{j=1, \dots, N}\right) = \prod_{j=1}^N \frac{1_{\mathcal{P}_n}(\mathbf{R}_j)}{Z_n(\alpha_{z_j})} \exp\left[-\frac{\alpha_{z_j}}{n} d(\mathbf{R}_j, \rho_{z_j})\right],$$

where conditional independence is assumed across the clusters. We also assume independent truncated exponential priors for the scale parameters and independent uniform priors for the consensus rankings. The cluster labels z_1, \dots, z_N are a priori assumed conditionally independent given the clusters mixing parameters τ_1, \dots, τ_C , and are assigned a uniform multinomial. Finally τ_1, \dots, τ_C (with $\tau_c \geq 0$, $c = 1, \dots, C$ and $\sum_{c=1}^C \tau_c = 1$) are assigned the standard symmetric Dirichlet prior of parameter Ψ , thus implying a conjugate scheme. The posterior density is then given by

$$P\left(\{\rho_c, \alpha_c, \tau_c\}_{c=1}^C, \{z_j\}_{j=1}^N \mid \mathbf{R}_1, \dots, \mathbf{R}_N\right) \propto \left[\prod_{c=1}^C e^{-\lambda \alpha_c \tau_c^{\Psi-1}} \right] \left[\prod_{j=1}^N \frac{\tau_{z_j} e^{-\frac{\alpha_{z_j}}{n} d(\mathbf{R}_j, \rho_{z_j})}}{Z_n(\alpha_{z_j})} \right]. \quad (8)$$

Computational considerations

In this section we briefly give some additional details regarding the implementation of the models described in Section 2. The BMM implementation is thoroughly described in Vitelli et al. (2018).

Details on the MCMC procedures

In order to obtain samples from the posterior density of equation (4), **BayesMallows** implements an MCMC scheme iterating between (i) updating ρ and (ii) updating α (Algorithm 1 of Vitelli et al., 2018). The leap-and-shift proposal distribution, which is basically a random local perturbation of width L of a given ranking, is used for updating ρ in step (i). The L parameter of the leap-and-shift proposal controls how far the proposed ranking is from the current one, and it is therefore linked to the acceptance rate. The recommendation given in Vitelli et al. (2018) is to set it to $L = n/5$, which is also the default value in **BayesMallows**, but the user is allowed to choose a different value. For updating α in step (ii), a log-normal density is used as proposal, and its variance σ_α^2 can be tuned to obtain a desired acceptance rate.

As mentioned in Section 2.4, the MCMC procedure for sampling from the posterior densities corresponding to the partial data cases (Algorithm 3 of Vitelli et al., 2018) has an additional M-H step to account for the update of the augmented data rankings $\{\tilde{\mathbf{R}}_j\}_{j=1}^N$. In the case of partial rankings, for updating the augmented data $\tilde{\mathbf{R}}_j$, $j = 1, \dots, N$ we use a uniform proposal on the set of rankings compatible with the partial data, \mathcal{S}_j . In the case of pairwise preferences, due to the increased sparsity in the data, we instead implemented a modified parameter-free leap-and-shift proposal distribution, which proposes a new augmented ranking by locally permuting the ranks in $\tilde{\mathbf{R}}_j$ within the constraints given by \mathcal{B}_j (Vitelli et al., 2018, Section 4.2). The generalization to non-transitive pairwise comparisons, outlined in Section 4 of Crispino et al. (2019), requires

further considerations. First, in the M-H step for updating the augmented data rankings, the modified parameter-free leap-and-shift proposal has to be replaced by a Swap proposal, whose tuning parameter L^* is the maximum allowed distance between the ranks of the swapped items. Second, the Bernoulli model for mistakes makes it necessary to add a Gibbs step for the update of θ . The MCMC algorithm for sampling from the mixture model posterior (8) (Algorithm 2 of Vitelli et al., 2018) alternates between updating $\{\rho_c, \alpha_c\}_{c=1}^C$ in an M-H step, and $\{\tau_c, z_j\}_{c=1, j=1}^{C, N}$ in a Gibbs sampling step, in addition to the necessary M-H steps for data augmentation or estimation of error models, as outlined above.

Partition Function

When the distance in the BMM is footrule or Spearman, the partition function $Z_n(\cdot)$ does not have a closed form. In these situations **BayesMallows** allows for three different choices, which the user may employ depending on the value of n : (a) exact calculation, (b) Importance Sampling (IS), and (c) the asymptotic approximation due to Mukherjee (2016).

The package contains integer sequences for exact calculation of the partition function with footrule distance for up to $n = 50$ items, and with the Spearman distance for up to $n = 14$ items (see Vitelli et al., 2018, Section 2.1). These sequences are downloaded from the On-Line Encyclopedia of Integer Sequences (Sloane, 2017).

The IS procedure can be used to compute an off-line approximation $\hat{Z}_n(\alpha)$ of $Z_n(\alpha)$ for the specific value of n which is needed in the application at hand. The IS estimate $\hat{Z}_n(\alpha)$ is computed on a grid of α values provided by the user, and then a smooth estimate obtained via a polynomial fit is returned to the user, who can also select the degree of the polynomial function. Finally, the user should set the number K of IS iterations, and we refer to Vitelli et al. (2018) for guidelines on how to select a large enough value for K . The procedure might be time-consuming, depending on K , n , and on how the grid for α is specified. In our experience, values of n larger than approximately 100 might require K to be as large as 10^8 in order for the IS to provide a good estimate, and hence a long computing time.

The IPFP algorithm (Mukherjee, 2016, Theorem 1.8) yields a numeric evaluation of $Z_{\lim}(\cdot)$, the asymptotic approximation to $Z_n(\cdot)$. In this case the user needs to specify two parameters: the number of iterations m to use in the IPFP, and the number of grid points K of the grid approximating the continuous domain where the limit is computed. Values of m and K have been suggested by Mukherjee (2016), and we refer to the Supplementary Material of Vitelli et al. (2018) for more details.

A simulation experiment was conducted comparing the methods for estimating $\log(Z(\alpha))$ with footrule distance for $\alpha = 0.1, 0.2, \dots, 20$. Let $\log(Z(\alpha))^K$ denote the IS estimate obtained with K iterations, and define the absolute relative difference between two IS estimates obtained with K_2 and K_1 iterations as

$$\rho(K_2, K_1) = \max_{\alpha} \left\{ \frac{|\log(Z(\alpha))^{K_2} - \log(Z(\alpha))^{K_1}|}{|\log(Z(\alpha))^{K_1}|} \right\}.$$

The IS algorithm was run with 10^5 , 10^6 , and 10^7 iterations, and we obtained $\rho(10^6, 10^5) < 0.3\%$ and $\rho(10^7, 10^6) < 0.15\%$, suggesting that using $K = 10^7$ yields low Monte Carlo variation over this range of α . The IPFP algorithm was used to estimate $\log(Z_{\lim}(\alpha))$, with $K = 10^3$ and $m = 10^3$. The results are summarized in Figure 1, in which also exact computation is included in the case of 50 items. With 50 items, the IS estimate perfectly overlaps the exact estimate, while the asymptotic estimate has a bias that increases with increasing α . As expected, the bias of the asymptotic estimate decreases when the number of items increases, as this brings it closer to the asymptotic limit.

Sampling from the Bayesian Mallows Model

To obtain random samples from the MM with Cayley, Hamming, Kendall, or Ulam distance and fixed α and ρ , we suggest using the **PerMallows** (Irurzoki et al., 2016a) package, which is optimized for this task. We instead provide a procedure for sampling from the MM with footrule and Spearman. The procedure to generate a random sample of size N from the MM is straightforward, and described in Appendix C of Vitelli et al. (2018). Basically we run a Metropolis-Hastings algorithm with fixed ρ and α and accept/reject the sample based on its acceptance probability. We then take N rankings with a large enough interval between each of them to achieve independence.

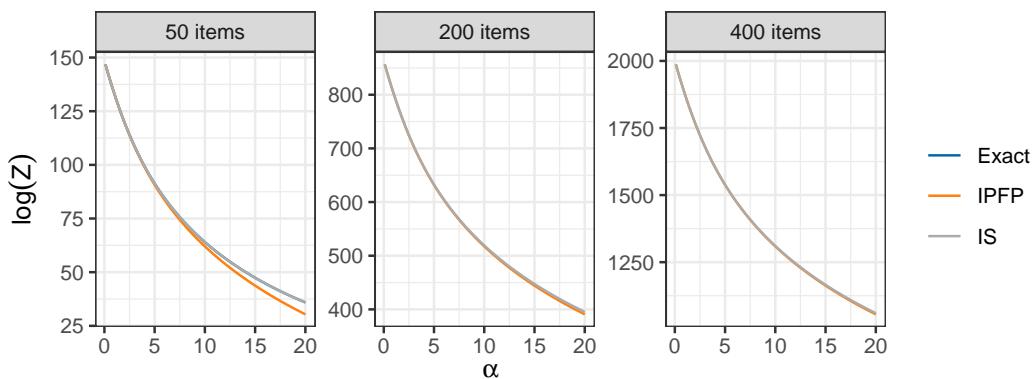


Figure 1: Estimates of the log partition function for the BMM with footrule distance computed using exact calculation (only in the case of 50 items), asymptotic estimation with the IPFP algorithm, and Monte Carlo simulation with the IS algorithm. Exact and IS estimates are perfectly overlapping in the case of 50 items. The bias of the IPFP estimates decreases when the number of items increases.

Packages implementing the Mallows model

This section gives an overview of existing packages for fitting the MM.

- **PerMallows** is the package that comes closest in functionality to **BayesMallows**. It contains functions for learning and sampling from the frequentist versions of the MM and generalized Mallows model (GMM) (Fligner and Verducci, 1986). Compared to **BayesMallows**, it lacks support for footrule or Spearman distance, it is not Bayesian, and does not compute uncertainty ranges for the estimated parameters. In addition **PerMallows** handles only complete rankings, and does not provide functionality for computation of mixture models. According to Irurozki et al. (2016a, Table 1), computing the maximum likelihood estimates (MLE) of α and ρ using the function `lmm` is possible when $n < 80$ for Kendall, $n < 250$ for Cayley, $n < 90$ for Hamming and $n < 100$ for Ulam. Our experiments suggest that these estimates are conservative, and that even larger numbers of items are fit rapidly. Hence, **PerMallows** seems to be a good choice for modeling with complete data without clusters, when the supported distance measures are appropriate and uncertainty estimates are not sought. **PerMallows** also has very efficient functions for sampling from the MM with Cayley, Hamming, Kendall, and Ulam distances.
- **pmr** (Lee and Yu, 2013) provides summary statistics, visualization, and model fitting tools for complete ranking data in the MM, as well as other models. The function `dbm` returns the MLE of α together with its variance. The MLE of ρ , however, is not returned, but printed to the console, and no uncertainty estimates are given. Internally, `dbm` generates a matrix of size $n! \times n$ containing all possible permutations of the n items. As a result, it quickly runs into memory issues. In our tests, **pmr** was not able to handle a ranking dataset with $n = 10$ items.
- **rankdist** (Qian, 2018) implements distance-based probability models for ranking data as described in Alvo and Yu (2014), returning MLEs for α and ρ , but no uncertainty estimates. The package handles a large number of distances and supports mixture models, but in our experiments a warning was issued when using mixtures with all distances except Kendall. **rankdist** also implements the GMM (Fligner and Verducci, 1986). However, for Cayley, footrule, Hamming, and Spearman distances, it generates an $n! \times n$ matrix internally, causing our R session to crash with $n \geq 10$ items, hence limiting its applicability. For Kendall, on the other hand, **rankdist** appears to work fine both with a large number of items, and with mixtures.

BayesMallows provides many new functionalities not implemented in these packages, as will be illustrated in the use cases of the following three sections.

Analysis of complete rankings with BayesMallows

We illustrate the case of complete rankings with the potato datasets described in Liu et al. (2019, Section 4). In short, a bag of 20 potatoes was bought, and 12 assessors were asked to rank the potatoes by weight, first by visual inspection, and next by holding the potatoes in hand. These datasets

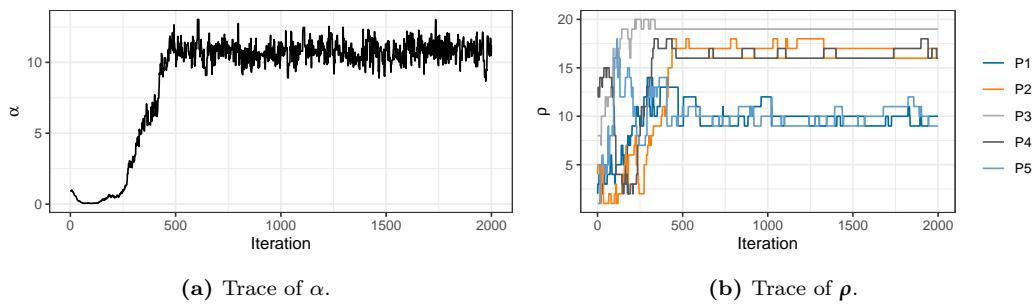


Figure 2: Trace plots of α and ρ for the MCMC algorithm with the `potato_visual` dataset. The plots indicating good mixing for both parameters after about 500 iterations.

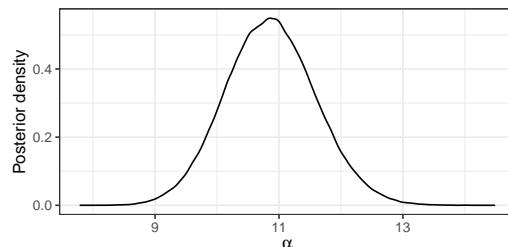


Figure 3: Posterior distribution of α with the potato_visual dataset. The posterior mass is symmetrically centered between 9 and 13, with a mean around 11.

are available in **BayesMallows** as matrices with names `potato_weighing` and `potato_visual`, respectively. The true ranking of the potatoes' weights is available in the vector `potato_true_ranking`. In general, `compute_mallows` expects ranking datasets to have one row for each assessor and one column for each item. Each row has to be a proper permutation, possibly with missing values. We are interested in the posterior distribution of both the level of agreement between assessors, as described by α , and in the latent ranking of the potatoes, as described by ρ . We refer to the attached replication script for random number seeds for exact reproducibility.

First, we do a test run to check convergence of the MCMC algorithm, and then get trace plots with `assess convergence`.

```
bmm_test <- compute_mallows(potato_visual)
assess_convergence(bmm_test)
```

By default, `assess_convergence` returns a trace plot for α , shown in Figure 2a. The algorithm seems to be mixing well after around 500 iterations. Next, we study the convergence of ρ . To avoid overly complex plots, we pick potatoes 1–5 by specifying this in the `items` argument.

```
assess convergence(bmm test, parameter = "rho", items = 1:5)
```

The corresponding plot is shown in Figure 2b, illustrating that the MCMC algorithm seems to have converged after around 1,000 iterations.

From the trace plots, we decide to discard the first 1,000 MCMC samples as burn-in. We rerun the algorithm to get 500,000 samples after burn-in. The object `bmm_visual` has S3 class "BayesMallows", so we plot the posterior distribution of α with `plot.BayesMallows`.

```
bmm_visual <- compute_mallows(potato_visual, nmc = 501000)
bmm_visual$burnin <- 1000 # Set burn-in to 1000
plot(bmm_visual) # Use S3 method for plotting
```

The plot is shown in Figure 3. We can also get posterior credible intervals for α using `compute_posterior_intervals`, which returns both highest posterior density intervals (HPDI) and central intervals in a `tibble` (Müller and Wickham, 2018). `BayesMallows` uses `tibbles` rather than `data.frames`, but both are accepted as function inputs. We refer to `tibbles` as dataframes henceforth.

```
compute_posterior_intervals(bmm_visual, decimals = 1L)

# A tibble: 1 x 6
```

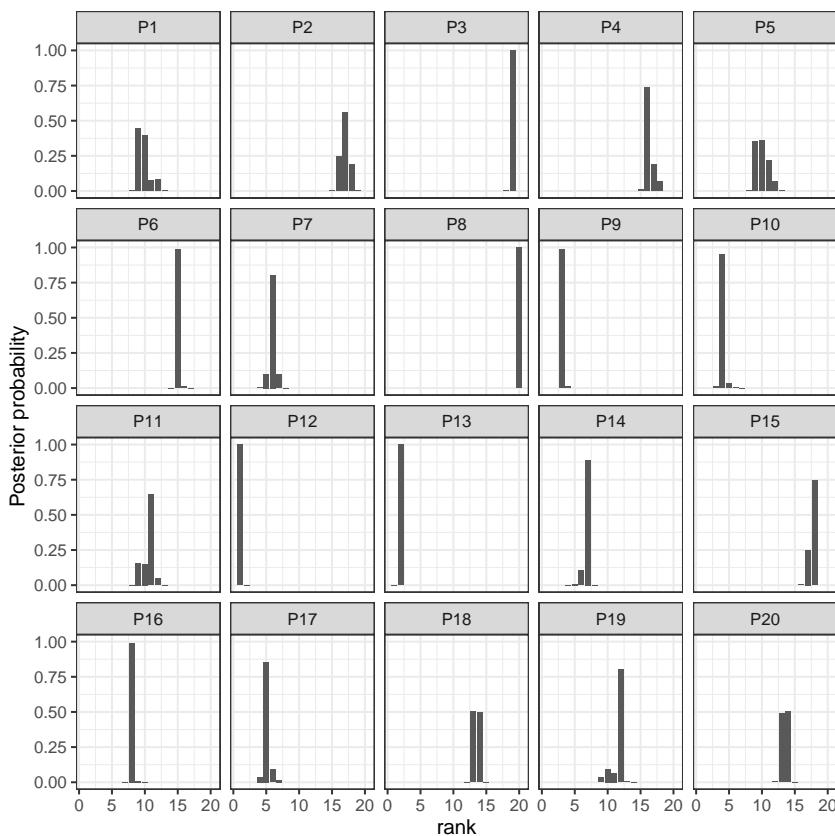


Figure 4: Posterior distribution of latent ranks ρ with the `potato_visual` dataset. Most potatoes have highly peaked posterior distributions, indicating low uncertainty about their ranking.

```
parameter mean median conf_level hpd1      central_interval
<ch   <dbl> <dbl> <ch   <ch   <chr>
1 alpha      10.9  10.9 95 %      [9.4,12.3] [9.5,12.3]
```

Next, we can go on to study the posterior distribution of ρ .

```
plot(bmm_visual, parameter = "rho", items = 1:20)
```

If the `items` argument is not provided, and the number of items exceeds five, five items are picked at random for plotting. To show all potatoes, we explicitly set `items = 1:20`. The corresponding plots are shown in Figure 4.

Jumping over the scale parameter

Updating α in every step of the MCMC algorithm may not be necessary, as the number of posterior samples typically is more than large enough to obtain good estimates of its posterior distribution. With the `alpha_jump` argument, we can tell the MCMC algorithm to update α only every `alpha_jump`-th iteration. To update α every 10th update of ρ , we do

```
bmm <- compute_mallows(potato_visual, nmc = 501000, alpha_jump = 10)
```

On a MacBook Pro 2.2 GHz Intel Core i7 running R version 3.5.1, the above call ran in 2.0 seconds on average over 1,000 replications using `microbenchmark` (Mersmann, 2018), while it took 4.2 seconds using the default value `alpha_jump = 1`, i.e., updating α less frequently more than halved the computing time.

Other distance metrics

By default, `compute_mallows` uses the footrule distance, but the user can also choose to use Cayley, Kendall, Hamming, Spearman, or Ulam distance. Running the same analysis of the potato data with Spearman distance is done with the command

```
bmm <- compute_mallows(potato_visual, metric = "spearman", nmc = 501000)
```

For the particular case of Spearman distance, **BayesMallows** only has integer sequences for computing the exact partition function with 14 or fewer items. In this case a precomputed importance sampling estimate is part of the package, and used instead.

Analysis of preference data with BayesMallows

Unless the argument `error_model` to `compute_mallows` is set, pairwise preference data are assumed to be consistent within each assessor. These data should be provided in a dataframe with the following three columns, with one row per pairwise comparison.

- `assessor` is an identifier for the assessor; either a numeric vector containing the assessor index, or a character vector containing the unique name of the assessor.
- `bottom_item` is a numeric vector containing the index of the item that was disfavored in each pairwise comparison.
- `top_item` is a numeric vector containing the index of the item that was preferred in each pairwise comparison.

A dataframe with this structure can be given in the `preferences` argument to `compute_mallows`. `compute_mallows` will generate the full set of implied rankings for each assessor using the function `generate_transitive_closure`, as well as an initial ranking matrix consistent with the pairwise preferences, using the function `generate_initial_ranking`.

We illustrate with the beach preference data containing stated pairwise preferences between random subsets of 15 images of beaches, by 60 assessors (Vitelli et al., 2018, Section 6.2). This dataset is provided in the dataframe `beach_preferences`.

Transitive closure and initial ranking

We start by generating the transitive closure implied by the pairwise preferences.

```
beach_tc <- generate_transitive_closure(beach_preferences)
```

The dataframe `beach_tc` contains all the pairwise preferences in `beach_preferences`, with all the implied pairwise preferences in addition. The latter are preferences that were not specifically stated by the assessor, but instead are implied by the stated preferences. As a consequence, the dataframe `beach_tc` has 2921 rows, while `beach_preferences` has 1442 rows. Initial rankings, i.e., a set of full rankings for each assessor that are consistent with the implied pairwise preferences are then generated, and we set the column names of the initial ranking matrix to "Beach 1", "Beach 2", ..., "Beach 15" in order have these names appear as labels in plots and output.

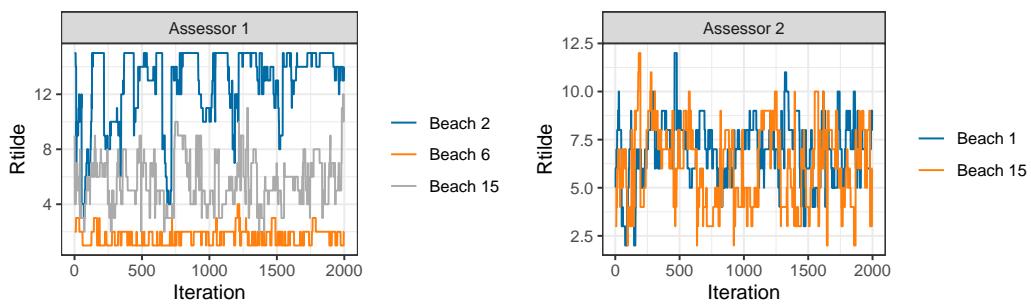
```
beach_init_rank <- generate_initial_ranking(beach_tc)
colnames(beach_init_rank) <- paste("Beach", 1:ncol(beach_init_rank))
```

If we had not generated the transitive closure and the initial ranking, `compute_mallows` would do this for us, but when calling `compute_mallows` repeatedly, it may save time to have these precomputed and saved for future re-use. In order to save time in the case of big datasets, the functions for generating transitive closures and initial rankings from transitive closures can all be run in parallel, as shown in the examples to the `compute_mallows` function. The key to the parallelization is that each assessor's preferences can be handled independently of the others, and this can speed up the process considerably with large dataset.

As an example, let us look at all preferences stated by assessor 1 involving beach 2. We use `filter` from `dplyr` (Wickham et al., 2018) to obtain the right set of rows.

```
library("dplyr")
# All preferences stated by assessor 1 involving item 2
filter(beach_preferences, assessor == 1, bottom_item == 2 | top_item == 2)

# A tibble: 1 x 3
  assessor bottom_item top_item
    <dbl>      <dbl>     <dbl>
1       1         2        15
```



(a) Beaches 2, 6, and 15 for assessor 1. The ordering of the traces is always consistent with the constraints between beaches 1 and 15 are implied by assessor 1's set by assessor 1's preferences.

(b) Beaches 1 and 15 for assessor 2. No orderings of preferences, and the traces are hence free to cross.

Figure 5: Trace plots of augmented ranks \tilde{R}

Assessor 1 has performed only one direct comparison involving beach 2, in which the assessor stated that beach 15 is preferred to beach 2. The implied orderings, on the other hand, contain two preferences involving beach 2:

```
# All implied orderings for assessor 1 involving item 2
filter(beach_tc, assessor == 1, bottom_item == 2 | top_item == 2)

assessor bottom_item top_item
1      1           2       6
2      1           2      15
```

In addition to the statement that beach 15 is preferred to beach 2, all the other orderings stated by assessor 1 imply that this assessor prefers beach 6 to beach 2.

Convergence diagnostics

As with the potato data, we can do a test run to assess the convergence of the MCMC algorithm. However, this time we provide the initial rankings `beach_init_rank` to the `rankings` argument and the transitive closure `beach_tc` to the `preferences` argument of `compute_mallows`. We also set `save_aug = TRUE` to save the augmented rankings in each MCMC step, hence letting us assess the convergence of the augmented rankings.

```
bmm_test <- compute_mallows(rankings = beach_init_rank,
                               preferences = beach_tc, save_aug = TRUE)
```

Running `assess_convergence` for α and ρ shows good convergence after 1000 iterations (not shown). To check the convergence of the data augmentation scheme, we need to set `parameter = "Rtilde"`, and also specify which items and assessors to plot. Let us start by considering items 2, 6, and 15 for assessor 1, which we studied above.

```
assess_convergence(bmm_test, parameter = "Rtilde",
                   items = c(2, 6, 15), assessors = 1)
```

The resulting plot is shown in Figure 5a. It illustrates how the augmented rankings vary, while also obeying their implied ordering.

By further investigation of `beach_tc`, we would find that no orderings are implied between beach 1 and beach 15 for assessor 2. With the following command, we create trace plots to confirm this:

```
assess_convergence(bmm_test, parameter = "Rtilde",
                   items = c(1, 15), assessors = 2)
```

The resulting plot is shown in Figure 5b. As expected, the traces of the augmented rankings for beach 1 and 15 for assessor 2 do cross each other, since no ordering is implied between them. Ideally, we should look at trace plots for augmented ranks for more assessors to be sure that the algorithm is close to convergence. We can plot assessors 1-8 by setting `assessors = 1:8`. We also quite arbitrarily pick items 13-15, but the same procedure can be repeated for other items.

```
assess_convergence(bmm_test, parameter = "Rtilde",
                   items = 13:15, assessors = 1:8)
```

The resulting plot is shown in Figure 6, indicating good mixing.

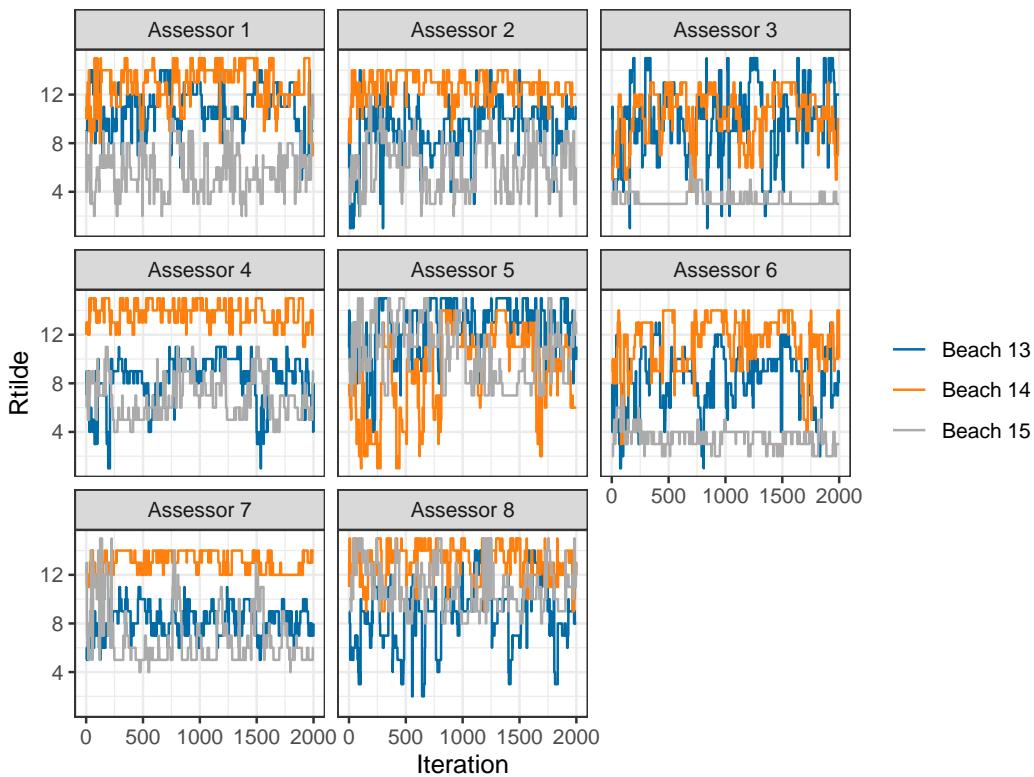


Figure 6: Trace plots of augmented ranks \tilde{R} for beaches 13-15 and assessors 1-8, indicating that the MCMC algorithm obtains good mixing after a low number of iterations.

Posterior distributions

Based on the convergence diagnostics, and being fairly conservative, we discard the first 2,000 MCMC iterations as burn-in, and take 100,000 additional samples.

```
bmm_beaches <- compute_mallows(rankings = beach_init_rank, preferences = beach_tc,
                                    nmc = 102000, save_aug = TRUE)
bmm_beaches$burnin <- 2000
```

The posterior distributions of α and ρ can be studied as shown in the previous sections. Posterior intervals for the latent rankings of each beach are obtained with `compute_posterior_intervals`:

```
compute_posterior_intervals(bmm_beaches, parameter = "rho")
```

item	parameter	mean	median	conf_level	hpdi	central_interval
<fct>	<chr>	<dbl>	<dbl>	<chr>	<chr>	<chr>
1 Beach 1	rho	7	7	95 %	[7]	[6,7]
2 Beach 2	rho	15	15	95 %	[15]	[15]
3 Beach 3	rho	3	3	95 %	[3,4]	[3,4]
4 Beach 4	rho	12	12	95 %	[11,13]	[11,14]
5 Beach 5	rho	9	9	95 %	[8,10]	[8,10]
6 Beach 6	rho	2	2	95 %	[1,2]	[1,2]
7 Beach 7	rho	9	8	95 %	[8,10]	[8,10]
8 Beach 8	rho	12	11	95 %	[11,13]	[11,14]
9 Beach 9	rho	1	1	95 %	[1,2]	[1,2]
10 Beach 10	rho	6	6	95 %	[5,6]	[5,7]
11 Beach 11	rho	4	4	95 %	[3,4]	[3,5]
12 Beach 12	rho	13	13	95 %	[12,14]	[12,14]
13 Beach 13	rho	10	10	95 %	[8,10]	[8,10]
14 Beach 14	rho	13	14	95 %	[12,14]	[11,14]
15 Beach 15	rho	5	5	95 %	[5,6]	[4,6]

We can also rank the beaches according to their cumulative probability (CP) consensus ([Vitelli](#)

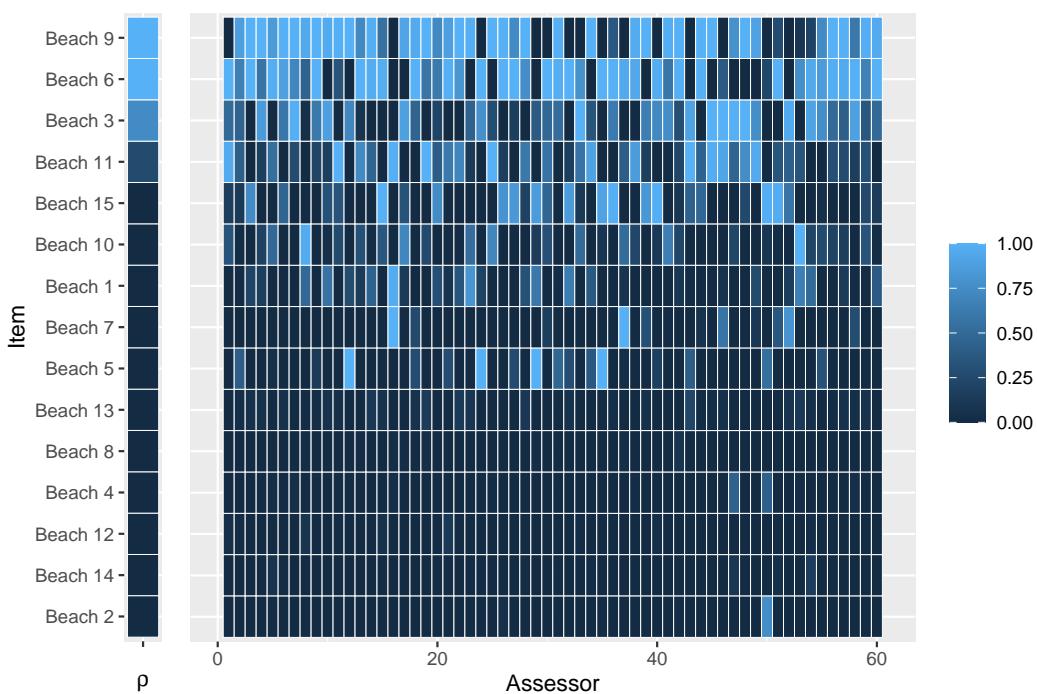


Figure 7: Probability of being ranked top-3 for each beach in the beach preference example (left) and the probability that each assessor ranks the given beach among top-3 (right). Beaches 6 and 9 are most popular overall, but the assessor differ considerably in their preference for these beaches, as can be seen by the varying pattern of light and dark blue.

et al., 2018, Section 5.1) and their maximum posterior (MAP) rankings. This is done with the function `compute_consensus`, and the following call returns the CP consensus:

```
compute_consensus(bmm_beaches, type = "CP")
```

```
# A tibble: 15 x 3
  ranking item      cumprob
  <dbl> <chr>     <dbl>
1      1 Beach 9  0.896
2      2 Beach 6  1
3      3 Beach 3  0.738
4      4 Beach 11 0.966
5      5 Beach 15 0.953
6      6 Beach 10 0.971
7      7 Beach 1  1
8      8 Beach 7  0.528
9      9 Beach 5  0.887
10     10 Beach 13 1.00
11     11 Beach 8  0.508
12     12 Beach 4  0.717
13     13 Beach 12 0.643
14     14 Beach 14 0.988
15     15 Beach 2  1
```

The column `cumprob` shows the probability of having the given rank or lower. Looking at the second row, for example, this means that beach 6 has probability 1 of having latent ranking 2 or lower. Next, beach 3 has probability 0.738 of having latent rank 3 or lower. This is an example of how the Bayesian framework can be used to not only rank items, but also to give posterior assessments of the uncertainty of the rankings. The MAP consensus is obtained similarly, by setting `type = "MAP"`.

Keeping in mind that the ranking of beaches is based on sparse pairwise preferences, we can also ask: for beach i , what is the probability of being ranked top- k by assessor j , and what is the probability of having latent rank among the top- k . The function `plot_top_k` plots these probabil-

ties. By default, it sets $k = 3$, so a heatplot of the probability of being ranked top-3 is obtained with the call:

```
plot_top_k(bmm_beaches)
```

The plot is shown in Figure 7. The left part of the plot shows the beaches ranked according to their CP consensus, and the probability $P(\rho_i) \leq 3$ for each beach i . The right part of the plot shows, for each beach as indicated on the left axis, the probability that assessor j ranks the beach among top-3. For example, we see that assessor 1 has a very low probability of ranking beach 9 among her top-3, while assessor 3 has a very high probability of doing this. The function `predict_top_k` returns a data frame with all the underlying probabilities. For example, in order to find all the beaches that are among the top-3 of assessors 1-5 with more than 90 % probability, we would do:

```
predict_top_k(bmm_beaches) %>%
  filter(prob > 0.9, assessor %in% 1:5)
```

```
# A tibble: 6 x 3
# Groups: assessor [4]
  assessor item      prob
  <dbl> <chr>    <dbl>
1 1 Beach 11 0.955
2 1 Beach 6 0.997
3 3 Beach 6 0.997
4 3 Beach 9 1
5 4 Beach 9 1.00
6 5 Beach 6 0.979
```

Note that assessor 2 does not appear in this table, i.e., there are no beaches for which we are at least 90 % certain that the beach is among assessor 2's top-3.

Clustering with BayesMallows

BayesMallows comes with a set of sushi preference data, in which 5,000 assessors each have ranked a set of 10 types of sushi (Kamishima, 2003). It is interesting to see if we can find subsets of assessors with similar preferences. The sushi dataset was analyzed with the BMM by Vitelli et al. (2018), but the results in that paper differ somewhat from those obtained here, due to a bug in the function that was used to sample cluster probabilities from the Dirichlet distribution.

Computing mixtures of Mallows distributions

The function `compute_mallows_mixtures` computes multiple Mallows models with different numbers of mixture components. It returns a list of models of class `BayesMallowsMixtures`, in which each list element contains a model with a given number of mixture components. Its arguments are `n_clusters`, which specifies the number of mixture components to compute, an optional parameter `c1` which can be set to the return value of the `makeCluster` function in the `parallel` package, and an ellipsis (...) for passing on arguments to `compute_mallows`.

Hypothesizing that we may not need more than 10 clusters to find a useful partitioning of the assessors, we start by doing test runs with 1, 4, 7, and 10 mixture components in order to assess convergence. We set the number of Monte Carlo samples to 5,000, and since this is a test run, we do not save cluster assignments nor within-cluster distances from each MCMC iteration and hence set `save_clus = FALSE` and `include_wcd = FALSE`. We also run the computations in parallel on four cores, one for each mixture component.

```
library("parallel")
cl <- makeCluster(4)
bmm <- compute_mallows_mixtures(n_clusters = c(1, 4, 7, 10),
                                 rankings = sushi_rankings, nmc = 5000,
                                 save_clus = FALSE, include_wcd = FALSE, cl = cl)
stopCluster(cl)
```

Convergence diagnostics

The function `assess_convergence` automatically creates a grid plot when given an object of class `BayesMallowsMixtures`, so we can check the convergence of α with the command

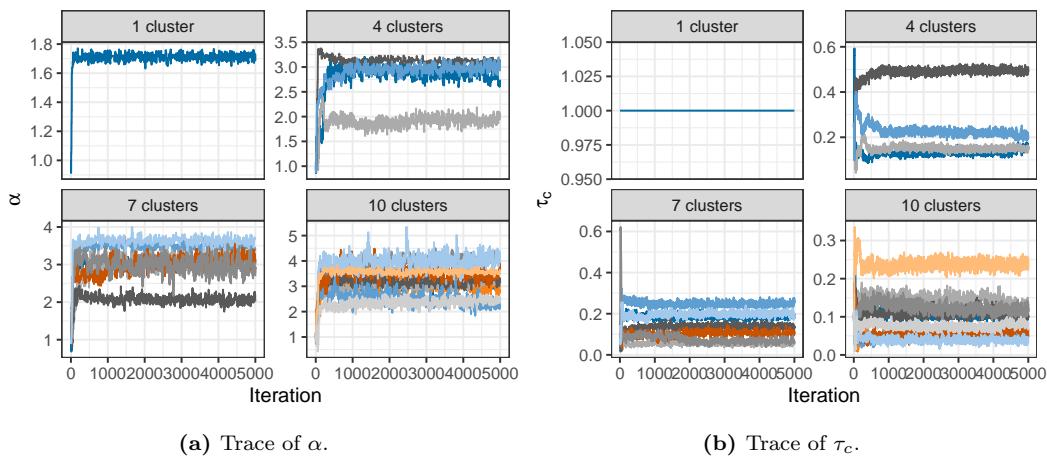


Figure 8: Trace plot of α and τ_c for the sushi dataset with 1, 4, 7, and 10 mixture components, respectively. Both trace plots indicate good mixing after a few thousand iterations.

```
assess_convergence(bmm)
```

The resulting plot is given in Figure 8a, showing that all the chains seem to be close to convergence quite quickly. We can also make sure that the posterior distributions of the cluster probabilities τ_c , ($c = 1, \dots, C$) have converged properly, by setting `parameter = "cluster_probs"`.

```
assess_convergence(bmm, parameter = "cluster_probs")
```

The trace plots for each number of mixture components are shown in Figure 8b. Note that with only one cluster, the cluster probability is fixed at the value 1, while for other number of mixture components, the chains seem to be mixing well.

Deciding on the number of mixtures

Given the convergence assessment of the previous section, we are fairly confident that a burn-in of 5,000 is sufficient. We run 95,000 additional iterations, and try from 1 to 10 mixture components. Our goal is now to determine the number of mixture components to use, and in order to create an elbow plot, we set `include_wcd = TRUE` to compute the within-cluster distances in each step of the MCMC algorithm. Since the posterior distributions of ρ_c ($c = 1, \dots, C$) are highly peaked, we save some memory by only saving every 10th value of ρ by setting `rho_thinning = 10`.

```
cl <- makeCluster(4)
bmm <- compute_mallows_mixtures(n_clusters = 1:10, rankings = sushi_rankings,
                                    nmc = 100000, rho_thinning = 10, save_clus = FALSE,
                                    include_wcd = TRUE, cl = cl)
stopCluster(cl)
plot_elbow(bmm, burnin = 5000) # Create elbow plot
```

The resulting elbow plot is a notched boxplot (Mcgill et al., 1978; Wickham, 2016) shown in Figure 9, for which the barely visible upper and lower whiskers represent approximate 95 % confidence intervals. Although not clear-cut, we see that the within-cluster sum of distances levels off at around 5 clusters, and hence we choose to use 5 clusters in our model.

Posterior distributions

Having chosen 5 mixture components, we go on to fit a final model, still running 95,000 iterations after burnin. This time we call `compute_mallows` and set `n_clusters = 5`. We also set `save_clus = TRUE` and `clus_thin = 10` to save the cluster assignments of each assessor in every 10th iteration, and `rho_thinning = 10` to save the estimated latent rank every 10th iteration.

```
bmm <- compute_mallows(rankings = sushi_rankings, n_clusters = 5, save_clus = TRUE,
                        clus_thin = 10, nmc = 100000, rho_thinning = 10)
bmm$burnin <- 5000
```

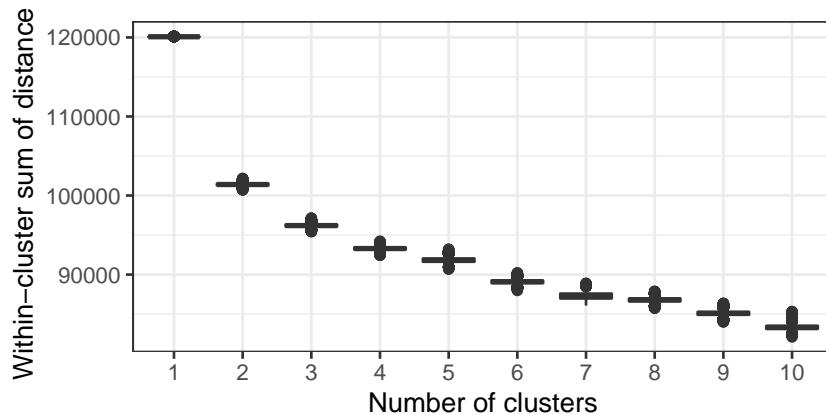


Figure 9: Elbow plot for the sushi mixture models. While it is not entirely clear where the elbow occurs, we choose the mixture distribution with five clusters.

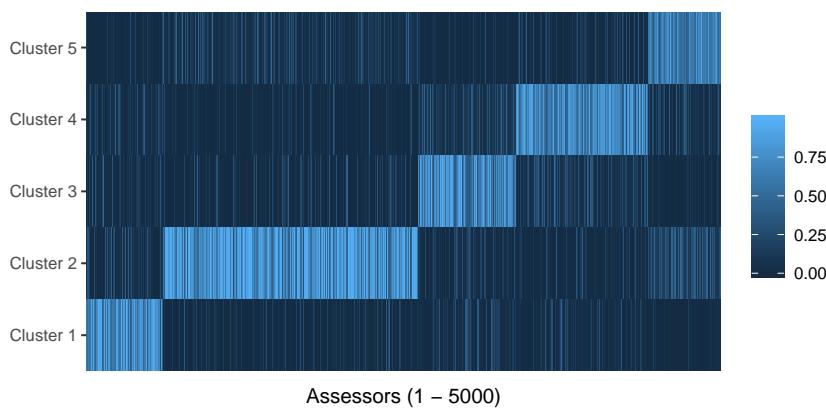


Figure 10: Posterior probabilities of assignment to each cluster for each of the 5000 assessors in the sushi dataset. The scale to the right shows the color coding of probabilities. The blocks of light colors along the anti-diagonal show the clusters to which the assessors were assigned. Darker colors within these blocks indicate assessors whose cluster assignment is uncertain.

We can plot the posterior distributions of α and ρ in each cluster using `plot.BayesMallows` as shown previously for the potato data. We can also show the posterior distributions of the cluster probabilities, using:

```
plot(bmm, parameter = "cluster_probs")
```

Using the argument `parameter = "cluster_assignment"`, we can visualize the posterior probability for each assessor of belonging to each cluster:

```
plot(bmm, parameter = "cluster_assignment")
```

The resulting plot is shown in Figure 10. The underlying numbers can be obtained using the function `assign_cluster`.

We can find clusterwise consensus rankings using `compute_consensus`. The following call finds the CP consensuses, and then uses `select` from `dplyr` and `spread` from `tidyverse` (Wickham and Henry, 2018) to create one column for each cluster. The result is shown in Table 1.

```
library("tidyverse")
compute_consensus(bmm) %>%
  select(-cumprob) %>%
  spread(key = cluster, value = item)
```

Note that for estimating cluster specific parameters, label switching is a potential problem that needs to be handled. `BayesMallows` ignores label switching issues inside the MCMC, because it has been shown that this approach is better for ensuring full convergence of the chain (Jasra et al., 2005; Celeux et al., 2000). MCMC iterations can be re-ordered after convergence is achieved, for

	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5
1	shrimp	fatty tuna	fatty tuna	fatty tuna	fatty tuna
2	sea eel	sea urchin	sea eel	tuna	sea urchin
3	egg	salmon roe	tuna	shrimp	shrimp
4	squid	sea eel	shrimp	tuna roll	tuna
5	salmon roe	tuna	tuna roll	squid	salmon roe
6	fatty tuna	shrimp	squid	salmon roe	squid
7	tuna	tuna roll	egg	egg	tuna roll
8	tuna roll	squid	cucumber roll	cucumber roll	sea eel
9	cucumber roll	egg	salmon roe	sea eel	egg
10	sea urchin	cucumber roll	sea urchin	sea urchin	cucumber roll

Table 1: CP consensus for each of the clusters found for sushi data.

example by using the implementation of Stephens' algorithm (Stephens, 2000) provided by the R package `label.switching` (Papastamoulis, 2016). A full example of how to assess label switching after running `compute_mallows` is provided by running the following command:

```
help("label_switching")
```

For the sushi data analyzed in this section, no label switching is detected by Stephen's algorithm.

Discussion

In this paper we discussed the methodological background and computational strategies for the **BayesMallows** package, implementing the inferential framework for the analysis of preference data based on the Bayesian Mallows model, as introduced in Vitelli et al. (2018). The package aims at providing a general probabilistic tool, capable of performing various inferential tasks (estimation, classification, prediction) with a proper uncertainty quantification. Moreover, the package widens the applicability of the Mallows model, by providing reliable algorithms for approximating the associated partition function, which has been the bottleneck for a successful use of this general and flexible model so far. Finally, it handles a variety of preference data types (partial rankings, pairwise preferences), and it could possibly handle many others which can lie in the above mentioned categories (noisy continuous measurements, clicking data, ratings).

One of the most important features of the **BayesMallows** package is that, despite implementing a Bayesian model, and thus relying on MCMC algorithms, its efficient implementation makes it possible to manage large datasets. The package can easily handle up to hundreds of items, and thousands of assessors; an example is the MovieLens data analyzed in Section 6.4 of (Vitelli et al., 2018). By using the log-sum-exp trick, the implementation of the importance sampler is able to handle at least ten thousand items without numerical overflow. We believe that all these features make the package a unique resource for fitting the Mallows model to large data, with the benefits of a fully probabilistic interpretation.

Nonetheless, we also recognize that the **BayesMallows** package can open the way for further generalizations. The Bayesian Mallows model for time-varying rankings that has been introduced in Asfaw et al. (2017) will be considered for a future release. Some further extensions which we might consider to implement in the **BayesMallows** in the future include: fitting an infinite mixture of Mallows models for automatically performing model selection; allowing for a non-uniform prior for ρ ; performing automatic item selection; estimating the assessors' quality as rankers; and finally including covariates, both on the assessors and on the items. In addition, since the data augmentation steps in the MCMC algorithm are independent across assessors, potential speedup in the case of missing data or pairwise preferences can be obtained by updating the augmented data in parallel, and this is likely to be part of a future package update.

Acknowledgments

The authors would like to thank Arnoldo Frigessi and Elja Arjas for fruitful discussions.

Bibliography

- M. Alvo and P. L. Yu. *Statistical Methods for Ranking Data*. Frontiers in Probability and the Statistical Sciences. Springer, New York, NY, USA, 2014. URL <https://doi.org/10.1007/978-1-4939-1471-5>. [p]
- D. Asfaw, V. Vitelli, Ø. Sørensen, E. Arjas, and A. Frigessi. Time-varying rankings with the Bayesian Mallows model. *Stat*, 6(1):14–30, 2017. URL <https://doi.org/10.1002/sta4.132>. [p]
- G. Celeux, M. Hurn, and C. Robert. Computational and inferential difficulties with mixture posterior distribution. *Journal of the American Statistical Association*, 95(451):957–970, 2000. URL <https://doi.org/10.2307/2669477>. [p]
- M. Crispino, E. Arjas, V. Vitelli, N. Barrett, and A. Frigessi. A Bayesian Mallows approach to nontransitive pair comparison data: How human are sounds? *The Annals of Applied Statistics*, 13(1):492–519, Mar. 2019. URL <https://doi.org/10.1214/18-aos1203>. [p]
- J. C. de Borda. Mémoire sur les élections au scrutin, histoire de l’académie royale des sciences. *Paris, France*, 1781. [p]
- P. Diaconis. *Group Representations in Probability and Statistics*, volume 11 of *Lecture Notes - Monograph Series*. Institute of Mathematical Statistics, Hayward, CA, USA, 1988. [p]
- M. A. Fligner and J. S. Verducci. Distance based ranking models. *Journal of the Royal Statistical Society: Series B (Methodological)*, 48(3):359–369, July 1986. URL <https://doi.org/10.1111/j.2517-6161.1986.tb01420.x>. [p]
- E. Irurozki, B. Calvo, and A. Lozano. Sampling and learning the Mallows and weighted Mallows models under the Hamming distance. *Technical Report*, 2014. URL <https://addi.ehu.es/handle/10810/11240>. [p]
- E. Irurozki, B. Calvo, and J. A. Lozano. PerMallows: An R Package for Mallows and Generalized Mallows Models. *Journal of Statistical Software*, 71(12), 2016a. URL <https://doi.org/10.18637/jss.v071.i12>. [p]
- E. Irurozki, B. Calvo, and J. A. Lozano. Sampling and learning Mallows and generalized Mallows models under the Cayley distance. *Methodology and Computing in Applied Probability*, 20(1):1–35, June 2016b. URL <https://doi.org/10.1007/s11009-016-9506-7>. [p]
- A. Jasra, C. C. Holmes, and D. A. Stephens. Markov chain Monte Carlo methods and the label switching problem in Bayesian mixture modeling. *Statistical Science*, 20(1):50–67, Feb. 2005. URL <https://doi.org/10.1214/088342305000000016>. [p]
- T. Kamishima. Nantonac collaborative filtering: Recommendation based on order responses. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’03, pages 583–588, New York, NY, USA, 2003. ACM. ISBN 1-58113-737-0. URL <https://doi.org/10.1145/956750.956823>. [p]
- P. H. Lee and P. L. Yu. An R package for analyzing and modeling ranking data. *BMC Medical Research Methodology*, 13(1):65, May 2013. ISSN 1471-2288. doi: 10.1186/1471-2288-13-65. URL <https://doi.org/10.1186/1471-2288-13-65>. [p]
- Q. Liu, M. Crispino, I. Scheel, V. Vitelli, and A. Frigessi. Model-based learning from preference data. *Annual Review of Statistics and Its Application*, 6(1):329–354, 2019. URL <https://doi.org/10.1146/annurev-statistics-031017-100213>. [p]
- T. Lu and C. Boutilier. Effective sampling and learning for Mallows models with pairwise-preference data. *Journal of Machine Learning Research*, 15:3783–3829, 2014. [p]
- C. L. Mallows. Non-null ranking models. i. *Biometrika*, 44(1-2):114–130, 1957. URL <https://doi.org/10.1093/biomet/44.1-2.114>. [p]
- J. I. Marden. *Analyzing and Modeling Rank Data*, volume 64 of *Monographs on Statistics and Applied Probability*. Chapman & Hall, Cambridge, MA, USA, 1995. [p]
- M. J. A. N. d. C. Marquis of Condorcet. Essai sur l’application de l’analyse à la probabilité des décisions rendues à la pluralité des voix. *Paris: De l’imprimerie royale*, 1785. [p]
- R. McGill, J. W. Tukey, and W. A. Larsen. Variations of box plots. *The American Statistician*, 32(1):12–16, 1978. URL <https://doi.org/10.1080/00031305.1978.10479236>. [p]

- M. Meilă and H. Chen. Dirichlet process mixtures of generalized Mallows models. In *Proceedings of the Twenty-Sixth Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-10)*, pages 358–367, Corvallis, OR, USA, 2010. AUAI Press. [p]
- O. Mersmann. *microbenchmark: Accurate Timing Functions*, 2018. URL <https://CRAN.R-project.org/package=microbenchmark>. R package version 1.4-6. [p]
- S. Mukherjee. Estimation in exponential families on permutations. *The Annals of Statistics*, 44(2):853–875, 2016. URL <https://doi.org/doi:10.1214/15-AOS1389>. [p]
- K. Müller and H. Wickham. *tibble: Simple Data Frames*, 2018. URL <https://CRAN.R-project.org/package=tibble>. R package version 1.4.2. [p]
- P. Papastamoulis. *label.switching*: An R package for dealing with the label switching problem in MCMC outputs. *Journal of Statistical Software*, 69, 2016. URL <https://doi.org/10.18637/jss.v069.c01>. [p]
- Z. Qian. *rankdist: Distance Based Ranking Models*, 2018. URL <https://CRAN.R-project.org/package=rankdist>. R package version 1.1-3. [p]
- N. J. A. Sloane. The On-Line Encyclopedia of Integer Sequences, 2017. URL <http://oeis.org>. [p]
- M. Stephens. Dealing with label switching in mixture models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 62(4):795–809, Nov. 2000. URL <https://doi.org/10.1111/1467-9868.00265>. [p]
- V. Vitelli, Ø. Sørensen, M. Crispino, A. Frigessi, and E. Arjas. Probabilistic preference learning with the Mallows rank model. *Journal of Machine Learning Research*, 18(1):5796–5844, Jan. 2018. [p]
- H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. ISBN 978-3-319-24277-4. URL <http://ggplot2.org>. [p]
- H. Wickham and L. Henry. *tidyr: Easily Tidy Data with 'spread()' and 'gather()' Functions*, 2018. URL <https://CRAN.R-project.org/package=tidyr>. R package version 0.8.1. [p]
- H. Wickham, R. François, L. Henry, and K. Müller. *dplyr: A Grammar of Data Manipulation*, 2018. URL <https://CRAN.R-project.org/package=dplyr>. R package version 0.7.7. [p]

Øystein Sørensen
Center for Lifespan Changes in Brain and Cognition
Department of Psychology
University of Oslo
ORCID: 0000-0003-0724-3542
oystein.sorensen@psykologi.uio.no

Marta Crispino
Univ. Grenoble Alpes, Inria, CNRS, LJK
38000 Grenoble, France
crispino.marta8@gmail.com

Qinghua Liu
Department of Mathematics
University of Oslo
qinghual@math.uio.no

Valeria Vitelli
Oslo Centre for Biostatistics and Epidemiology
Department of Biostatistics
University of Oslo
OORCID: 0000-0002-6746-0453
valeria.vitelli@medisin.uio.no

Variable Importance Plots—An Introduction to the `vip` Package

by Brandon M. Greenwell, Bradley C. Boehmke

Abstract In the era of “big data”, it is becoming more of a challenge to not only build state-of-the-art predictive models, but also gain an understanding of what’s really going on in the data. For example, it is often of interest to know which, if any, of the predictors in a fitted model are relatively influential on the predicted outcome. Some modern algorithms—like random forests (RFs) and gradient boosted decision trees (GBMs)—have a natural way of quantifying the importance or relative influence of each feature. Other algorithms—like naive Bayes classifiers and support vector machines—are not capable of doing so and *model-agnostic approaches* are generally used to measure each predictor’s importance. Enter `vip`, an R package for constructing variable importance scores/plots for many types of supervised learning algorithms using model-specific and novel model-agnostic approaches. We’ll also discuss a novel way to display both feature importance and feature effects together using `sparklines`, a very small line chart conveying the general shape or variation in some feature that can be directly embedded in text or tables.

Introduction

Too often machine learning (ML) models are summarized using a single metric (e.g., cross-validated accuracy) and then put into production. Although we often care about the predictions from these models, it is becoming routine (and good practice) to also better understand the predictions! Understanding how an ML model makes its predictions helps build trust in the model and is the fundamental idea of the emerging field of *interpretable machine learning* (IML).¹ For an in-depth discussion on IML, see Molnar (2019b). In this paper, we focus on *global methods* for quantifying the importance² of features in an ML model; that is, methods that help us understand the global contribution each feature has to a model’s predictions. Computing variable importance (VI) and communicating them through variable importance plots (VIPs) is a fundamental component of IML and is the main topic of this paper.

While many of the procedures discussed in this paper apply to any model that makes predictions, it should be noted that these methods heavily depend on the accuracy and importance of the fitted model; hence, unimportant features may appear relatively important (albeit not predictive) in comparison to the other included features. For this reason, we stress the usefulness of understanding the scale on which VI scores are calculated and take that into account when assessing the importance of each feature and communicating the results to others. Also, we should point out that this work focuses mostly on *post-hoc interpretability* where a trained model is given and the goal is to understand what features are driving the model’s predictions. Consequently, our work focuses on functional understanding of the model in contrast to the lower-level mechanistic understanding (Montavon et al., 2018). That is, we seek to explain the relationship between the model’s prediction behavior and features without explaining the full internal representation of the model.³

VI scores and VIPs can be constructed for general ML models using a number of available packages. The `iml` package (Molnar, 2019a) provides the `FeatureImp()` function which computes feature importance for general prediction models using the permutation approach (discussed later). It is written in `R6` (Chang, 2019) and allows the user to specify a generic loss function or select one from a pre-defined list (e.g., `loss = "mse"` for mean squared error). It also allows the user to specify whether importance is measured as the difference or as the ratio of the original model error and the model error after permutation. The user can also specify the number of repetitions used when permuting each feature to help stabilize the variability in the procedure. The `iml::FeatureImp()` function can also be run in parallel using any parallel backend supported by the `foreach` package (Revolution Analytics and Weston).

The `ingredients` package (Biecek et al., 2019a) also provides permutation-based VI scores through the `feature_importance()` function. (Note that this function recently replaced the now deprecated

¹Although “interpretability” is difficult to formally define in the context of ML, we follow Doshi-Velez and Kim (2017) and describe “interpretable” as the “...ability to explain or to present in understandable terms to a human.”

²In this context “importance” can be defined in a number of different ways. In general, we can describe it as *the extent to which a feature has a “meaningful” impact on the predicted outcome*. A more formal definition and treatment can be found in van der Laan (2006).

³We refer the reader to Poulin et al. (2006), Caruana et al. (2015), Bibal and Frénay (2016), and Bau et al. (2017), for discussions around model structure interpretation.

DALEX function `variable_importance()` (Biecek, 2019).) Similar to `iml::FeatureImp()`, this function allows the user to specify a loss function and how the importance scores are computed (e.g., using the difference or ratio). It also provides an option to sample the training data before shuffling the data to compute importance (the default is to use `n_sample = 1000`), which can help speed up computation.

The `mmpf` package (Jones, 2018) also provides permutation-based VI scores via the `mmpf::permutationImportance()` function. Similar to the `iml` and `ingredients` implementation, this function is flexible enough to be applied to any class of ML models in R.

The `varImp` package (Probst, 2019) extends the permutation-based method for RFs in package `party` (Hothorn et al., 2019) to arbitrary measures from the `measures` package (Probst, 2018). Additionally, the functions in `varImp` include the option of using the conditional approach described in Strobl et al. (2008) which is more reliable in the presence of correlated features. A number of other RF-specific VI packages exist on CRAN, including, but not limited to, `vita` (Celik, 2015), `rfVarImpOOB` (Loecher, 2019), `randomForestExplainer` (Paluszynska et al., 2019), and `tree.interpreter` (Sun, 2019).⁴.

The `caret` package (Kuhn, 2020) includes a general `varImp()` function for computing model-specific and filter-based VI scores. Filter-based approaches, which are described in Kuhn and Johnson (2013), do not make use of the fitted model to measure VI. They also do not take into account the other predictors in the model. For regression problems, a popular filter-based approach to measuring the VI of a numeric predictor x is to first fit a flexible nonparametric model between x and the target Y ; for example, the locally-weighted polynomial regression (LOWESS) method developed by Cleveland (1979). From this fit, a pseudo- R^2 measure can be obtained from the resulting residuals and used as a measure of VI. For categorical predictors, a different method based on standard statistical tests (e.g., t -tests and ANOVAs) can be employed; see Kuhn and Johnson (2013) for details. For classification problems, an area under the ROC curve (AUC) statistic can be used to quantify predictor importance. The AUC statistic is computed by using the predictor x as input to the ROC curve. If x can reasonably separate the classes of Y , that is a clear indicator that x is an important predictor (in terms of class separation) and this is captured in the corresponding AUC statistic. For problems with more than two classes, extensions of the ROC curve or a one-vs-all approach can be used.

If you use the `mlr` interface for fitting ML models (Bischl et al., 2020), then you can use the `getFeatureImportance()` function to extract model-specific VI scores from various tree-based models (e.g., RFs and GBMs). Unlike `caret`, the model needs to be fit via the `mlr` interface; for instance, you cannot use `getFeatureImportance()` on a `ranger` (Wright et al., 2020) model unless it was fit using `mlr`.

While the `iml` and `DALEX` packages provide model-agnostic approaches to computing VI, `caret`, and to some extent, `mlr`, provide model-specific approaches (e.g., using the absolute value of the t -statistic for linear models) as well as less accurate filter-based approaches. Furthermore, each package has a completely different interface (e.g., `iml` is written in R6). The `vip` package (Greenwell et al., 2019) strives to provide a consistent interface to both model-specific and model-agnostic approaches to feature importance that is simple to use. The three most important functions exported by `vip` are described below:

- `vi()` computes VI scores using model-specific or model-agnostic approaches (the results are always returned as a tibble (Müller and Wickham, 2019));
- `vip()` constructs VIPs using model-specific or model-agnostic approaches with `ggplot2`-style graphics (Wickham et al., 2019);
- `add_sparklines()` adds a novel sparkline representation of feature effects (e.g., *partial dependence plots*) to any VI table produced by `vi()`.

There's also a function called `vint()` (for variable interactions) but it is experimental and will not be discussed here; the interested reader is pointed to Greenwell et al. (2018). Note that `vi()` is actually a wrapper around four workhorse functions, `vi_model()`, `vi_firm()`, `vi_permute()`, and `vi_shap()`, that compute various types of VI scores. The first computes model-specific VI scores, while the latter three produce model-agnostic ones. The workhorse function that actually gets called is controlled by the `method` argument in `vi()`; the default is `method = "model"` which corresponds to model-specific VI (see `?vip::vi` for details and links to further documentation).

⁴These packages were discovered using `pkgsearch`'s `ps()` function (Csárdi and Salmon, 2019) with the key phrases “variable importance” and “feature importance”.

Constructing VIPs in R

We'll illustrate major concepts using the Friedman 1 benchmark problem described in Friedman (1991) and Breiman (1996):

$$Y_i = 10 \sin(\pi X_{1i} X_{2i}) + 20(X_{3i} - 0.5)^2 + 10X_{4i} + 5X_{5i} + \epsilon_i, \quad i = 1, 2, \dots, n, \quad (1)$$

where $\epsilon_i \stackrel{iid}{\sim} N(0, \sigma^2)$. Data from this model can be generated using the `vip::gen_friedman()`. By default, the features consist of 10 independent variables uniformly distributed on the interval $[0, 1]$; however, only 5 out of these 10 are actually used in the true model. The code chunk below simulates 500 observations from the model in Equation (1) with $\sigma = 1$; see `?vip::gen_friedman` for details.

```
trn <- vip::gen_friedman(500, sigma = 1, seed = 101) # simulate training data
tibble::as_tibble(trn) # inspect output

#> # A tibble: 500 x 11
#>   y     x1     x2     x3     x4     x5     x6     x7     x8     x9     x10
#>   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#> 1 14.9  0.372  0.406  0.102  0.322  0.693  0.758  0.518  0.530  0.878  0.763
#> 2 15.3  0.0438 0.602  0.602  0.999  0.776  0.533  0.509  0.487  0.118  0.176
#> 3 15.1  0.710  0.362  0.254  0.548  0.0180 0.765  0.715  0.844  0.334  0.118
#> 4 10.7  0.658  0.291  0.542  0.327  0.230  0.301  0.177  0.346  0.474  0.283
#> 5 17.6  0.250  0.794  0.383  0.947  0.462  0.00487 0.270  0.114  0.489  0.311
#> 6 18.3  0.300  0.701  0.992  0.386  0.666  0.198  0.924  0.775  0.736  0.974
#> 7 14.6  0.585  0.365  0.283  0.488  0.845  0.466  0.715  0.202  0.905  0.640
#> 8 17.0  0.333  0.552  0.858  0.509  0.697  0.388  0.260  0.355  0.517  0.165
#> 9  8.54  0.622  0.118  0.490  0.390  0.468  0.360  0.572  0.891  0.682  0.717
#> 10 15.0  0.546  0.150  0.476  0.706  0.829  0.373  0.192  0.873  0.456  0.694
#> # ... with 490 more rows
```

From Equation (1), it should be clear that features X_1 – X_5 are the most important! (The others don't influence Y at all.) Also, based on the form of the model, we'd expect X_4 to be the most important feature, probably followed by X_1 and X_2 (both comparably important), with X_5 probably being less important. The influence of X_3 is harder to determine due to its quadratic nature, but it seems likely that this nonlinearity will suppress the variable's influence over its observed range (i.e., 0–1).

Model-specific VI

Some machine learning algorithms have their own way of quantifying the importance of each feature, which we refer to as *model-specific VI*. We describe some of these in the subsections that follow. One particular issue with model-specific VI scores is that they are not necessarily comparable across different types of models. For example, directly comparing the impurity-based VI scores from tree-based models to the absolute value of the t -statistic in linear models.

Decision trees and tree ensembles

Decision trees probably offer the most natural model-specific approach to quantifying the importance of each feature. In a binary decision tree, at each node t , a single predictor is used to partition the data into two homogeneous groups. The chosen predictor is the one that maximizes some measure of improvement i^t . The relative importance of predictor X is the sum of the squared improvements over all internal nodes of the tree for which X was chosen as the partitioning variable; see Breiman et al. (1984) for details. This idea also extends to ensembles of decision trees, such as RFs and GBMs. In ensembles, the improvement score for each predictor is averaged across all the trees in the ensemble. Fortunately, due to the stabilizing effect of averaging, the improvement-based VI metric is often more reliable in large ensembles; see Hastie et al. (2009, p. 368).

RFs offer an additional method for computing VI scores. The idea is to use the leftover *out-of-bag* (OOB) data to construct validation-set errors for each tree. Then, each predictor is randomly shuffled in the OOB data and the error is computed again. The idea is that if variable X is important, then the validation error will go up when X is perturbed in the OOB data. The difference in the two errors is recorded for the OOB data then averaged across all trees in the forest. Note that both methods for constructing VI scores can be unreliable in certain situations;

for example, when the predictor variables vary in their scale of measurement or their number of categories (Strobl et al., 2007), or when the predictors are highly correlated (Strobl et al., 2008). The `varImp` package discussed earlier provides methods to address these concerns for random forests in package `party`, with similar functionality also built into the `partykit` package (Hothorn and Zeileis, 2019). The `vip` package also supports the conditional importance described in (Strobl et al., 2008) for both `party`- and `partykit`-based RFs; see `?vip::vi_model` for details. Later on, we'll discuss a more general permutation method that can be applied to any supervised learning model.

To illustrate, we fit a CART-like regression tree, RF, and GBM to the simulated training data. (Note: there are a number of different packages available for fitting these types of models, we just picked popular implementations for illustration.)

```
# Load required packages
library(rpart)          # for fitting CART-like decision trees
library(randomForest)    # for fitting RFs
library(xgboost)         # for fitting GBMs

# Fit a single regression tree
tree <- rpart(y ~ ., data = trn)

# Fit an RF
set.seed(101) # for reproducibility
rfo <- randomForest(y ~ ., data = trn, importance = TRUE)

# Fit a GBM
set.seed(102) # for reproducibility
bst <- xgboost(
  data = data.matrix(subset(trn, select = -y)),
  label = trn$y,
  objective = "reg:squarederror",
  nrounds = 100,
  max_depth = 5,
  eta = 0.3,
  verbose = 0 # suppress printing
)
```

Each of the above packages include the ability to compute VI scores for all the features in the model; however, the implementation is rather package-specific, as shown in the code chunk below. The results are displayed in Figure 1 (the code to reproduce these plots has been omitted but can be made available upon request).

```
# Extract VI scores from each model
vi_tree <- tree$variable.importance
vi_rfo <- rfo$variable.importance # or use `randomForest::importance(rfo)`
vi_bst <- xgb.importance(model = bst)
```

As we would expect, all three methods rank the variables `x1`–`x5` as more important than the others. While this is good news, it is unfortunate that we have to remember the different functions and ways of extracting and plotting VI scores from various model fitting functions. This is one place where `vip` can help...one function to rule them all! Once `vip` is loaded, we can use `vi()` to extract a tibble of VI scores.⁵

```
# Load required packages
library(vip)

# Compute model-specific VI scores
vi(tree) # CART-like decision tree

#> # A tibble: 10 x 2
#>   Variable Importance
#>   <chr>        <dbl>
#> 1 x4            4234.
```

⁵In order to avoid deprecation warnings due to recent updates to `tibble` and `ggplot2`, the code examples in this article are based on the latest development versions of both `vip` (version 0.2.2.9000) and `pdp` (version 0.7.0.9000); the URL to the development version of each package is available on its associated CRAN landing page.

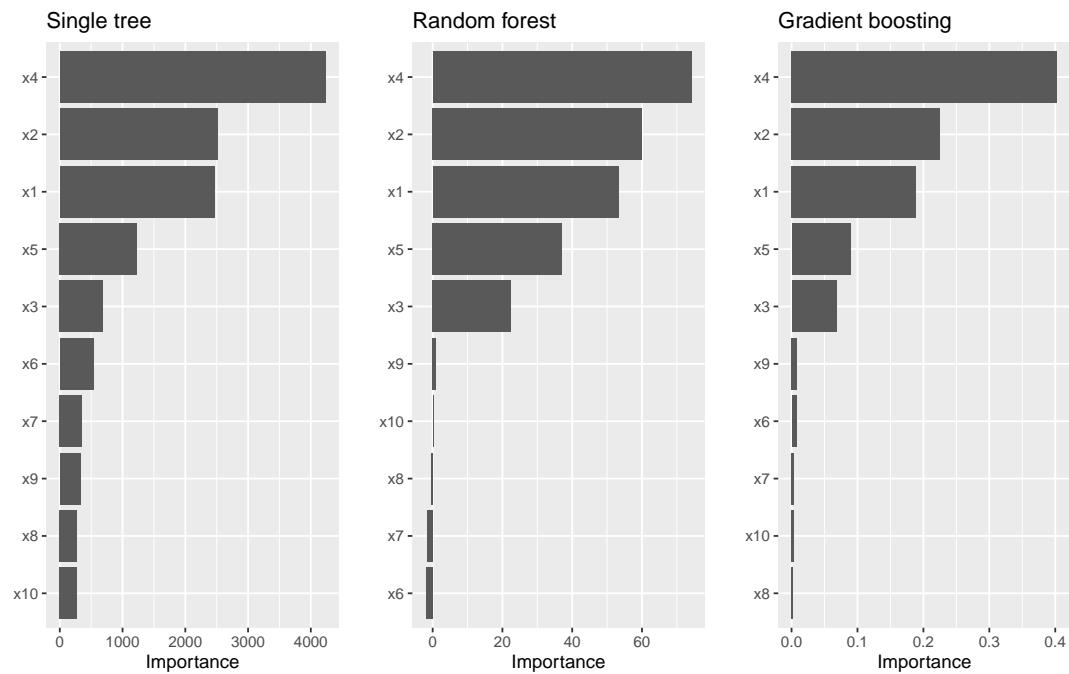


Figure 1: Model-specific VIPs for the three different tree-based models fit to the simulated Friedman data.

```
#> 2 x2      2513.
#> 3 x1      2461.
#> 4 x5      1230.
#> 5 x3      688.
#> 6 x6      533.
#> 7 x7      357.
#> 8 x9      331.
#> 9 x8      276.
#> 10 x10    275.

vi(rfo) # RF

#> # A tibble: 10 x 2
#>   Variable Importance
#>   <chr>        <dbl>
#> 1 x4          74.2
#> 2 x2          59.9
#> 3 x1          53.3
#> 4 x5          37.1
#> 5 x3          22.5
#> 6 x9          1.05
#> 7 x10         0.254
#> 8 x8          -0.408
#> 9 x7          -1.56
#> 10 x6         -2.00

vi(bst) # GBM

#> # A tibble: 10 x 2
#>   Variable Importance
#>   <chr>        <dbl>
#> 1 x4          0.403
#> 2 x2          0.225
#> 3 x1          0.189
#> 4 x5          0.0894
#> 5 x3          0.0682
#> 6 x9          0.00802
#> 7 x6          0.00746
```

```
#> 8 x7      0.00400
#> 9 x10     0.00377
#> 10 x8     0.00262
```

Notice how the `vi()` function always returns a tibble⁶ with two columns: `Variable` and `Importance` (the exceptions are coefficient-based models which also include a `Sign` column giving the sign of the corresponding coefficient, and permutation importance involving multiple Monte Carlo simulations, but more on that later). Also, by default, `vi()` always orders the VI scores from highest to lowest; this, among other options, can be controlled by the user (see `?vip::vi` for details). Plotting VI scores with `vip()` is just as straightforward. For example, the following code can be used to reproduce Figure 1.

```
p1 <- vip(tree) + ggtitle("Single tree")
p2 <- vip(rfo) + ggtitle("Random forest")
p3 <- vip(bst) + ggtitle("Gradient boosting")

# Display plots in a grid (Figure 1)
grid.arrange(p1, p2, p3, nrow = 1)
```

Notice how the `vip()` function always returns a "ggplot" object (by default, this will be a bar plot). For large models with many features, a Cleveland dot plot is more effective (in fact, a number of useful plotting options can be fiddled with). Below we call `vip()` and change a few useful options (the resulting plot is displayed in Figure 2). Note that we can also call `vip()` directly on a "`vi`" object if it's already been constructed.

```
# Construct VIP (Figure 2)
library(ggplot2) # for theme_light() function
vip(bst, num_features = 5, geom = "point", horizontal = FALSE,
    aesthetics = list(color = "red", shape = 17, size = 5)) +
  theme_light()
```

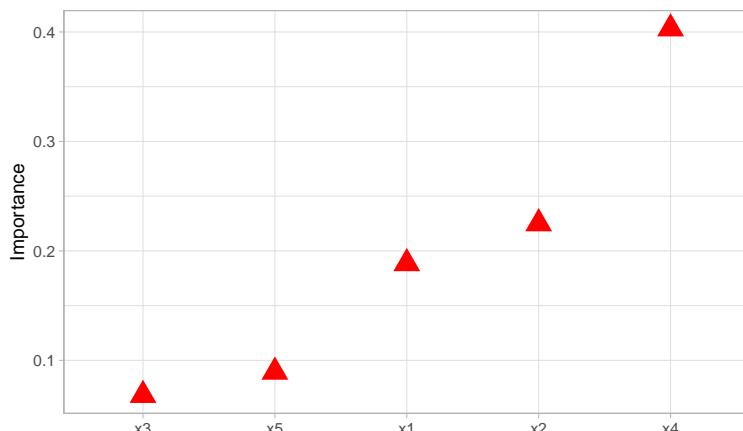


Figure 2: Illustrating various plotting options.

Linear models

In multiple linear regression, or linear models (LMs), the absolute value of the t -statistic (or some other scaled variant of the estimated coefficients) is commonly used as a measure of VI.⁷ The same idea also extends to generalized linear models (GLMs). In the code chunk below, we fit an LM to the simulated Friedman data (`trn`) allowing for all main effects and two-way interactions, then use the `step()` function to perform backward elimination. The resulting VIP is displayed in Figure 3.

```
# Fit a LM
linmod <- lm(y ~ .^2, data = trn)
backward <- step(linmod, direction = "backward", trace = 0)
```

⁶Technically, it's a tibble with an additional "vi" class.

⁷Since this approach is biased towards large-scale features it is important to properly standardize the predictors (before fitting the model) or the estimated coefficients.

```
# Extract VI scores
(vi_backward <- vi(backward))

#> # A tibble: 21 x 3
#>   Variable Importance Sign
#>   <chr>      <dbl> <chr>
#> 1 x4          14.2 POS
#> 2 x2           7.31 POS
#> 3 x1            5.63 POS
#> 4 x5            5.21 POS
#> 5 x3:x5         2.46 POS
#> 6 x1:x10        2.41 NEG
#> 7 x2:x6         2.41 NEG
#> 8 x1:x5          2.37 NEG
#> 9 x10           2.21 POS
#> 10 x3:x4         2.01 NEG
#> # ... with 11 more rows

# Plot VI scores; by default, `vip()` displays the top ten features
pal <- palette.colors(2, palette = "Okabe-Ito") # colorblind friendly palette
vip(vi_backward, num_features = length(coef(backward)), # Figure 3
    geom = "point", horizontal = FALSE, mapping = aes(color = Sign)) +
  scale_color_manual(values = unname(pal)) +
  theme_light() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```

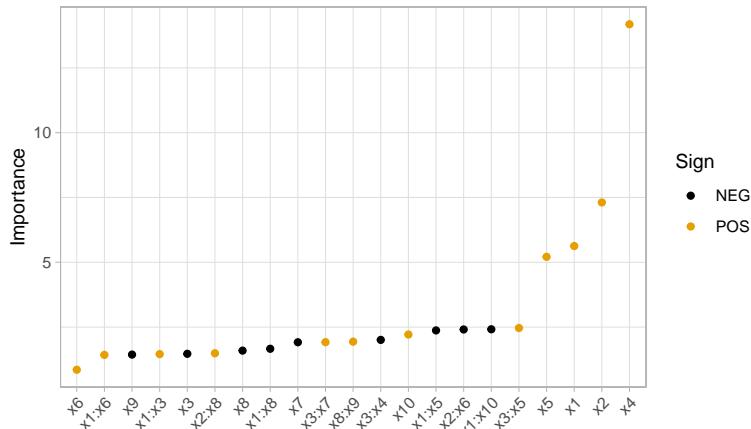


Figure 3: Example VIP from a linear model fit to the simulated Friedman data. The points are colored according to the sign of the associated coefficient.

A major limitation of this approach is that a VI score is assigned to each term in the model, rather than to each individual feature! We can solve this problem using one of the model-agnostic approaches discussed later.

Multivariate adaptive regression splines (MARS), which were introduced in Friedman (1991), is an automatic regression technique and can be seen as a generalization of LMs and GLMs. In the MARS algorithm, the contribution (or VI score) for each predictor is determined using a generalized cross-validation (GCV) statistic (though, other statistics can also be used; see `?vip::vi_model` for details). An example using the `earth` package (Milborrow, 2019) is given below (the results are plotted in Figure 4):

```
# Load required packages
library(earth)

# Fit a MARS model
mars <- earth(y ~ ., data = trn, degree = 2, pmethod = "exhaustive")

# Extract VI scores
vi(mars, type = "gcv")
```

```
#> # A tibble: 10 x 2
#>   Variable Importance
#>   <chr>      <dbl>
#> 1 x4          100
#> 2 x1           83.2
#> 3 x2           83.2
#> 4 x5           59.3
#> 5 x3           43.5
#> 6 x6            0
#> 7 x7            0
#> 8 x8            0
#> 9 x9            0
#> 10 x10          0

# Plot VI scores (Figure 4)
vip(mars)
```

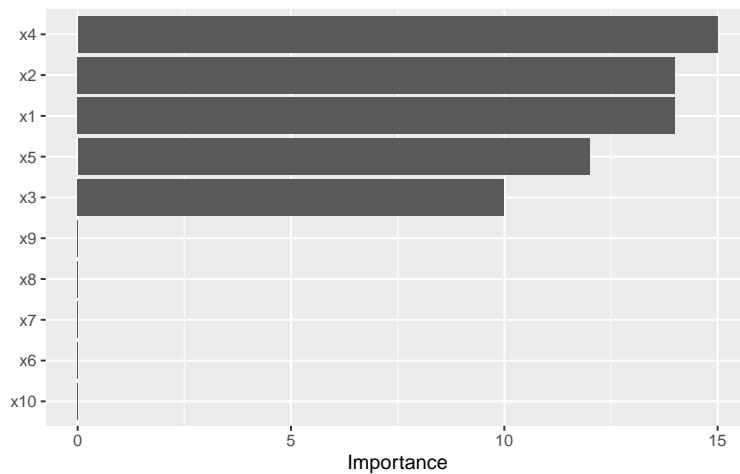


Figure 4: Example VIP from a MARS model fit to the simulated Friedman data.

To access VI scores directly in `earth`, you can use the `earth::evimp()` function.

Neural networks

For neural networks (NNs), two popular methods for constructing VI scores are the Garson algorithm (Garson, 1991), later modified by Goh (1995), and the Olden algorithm (Olden et al., 2004). For both algorithms, the basis of these VI scores is the network's connection weights. The Garson algorithm determines VI by identifying all weighted connections between the nodes of interest. Olden's algorithm, on the other hand, uses the products of the raw connection weights between each input and output neuron and sums these products across all hidden neurons. This has been shown to outperform the Garson method in various simulations. For DNNs, a similar method due to Gedeon (1997) considers the weights connecting the input features to the first two hidden layers (for simplicity and speed); but this method can be slow for large networks. We illustrate these two methods below using `vip()` with the `nnet` package (Ripley, 2016) (see the results in Figure 5).

```
# Load required packages
library(nnet)

# Fit a neural network
set.seed(0803) # for reproducibility
nn <- nnet(y ~ ., data = trn, size = 7, decay = 0.1,
            linout = TRUE, trace = FALSE)

# Construct VIPs
p1 <- vip(nn, type = "garson")
p2 <- vip(nn, type = "olden")
```

```
# Display plots in a grid (Figure 5)
grid.arrange(p1, p2, nrow = 1)
```

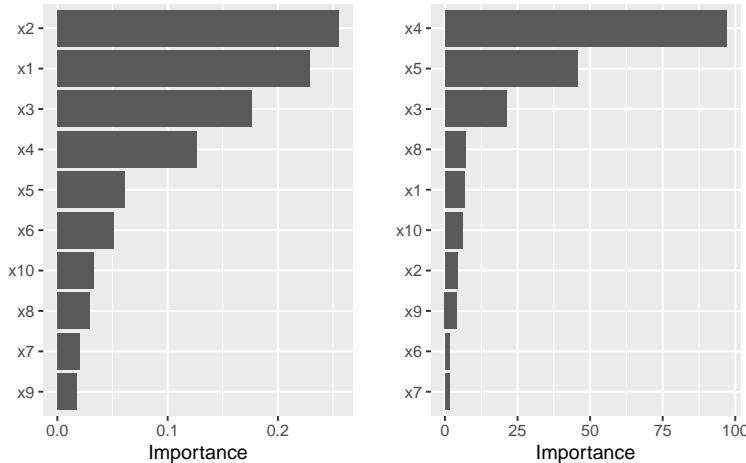


Figure 5: Example VIPs from a single-hidden-layer NN fit to the simulated Friedman data.

Model-agnostic VI

Model-agnostic interpretability separates interpretation from the model. Compared to model-specific approaches, model-agnostic VI methods are more flexible and can be applied to any supervised learning algorithm. In this section, we discuss model-agnostic methods for quantifying global feature importance using three different approaches: 1) a simple variance-based approach, 2) permutation-based feature importance, and 3) Shapley-based feature importance.

Variance-based methods

Our first model-agnostic method is based on a simple *feature importance ranking measure* (FIRM); for details, see Greenwell et al. (2018), Zien et al. (2009), and Scholbeck et al. (2019). The specific approach used here is based on quantifying the “flatness” of the effects of each feature.⁸ Feature effects can be assessed using *partial dependence plots* (PDPs) (Friedman, 2001) or *individual conditional expectation* (ICE) curves (Goldstein et al., 2015). PDPs and ICE curves help visualize the effect of low cardinality subsets of the feature space on the estimated prediction surface (e.g., main effects and two/three-way interaction effects.). They are also model-agnostic and can be constructed in the same way for any supervised learning algorithm. Below, we fit a *projection pursuit regression* (PPR) model (see `?stats::ppr` for details and references) and construct PDPs for each feature using the `pdp` package Greenwell (2017). The results are displayed in Figure 6. Notice how the PDPs for the uninformative features are relatively flat compared to the PDPs for features `x1`–`x5`.

Next, we compute PDP-based VI scores for the fitted PPR and NN models. The PDP method constructs VI scores that quantify the relative “flatness” of each PDP (by default, this is defined by computing the standard deviation of the y -axis values for each PDP). To use the PDP method, specify `method = "firm"` in the call to `vi()` or `vip()` (or just use `vi_firm()` directly):

```
# Fit a PPR model (nterms was chosen using the caret package with 5 repeats of
# 5-fold cross-validation)
pp <- ppr(y ~ ., data = trn, nterms = 11)

# Construct VIPs
p1 <- vip(pp, method = "firm") + ggtitle("PPR")
p2 <- vip(nn, method = "firm") + ggtitle("NN")

# Display plots in a grid (Figure 7)
grid.arrange(p1, p2, ncol = 2)
```

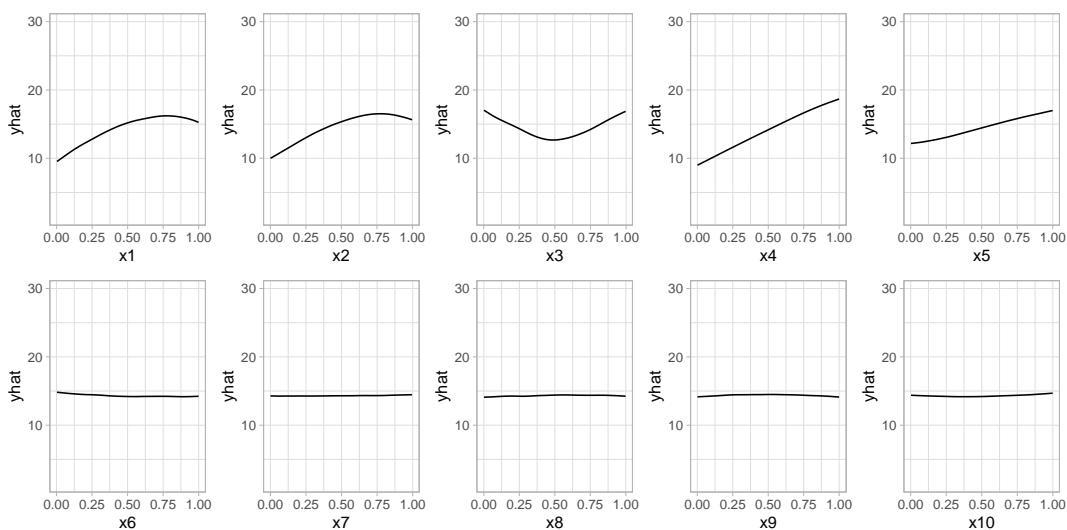


Figure 6: PDPs of main effects in the PPR model fit to the simulated Friedman data.

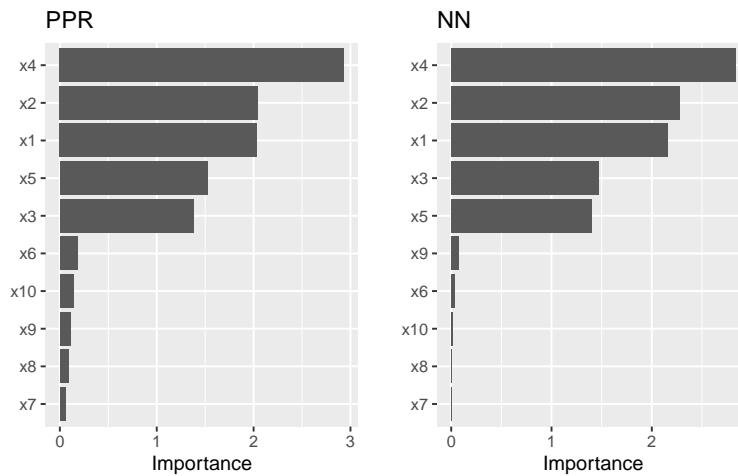


Figure 7: PDP-based feature importance for the PPR and NN models fit to the simulated Friedman data.

In Figure 7 we display the PDP-based feature importance for the previously obtained PPR and NN models. These VI scores essentially capture the variability in the partial dependence values for each main effect.

The ICE curve method is similar to the PDP method, except that we measure the “flatness” of each individual ICE curve and then aggregate the results (e.g., by averaging). If there are no (substantial) interaction effects, using ICE curves will produce results similar to using PDPs (which are just averaged ICE curves). However, if strong interaction effects are present, they can obfuscate the main effects and render the PDP-based approach less useful (since the PDPs for important features can be relatively flat when certain interactions are present; see Goldstein et al. (2015) for details). In fact, it is probably safest to always use ICE curves when employing the FIRM method.

Below, we display the ICE curves for each feature in the fitted PPR model using the same y -axis scale; see Figure 8. Again, there is a clear difference between the ICE curves for features x_1 – x_5 and x_6 – x_{10} ; the later being relatively flat by comparison. Also, notice how the ICE curves within each feature are relatively parallel (if the ICE curves within each feature were perfectly parallel, the standard deviation for each curve would be the same and the results will be identical to the PDP method). In this example, the interaction term between x_1 and x_2 does not obfuscate the PDPs for the main effects and the results are not much different.

Obtaining the ICE-based feature importance scores is also straightforward, just specify `ice = TRUE` when using the FIRM approach. This is illustrated in the code chunk below and the results, which are displayed in Figure 9, are similar to those obtained using the PDP method.

⁸A similar approach is taken in the `vivo` package (Kozak and Biecek, 2019).

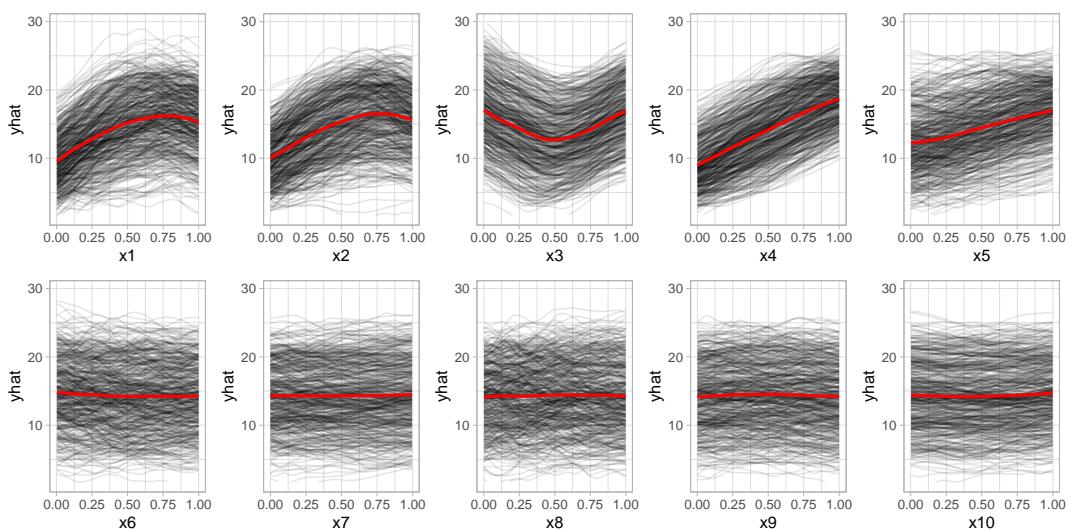


Figure 8: ICE curves for each feature in the PPR model fit to the simulated Friedman data. The red curve represents the PDP (i.e., the averaged ICE curves).

```
# Construct VIPs
p1 <- vip(pp, method = "firm", ice = TRUE) + ggtitle("PPR")
p2 <- vip(nn, method = "firm", ice = TRUE) + ggtitle("NN")

# Display plots in a grid (Figure 9)
grid.arrange(p1, p2, ncol = 2)
```

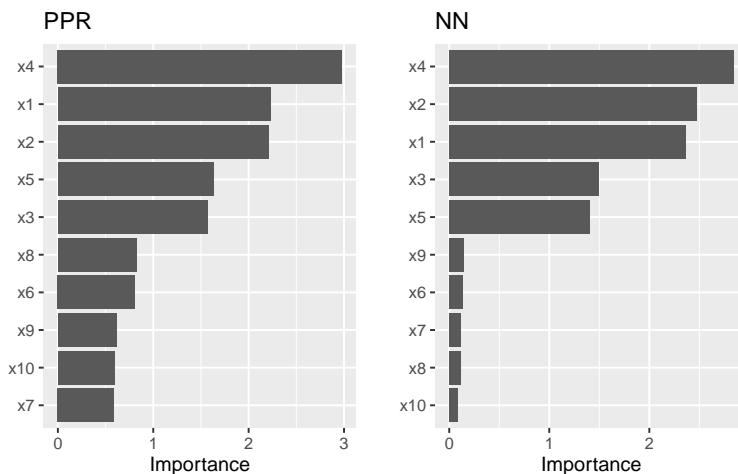


Figure 9: ICE-based feature importance for the PPR and NN models fit to the simulated Friedman data.

When using `method = "firm"`, the feature effect values are stored in an attribute called "`effects`". This is a convenience so that the feature effect plots (e.g., PDPs and ICE curves) can easily be reconstructed and compared with the VI scores, as demonstrated in the example below (see Figure 10):

```
# Construct PDP-based VI scores
(vis <- vi(pp, method = "firm"))

#> # A tibble: 10 x 2
#>   Variable Importance
#>   <chr>        <dbl>
#> 1 x4            2.93
#> 2 x2            2.05
#> 3 x1            2.04
#> 4 x5            1.53
```

```
#> 5 x3          1.38
#> 6 x6          0.183
#> 7 x10         0.139
#> 8 x9          0.113
#> 9 x8          0.0899
#> 10 x7         0.0558

# Reconstruct PDPs for all 10 features (Figure 10)
par(mfrow = c(2, 5))
for (name in paste0("x", 1:10)) {
  plot(attr(vis, which = "effects")[[name]], type = "l", ylim = c(9, 19), las = 1)
}
```

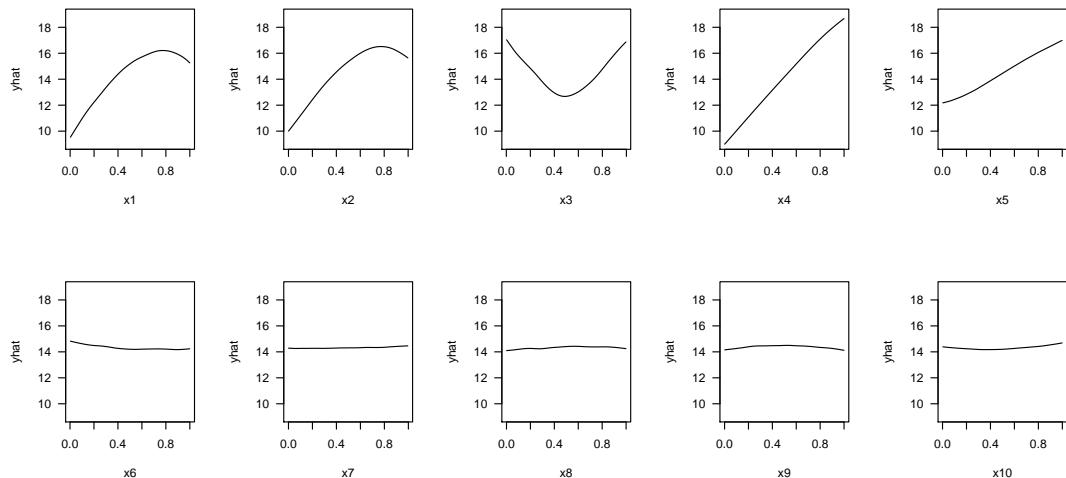


Figure 10: PDPs for all ten features reconstructed from the `pdp` attribute of the `vis` object.

Permutation method

The permutation method exists in various forms and was made popular in Breiman (2001) for RFs, before being generalized and extended in Fisher et al. (2018). The permutation approach used in `vip` is quite simple and is outlined in Algorithm 1 below. The idea is that if we randomly permute the values of an important feature in the training data, the training performance would degrade (since permuting the values of a feature effectively destroys any relationship between that feature and the target variable). This of course assumes that the model has been properly tuned (e.g., using cross-validation) and is not over fitting. The permutation approach uses the difference between some baseline performance measure (e.g., training R^2 , AUC, or RMSE) and the same performance measure obtained after permuting the values of a particular feature in the training data (**Note:** the model is NOT refit to the training data after randomly permuting the values of a feature). It is also important to note that this method may not be appropriate when you have, for example, highly correlated features (since permuting one feature at a time may lead to unlikely data instances).

Let X_1, X_2, \dots, X_j be the features of interest and let $\mathcal{M}_{\text{orig}}$ be the baseline performance metric for the trained model; for brevity, we'll assume smaller is better (e.g., classification error or RMSE). The permutation-based importance scores can be computed as follows:

1. For $i = 1, 2, \dots, j$:
 - (a) Permute the values of feature X_i in the training data.
 - (b) Recompute the performance metric on the permuted data $\mathcal{M}_{\text{perm}}$.
 - (c) Record the difference from baseline using $\text{imp}(X_i) = \mathcal{M}_{\text{perm}} - \mathcal{M}_{\text{orig}}$.
2. Return the VI scores $\text{imp}(X_1), \text{imp}(X_2), \dots, \text{imp}(X_j)$.

Algorithm 1: A simple algorithm for constructing permutation-based VI scores.

Algorithm 1 can be improved or modified in a number of ways. For instance, the process can be repeated several times and the results averaged together. This helps to provide more stable VI scores, and also the opportunity to measure their variability. Rather than taking the difference in step (c), Molnar (2019b, sec. 5.5.4) argues that using the ratio M_{perm}/M_{orig} makes the importance scores more comparable across different problems. It's also possible to assign importance scores to groups of features (e.g., by permuting more than one feature at a time); this would be useful if features can be categorized into mutually exclusive groups, for instance, categorical features that have been *one-hot-encoded*.

To use the permutation approach in `vip`, specify `method = "permute"` in the call to `vi()` or `vip()` (or you can use `vi_permute()` directly). Note that using `method = "permute"` requires specifying a few additional arguments (e.g., the training data, target name or vector of target values, a prediction function, etc.); see `?vi_permute` for details.

An example is given below for the previously fitted PPR and NN models. Here we use R^2 (`metric = "rsquared"`) as the evaluation metric. The results, which are displayed in Figure 11, agree with those obtained using the PDP- and ICE-based methods.

```
# Plot VI scores
set.seed(2021) # for reproducibility
p1 <- vip(pp, method = "permute", target = "y", metric = "rsquared",
           pred_wrapper = predict) + ggtitle("PPR")
p2 <- vip(nn, method = "permute", target = "y", metric = "rsquared",
           pred_wrapper = predict) + ggtitle("NN")

# Display plots in a grid (Figure 11)
grid.arrange(p1, p2, ncol = 2)
```

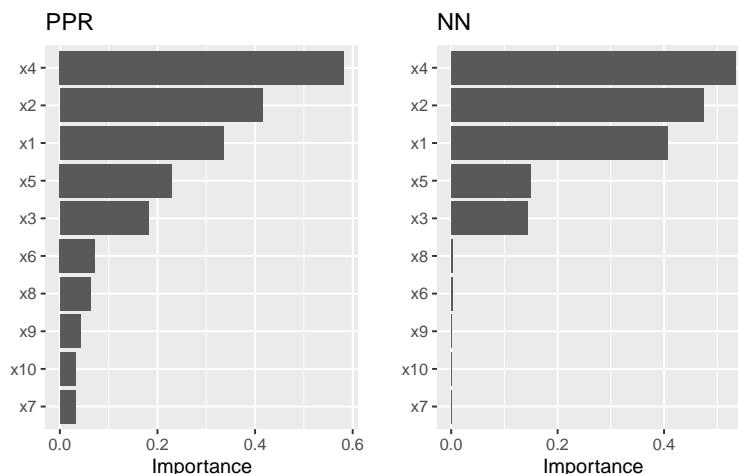


Figure 11: Permutation-based feature importance for the PPR and NN models fit to the simulated Friedman data.

The permutation approach introduces randomness into the procedure and therefore should be run more than once if computationally feasible. The upside to performing multiple runs of Algorithm 1 is that it allows us to compute standard errors (among other metrics) for the estimated VI scores, as illustrated in the example below; here we specify `nsim = 10` to request that each feature be permuted 10 times and the results averaged together. (Additionally, if `nsim > 1`, you can set `geom = "boxplot"` in the call to `vip()` to construct boxplots of the raw permutation-based VI scores. This is useful if you want to visualize the variability in each of the VI estimates; see Figure 12 for an example.)

```
# Use 10 Monte Carlo reps
set.seed(403) # for reproducibility
vis <- vi(pp, method = "permute", target = "y", metric = "rsquared",
           pred_wrapper = predict, nsim = 15)
vip(vis, geom = "boxplot") # Figure 12
```

All available performance metrics for regression and classification can be listed using the `list_metrics()` function, for example:

```
list_metrics()
```

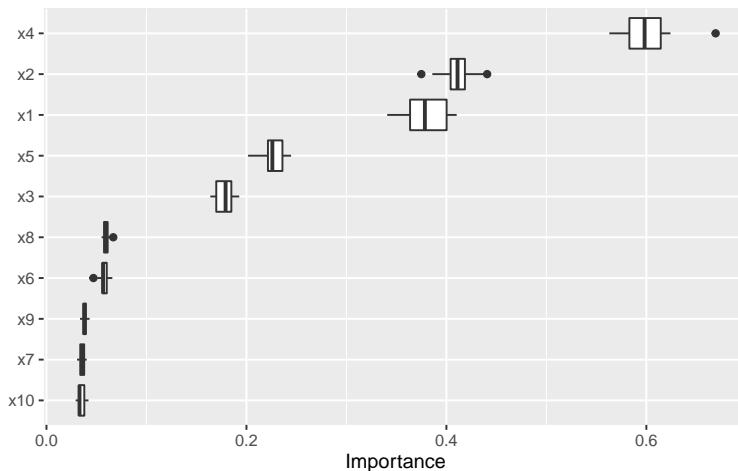


Figure 12: Boxplots of VI scores using the permutation method with 15 Monte Carlo repetitions.

#>	Metric	Description	Task
#> 1	accuracy	Classification accuracy	Binary/multiclass classification
#> 2	error	Misclassification error	Binary/multiclass classification
#> 3	auc	Area under (ROC) curve	Binary classification
#> 4	logloss	Log loss	Binary classification
#> 5	mauc	Multiclass area under (ROC) curve	Multiclass classification
#> 6	mae	Mean absolute error	Regression
#> 7	mse	Mean squared error	Regression
#> 8	r2	R squared	Regression
#> 9	rsquared	R squared	Regression
#> 10	rmse	Root mean squared error	Regression
#> 11	sse	Sum of squared errors	Regression

We can also use a custom metric (i.e., loss function). Suppose for example you want to measure importance using the *mean absolute error* (MAE):

$$MAE = \frac{1}{n} \sum_{i=1}^n |Y_i - \hat{f}(X_i)|, \quad (2)$$

where $\hat{f}(X_i)$ is the predicted value of Y_i . A simple function implementing this metric is given below (note that, according to the documentation in `?vi_permute`, user-supplied metric functions require two arguments: `actual` and `predicted`).

```
mae <- function(actual, predicted) {
  mean(abs(actual - predicted))
}
```

To use this for computing permutation-based VI scores just pass it via the `metric` argument (be warned, however, that the metric used for computing permutation importance should be the same as the metric used to train and tune the model). Also, since this is a custom metric, we need to specify whether a smaller value indicates better performance by setting `smaller_is_better = TRUE`. The results, which are displayed in Figure 13, are similar to those in Figure 11, albeit a different scale.

```
# Construct VIP (Figure 13)
set.seed(2321) # for reproducibility
pfun <- function(object, newdata) predict(object, newdata = newdata)
vip(nn, method = "permute", target = "y", metric = mae,
  smaller_is_better = TRUE, pred_wrapper = pfun) +
  ggtitle("Custom loss function: MAE")
```

Although permutation importance is most naturally computed on the training data, it may also be useful to do the shuffling and measure performance on new data! This is discussed in depth in Molnar (2019b, sec. 5.2). For users interested in computing permutation importance using new

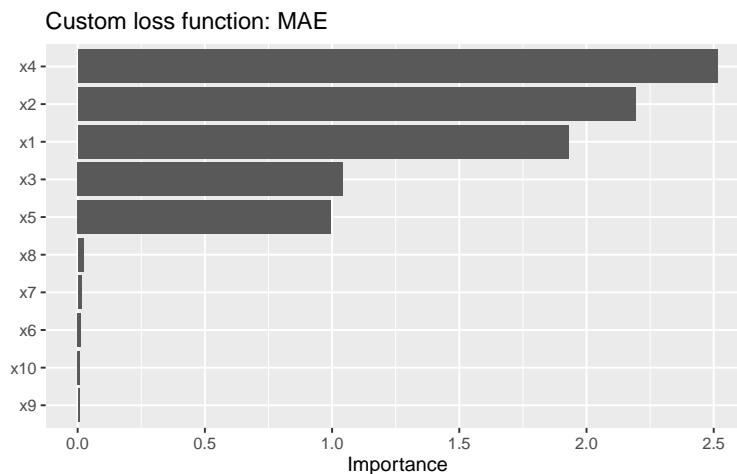


Figure 13: Permutation-based VI scores for the NN model fit to the simulated Friedman data. In this example, permutation importance is based on the MAE metric.

data, just supply it to the `train` argument in the call to `vi()`, `vip()`, or `vi_permute()`. For instance, suppose we wanted to only use a fraction of the original training data to carry out the computations. In this case, we could simply pass the sampled data to the `train` argument as follows:

```
# Construct VIP (Figure 14)
set.seed(2327) # for reproducibility
vip(nn, method = "permute", pred_wrapper = pfun, target = "y", metric = "rmse",
     train = trn[sample(nrow(trn), size = 400), ]) + # sample 400 observations
ggtitle("Using a random subset of training data")
```

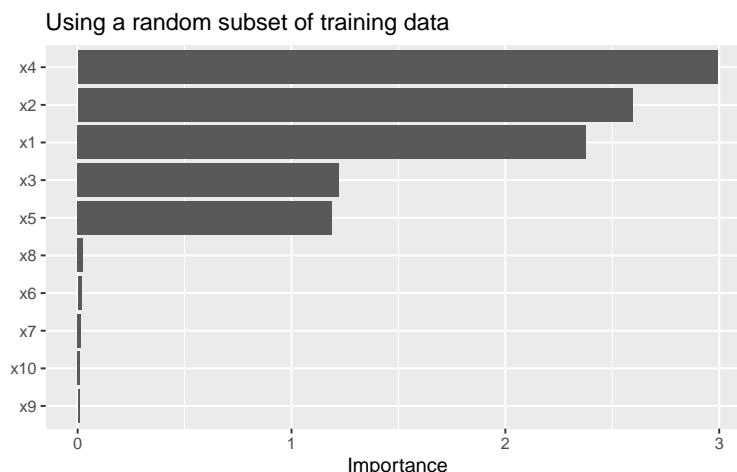


Figure 14: Permutation-based feature importance for the NN model fit to the simulated Friedman data. In this example, permutation importance is based on a random sample of 400 training observations.

When using the permutation method with `nsim > 1`, the default is to keep all the permutation scores as an attribute called `"raw_scores"`; you can turn this behavior off by setting `keep = FALSE` in the call to `vi_permute()`, `vi()`, or `vip()`. If `keep = TRUE` and `nsim > 1`, you can request all permutation scores to be plotted by setting `all_permutation = TRUE` in the call to `vip()`, as demonstrated in the code chunk below (see Figure 15). This also lets you visually inspect the variability in the permutation scores within each feature.

```
# Construct VIP (Figure 15)
set.seed(8264) # for reproducibility
vip(nn, method = "permute", pred_wrapper = pfun, target = "y", metric = "mae",
     nsim = 10, geom = "point", all_permutations = TRUE, jitter = TRUE) +
ggtitle("Plotting all permutation scores")
```

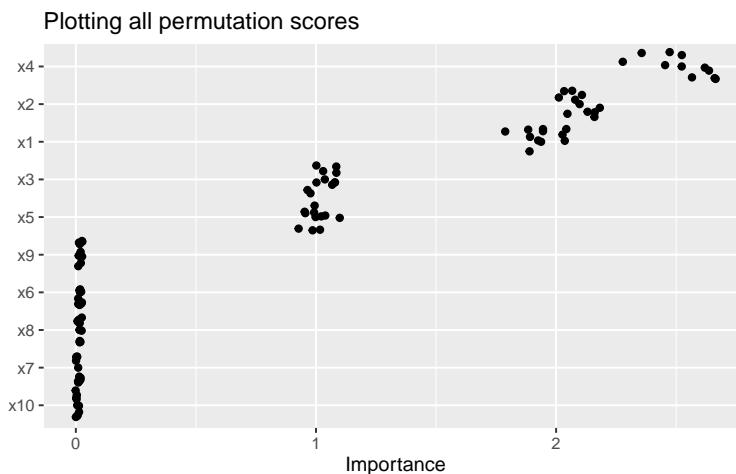


Figure 15: Permutation-based feature importance for the NN model fit to the simulated Friedman data. In this example, all the permutation importance scores (points) are displayed for each feature along with their average (bars).

Benchmarks

In this section, we compare the performance of four implementations of permutation-based VI scores: `iml::FeatureImp()` (version 0.10.0), `ingredients::feature_importance()` (version 1.3.1), `mmpf::permutationImportance` (version 0.0.5), and `vip::vi()` (version 0.2.2.9000).

We simulated 10,000 training observations from the Friedman 1 benchmark problem and trained a random forest using the `ranger` package. For each implementation, we computed permutation-based VI scores 100 times using the `microbenchmark` package (Mersmann, 2019). For this benchmark we did not use any of the parallel processing capability available in the `iml` and `vip` implementations. The results from `microbenchmark` are displayed in Figure 16 and summarized in the output below. In this case, the `vip` package (version 0.2.2.9000) was the fastest, followed closely by `ingredients` and `mmpf`. It should be noted, however, that the implementations in `vip` and `iml` can be parallelized. To the best of our knowledge, this is not the case for `ingredients` or `mmpf` (although it would not be difficult to write a simple parallel wrapper for either). The code used to generate these benchmarks can be found at <http://bit.ly/2TogXrq>.

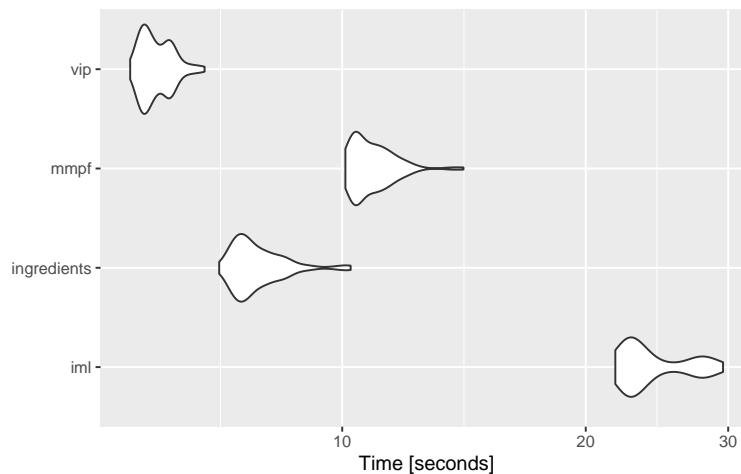


Figure 16: Violin plots comparing the computation time from three different implementations of permutation-based VI scores across 100 simulations.

Shapley method

Although `vip` focuses on global VI methods, it is becoming increasingly popular to assess global importance by aggregating local VI measures; in particular, *Shapley explanations* (Štrumbelj and Kononenko, 2014). Using Shapley values (a method from coalitional game theory), the prediction

for a single instance x^* can be explained by assuming that each feature value in x^* is a “player” in a game with a payout equal to the corresponding prediction $\hat{f}(x^*)$. Shapley values tell us how to fairly distribute the “payout” (i.e., prediction) among the features. Shapley values have become popular due to the attractive fairness properties they posses (Lundberg and Lee, 2017). The most popular implementation is available in the Python **shap** package (Lundberg and Lee, 2017); although a number of implementations are now available in R; for example, **iml**, **iBreakDown** (Biecek et al., 2019b), and **fastshap** (Greenwell, 2019).

Obtaining a global VI score from Shapley values requires aggregating the Shapley values for each feature across the entire training set (or at least a reasonable sample thereof). In particular, we use the mean of the absolute value of the individual Shapley values for each feature. Unfortunately, Shapley values can be computationally expensive, and therefore this approach may not be feasible for large training sets (say, >3000 observations). The **fastshap** package provides some relief by exploiting a few computational tricks, including the option to perform computations in parallel (see `?fastshap::explain` for details). Also, fast and exact algorithms (Lundberg et al., 2019) can be exploited for certain classes of models.

Starting with **vip** version 0.2.2.9000 you can now use `method = "shap"` in the call to `vi()` (or use `vi_shap()` directly) to compute global Shapley-based VI scores using the method described above (provided you have the **fastshap** package installed)—see `?vip::vi_shap` for details. To illustrate, we compute Shapley-based VI scores from an **xgboost** model (Chen et al., 2019) using the Friedman data from earlier; the results are displayed in Figure 17.⁹ (Note: specifying `include_type = TRUE` in the call to `vip()` causes the type of VI computed to be displayed as part of the axis label.)

```
# Load required packages
library(xgboost)

# Feature matrix
X <- data.matrix(subset(trn, select = -y)) # matrix of feature values

# Fit an XGBoost model; hyperparameters were tuned using 5-fold CV
set.seed(859) # for reproducibility
bst <- xgboost(X, label = trn$y, nrounds = 338, max_depth = 3, eta = 0.1,
                verbose = 0)

# Construct VIP (Figure 17)
vip(bst, method = "shap", train = X, exact = TRUE, include_type = TRUE)
```

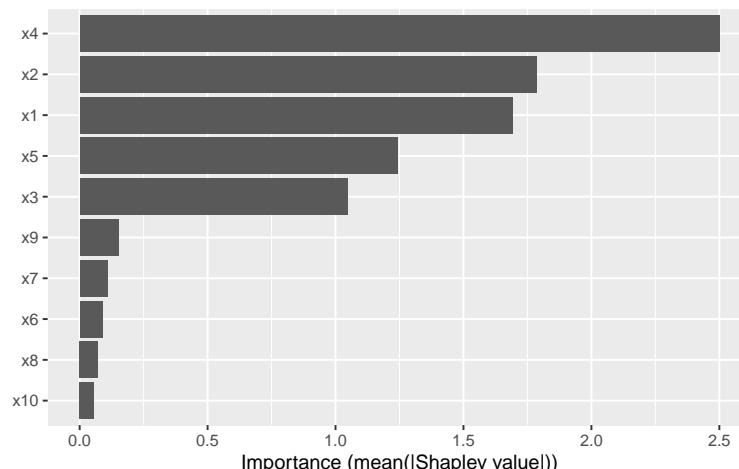


Figure 17: Shapley-based VI scores from an XGBoost model fit to the simulated Friedman data.

⁹Note that the `exact = TRUE` option is only available if you have **fastshap** version 0.0.4 or later

Drawbacks of existing methods

As discussed in Hooker and Mentch (2019), *permute-and-predict* methods—like PDPs, ICE curves, and permutation importance—can produce results that are highly misleading.¹⁰ For example, the standard approach to computing permutation-based VI scores involves independently permuting individual features. This implicitly makes the assumption that the observed features are statistically independent. In practice, however, features are often not independent which can lead to nonsensical VI scores. One way to mitigate this issue is to use the conditional approach described in Strobl et al. (2008); Hooker and Mentch (2019) provides additional alternatives, such as *permute-and-relearn importance*. Unfortunately, to the best of our knowledge, this approach is not yet available for general purpose. A similar modification can be applied to PDPs (Parr and Wilson, 2019)¹¹ which seems reasonable to use in the FIRM approach when strong dependencies among the features are present (though, we have not given this much thought or consideration).

We already mentioned that PDPs can be misleading in the presence of strong interaction effects. This drawback, of course, equally applies to the FIRM approach using PDPs for computing VI scores. As discussed earlier, this can be mitigated by using ICE curves instead. Another alternative would be to use *accumulated local effect* (ALE) plots (Apley and Zhu, 2016) (though we haven't really tested this idea). Compared to PDPs, ALE plots have the advantage of being faster to compute and less affected by strong dependencies among the features. The downside, however, is that ALE plots are more complicated to implement (hence, they are not currently available when using `method = "firm"`). ALE plots are available in the `ALEPlot` (Apley, 2018) and `iml` packages.

Hooker (2007) also argues that feature importance (which concern only *main effects*) can be misleading in high dimensional settings, especially when there are strong dependencies and interaction effects among the features, and suggests an approach based on a *generalized functional ANOVA decomposition*—though, to our knowledge, this approach is not widely implemented in open source.

Use sparklines to characterize feature effects

Starting with `vip` 0.1.3, we have included a new function `add_sparklines()` for constructing HTML-based VI tables; however, this feature requires the `DT` package (Xie et al., 2019). The primary difference between `vi()` and `add_sparklines()` is that the latter includes an `Effect` column that displays a sparkline representation of the partial dependence function for each feature. This is a concise way to display both feature importance and feature effect information in a single (interactive) table. See `?vip::add_sparklines` for details. We illustrate the basic use of `add_sparklines()` in the code chunk below where we fit a `ranger`-based random forest using the `mlr3` package (Lang et al., 2019).¹²

```
# Load required packages
library(mlr3)
library(mlr3learners)

# Fit a ranger-based random forest using the mlr3 package
set.seed(101)
task <- TaskRegr$new("friedman", backend = trn, target = "y")
lrnr <- lrn("regr.ranger", importance = "impurity")
lrnr$train(task)

# First, compute a tibble of VI scores using any method
var_imp <- vi(lrnr)

# Next, convert to an HTML-based data table with sparklines
add_sparklines(var_imp, fit = lrnr$model, train = trn) # Figure 18
```

¹⁰It's been argued that approximate Shapley values share the same drawback, however, Janzing et al. (2019) makes a compelling case against those arguments.

¹¹A basic R implementation is available at <https://github.com/bgreenwell/rstratx>.

¹²**Note:** Here we use the `...` argument to pass the original training to `pdp::partial()`; this is to avoid conflicts caused by `mlr3`'s `data.table` backend (Dowle and Srinivasan, 2019).

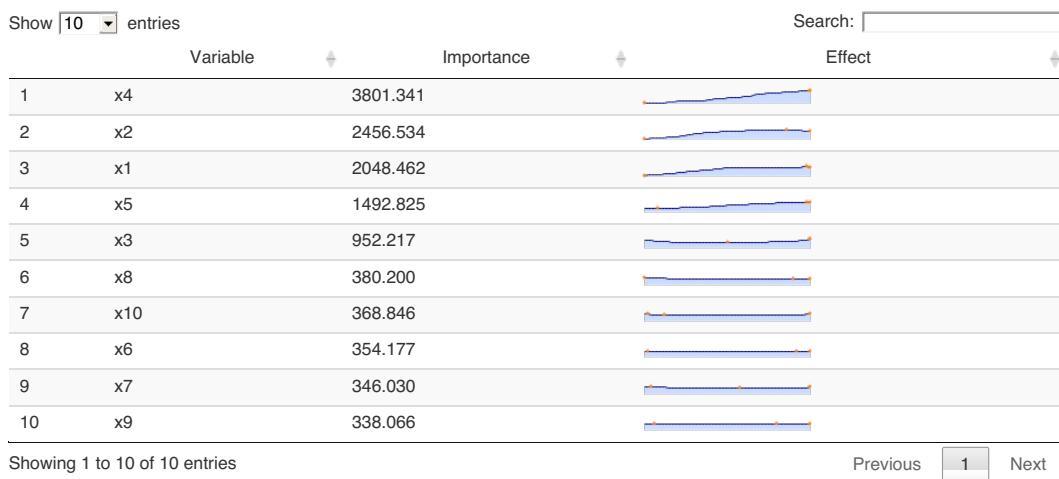


Figure 18: Variable importance scores along with a sparkline representation of feature effects.

Ames housing example

For illustration, we'll use the Ames housing data (Cock, 2011) which are available in the `AmesHousing` package (Kuhn, 2017). These data describe the sale of individual residential properties in Ames, Iowa from 2006–2010. The data set contains 2930 observations, 80 features (23 nominal, 23 ordinal, 14 discrete, and 20 continuous), and a continuous target giving the sale price of the home. The version we'll load is a cleaned up version of the original data set and treats all categorical variables as nominal (see `?AmesHousing::make_ames` for details).

Using the R package `SuperLearner` (Polley et al., 2019), we trained five models using 5-fold cross-validation: a GBM using the `xgboost` package, an RF using the `ranger` package, a MARS model using the `earth` package, a GLMNET model using the `glmnet` package (Friedman et al., 2019), and a support vector regression model using the `kernlab` package (Karatzoglou et al., 2019). The magnitude of the coefficients from the meta learner indicate which models contribute the most (if at all) to new predictions.

```
# Load the Ames housing data
ames <- AmesHousing::make_ames()
X <- subset(ames, select = -Sale_Price)
y <- ames$Sale_Price

# Load required packages
library(SuperLearner)

# List of base learners
learners <- c("SL.xgboost", "SL.ranger", "SL.earth", "SL.glmnet", "SL.ksvm")

# Stack models
set.seed(840) # for reproducibility
ctrl <- SuperLearner.CV.control(V = 5L, shuffle = TRUE)
sl <- SuperLearner(Y = y, X = X, SL.library = learners, verbose = TRUE,
                    cvControl = ctrl)
sl

#>
#> Call:
#> SuperLearner(Y = y, X = X, SL.library = learners, verbose = TRUE, cvControl = ctrl)
#>
#>
#>
#> Risk      Coef
#> SL.xgboost_All 580713682 0.41384425
#> SL.ranger_All 666208088 0.08083034
#> SL.earth_All 553872844 0.50532541
```

```
#> SL.glmnet_All    908881559 0.00000000
#> SL.ksvm_All     6784289108 0.00000000
```

In the code chunks below we request permutation-based VI scores and a sparkline representation of the PDPs for the top ten features. For this we need to define a couple of wrapper functions: one for computing predictions (for the permutation VI scores), and one for computing averaged predictions (for the PDPs).

```
# Prediction wrapper functions
imp_fun <- function(object, newdata) { # for permutation-based VI scores
  predict(object, newdata = newdata)$pred
}
par_fun <- function(object, newdata) { # for PDPs
  mean(predict(object, newdata = newdata)$pred)
}
```

To speed up the process, we perform the computations in parallel by setting `parallel = TRUE` in the calls to `vi()` and `add_sparklines()`. Note that we first need to set up a parallel backend for this to work. Both `vip` and `pdp` use `plyr` (Wickham, 2019)—which relies on `foreach`—so any parallel backend supported by the `foreach` package should work. Below we use a `socket` approach with the `doParallel` backend (Corporation and Weston, 2019) using a cluster of size five.

```
# Setup parallel backend
library(doParallel) # load the parallel backend
cl <- makeCluster(5) # use 5 workers
registerDoParallel(cl) # register the parallel backend

# Permutation-based feature importance
set.seed(278) # for reproducibility
var_imp <- vi(sl, method = "permute", train = X, target = y, metric = "rmse",
               pred_wrapper = imp_fun, nsim = 5, parallel = TRUE)

# Add sparkline representation of feature effects (# Figure 19)
add_sparklines(var_imp[1L:10L, ], fit = sl, pred.fun = par_fun, train = X,
               digits = 2, verbose = TRUE, trim.outliers = TRUE,
               grid.resolution = 20, parallel = TRUE)
```

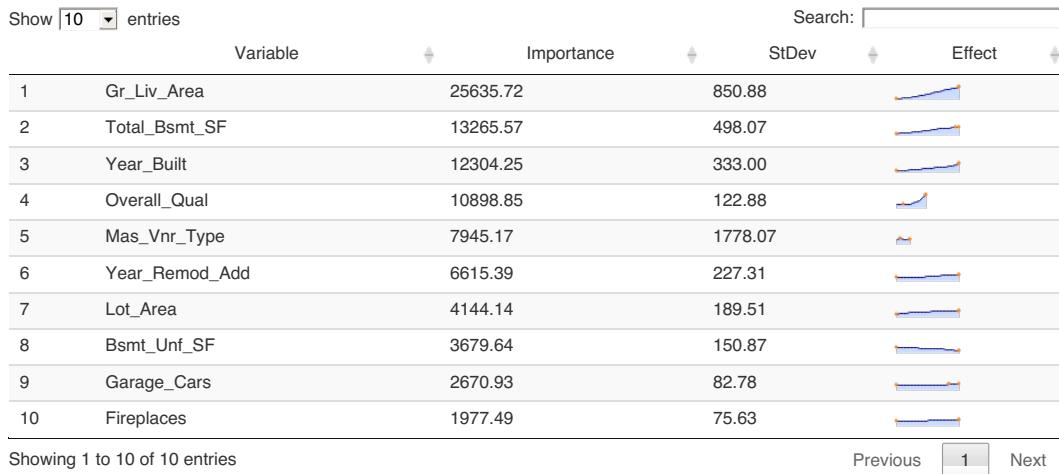


Figure 19: VIP with sparkline representation of feature effects for the top ten features from a Super Learner fit to the Ames housing data.

```
# Shut down cluster
stopCluster(cl)
```

Summary

VIPs help to visualize the strength of the relationship between each feature and the predicted response, while accounting for all the other features in the model. We've discussed two types of

VI: model-specific and model-agnostic, as well as some of their strengths and weaknesses. In this paper, we showed how to construct VIPs for various types of “black box” models in R using the **vip** package. We also briefly discussed related approaches available in a number of other R packages. Suggestions to avoid high execution times were discussed and demonstrated via examples. This paper is based on **vip** version 0.2.2.9000. In terms of future development, **vip** can be expanded in a number of ways. For example, we plan to incorporate the option to compute group-based and conditional permutation scores. Although not discussed in this paper, **vip** also includes a promising statistic (similar to the variance-based VI scores previously discussed) for measuring the relative strength of interaction between features. Although VIPs can help understand which features are driving the model’s predictions, ML practitioners should be cognizant of the fact that none of the methods discussed in this paper are uniformly best across all situations; they require an accurate model that has been properly tuned, and should be checked for consistency with human domain knowledge.

Acknowledgments

The authors would like to thank the anonymous reviewers and the Editor for their helpful comments and suggestions. We would also like to thank the members of the 84.51° Interpretable Machine Learning Special Interest Group for their thoughtful discussions on the topics discussed herein.

Bibliography

- D. Apley. *ALEPlot: Accumulated Local Effects (ALE) Plots and Partial Dependence (PD) Plots*, 2018. URL <https://CRAN.R-project.org/package=ALEPlot>. R package version 1.1. [p]
- D. W. Apley and J. Zhu. Visualizing the effects of predictor variables in black box supervised learning models, 2016. [p]
- D. Bau, B. Zhou, A. Khosla, A. Oliva, and A. Torralba. Network dissection: Quantifying interpretability of deep visual representations. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017. [p]
- A. Bibal and B. Frénay. Interpretability of machine learning models and representations: an introduction. In *ESANN*, 2016. [p]
- P. Biecek. *DALEX: Descriptive mAchine Learning EXplanations*, 2019. URL <https://CRAN.R-project.org/package=DALEX>. R package version 0.4.9. [p]
- P. Biecek, H. Baniecki, and A. Izdebski. *ingredients: Effects and Importances of Model Ingredients*, 2019a. URL <https://CRAN.R-project.org/package=ingredients>. R package version 0.5.0. [p]
- P. Biecek, A. Gosiewska, H. Baniecki, and A. Izdebski. *iBreakDown: Model Agnostic Instance Level Variable Attributions*, 2019b. URL <https://CRAN.R-project.org/package=iBreakDown>. R package version 0.9.9. [p]
- B. Bischl, M. Lang, L. Kotthoff, J. Schiffner, J. Richter, Z. Jones, G. Casalicchio, M. Gallo, and P. Schratz. *mlr: Machine Learning in R*, 2020. URL <https://CRAN.R-project.org/package=mlr>. R package version 2.17.0. [p]
- L. Breiman. Bagging predictors. *Machine Learning*, 8(2):209–218, 1996. URL <https://doi.org/10.1023/A:1018054314350>. [p]
- L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001. URL <https://doi.org/10.1023/A:1010933404324>. [p]
- L. Breiman, J. Friedman, and R. A. O. Charles J. Stone. *Classification and Regression Trees*. The Wadsworth and Brooks-Cole statistics-probability series. Taylor & Francis, 1984. ISBN 9780412048418. [p]
- R. Caruana, Y. Lou, J. Gehrke, P. Koch, M. Sturm, and N. Elhadad. Intelligible models for healthcare: Predicting pneumonia risk and hospital 30-day readmission. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’15, pages 1721–1730, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450336642. URL <https://doi.org/10.1145/2783258.2788613>. [p]

- E. Celik. *vita: Variable Importance Testing Approaches*, 2015. URL <https://CRAN.R-project.org/package=vita>. R package version 1.0.0. [p]
- W. Chang. *R6: Encapsulated Classes with Reference Semantics*, 2019. URL <https://CRAN.R-project.org/package=R6>. R package version 2.4.1. [p]
- T. Chen, T. He, M. Benesty, V. Khotilovich, Y. Tang, H. Cho, K. Chen, R. Mitchell, I. Cano, T. Zhou, M. Li, J. Xie, M. Lin, Y. Geng, and Y. Li. *xgboost: Extreme Gradient Boosting*, 2019. URL <https://CRAN.R-project.org/package=xgboost>. R package version 0.90.0.2. [p]
- W. S. Cleveland. Robust locally weighted regression and smoothing scatterplots. *Journal of the American Statistical Association*, 74(368):829–836, 1979. doi: <https://doi.org/10.1080/01621459.1979.10481038>. [p]
- D. D. Cock. Ames, iowa: Alternative to the boston housing data as an end of semester regression project. *Journal of Statistics Education*, 19(3):1–15, 2011. URL <https://doi.org/10.1080/10691898.2011.11889627>. [p]
- M. Corporation and S. Weston. *doParallel: Foreach Parallel Adaptor for the 'parallel' Package*, 2019. URL <https://CRAN.R-project.org/package=doParallel>. R package version 1.0.15. [p]
- G. Csárdi and M. Salmon. *pkgsearch: Search and Query CRAN R Packages*, 2019. URL <https://CRAN.R-project.org/package=pkgsearch>. R package version 3.0.2. [p]
- F. Doshi-Velez and B. Kim. Towards a rigorous science of interpretable machine learning, 2017. [p]
- M. Dowle and A. Srinivasan. *data.table: Extension of 'data.frame'*, 2019. URL <https://CRAN.R-project.org/package=data.table>. R package version 1.12.8. [p]
- A. Fisher, C. Rudin, and F. Dominici. Model class reliance: Variable importance measures for any machine learning model class, from the "rashomon" perspective. *arXiv preprint arXiv:1801.01489*, 2018. [p]
- J. Friedman, T. Hastie, R. Tibshirani, B. Narasimhan, and N. Simon. *glmnet: Lasso and Elastic-Net Regularized Generalized Linear Models*, 2019. URL <https://CRAN.R-project.org/package=glmnet>. R package version 3.0-2. [p]
- J. H. Friedman. Multivariate adaptive regression splines. *The Annals of Statistics*, 19(1):1–67, 1991. URL <https://doi.org/10.1214/aos/1176347963>. [p]
- J. H. Friedman. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29(5):1189–1232, 2001. URL <https://doi.org/10.1214/aos/1013203451>. [p]
- D. G. Garson. Interpreting neural-network connection weights. *Artificial Intelligence Expert*, 6(4):46–51, 1991. [p]
- T. Gedeon. Data mining of inputs: Analysing magnitude and functional measures. *International Journal of Neural Systems*, 24(2):123–140, 1997. URL <https://doi.org/10.1007/s10994-006-6226-1>. [p]
- A. Goh. Back-propagation neural networks for modeling complex systems. *Artificial Intelligence in Engineering*, 9(3):143–151, 1995. URL [https://dx.doi.org/10.1016/0954-1810\(94\)00011-S](https://dx.doi.org/10.1016/0954-1810(94)00011-S). [p]
- A. Goldstein, A. Kapelner, J. Bleich, and E. Pitkin. Peeking inside the black box: Visualizing statistical learning with plots of individual conditional expectation. *Journal of Computational and Graphical Statistics*, 24(1):44–65, 2015. URL <https://doi.org/10.1080/10618600.2014.907095>. [p]
- B. Greenwell. *fastshap: Fast Approximate Shapley Values*, 2019. URL <https://github.com/bgreenwell/fastshap>. R package version 0.0.3.9000. [p]
- B. Greenwell, B. Boehmke, and B. Gray. *vip: Variable Importance Plots*, 2019. URL <https://github.com/koalaverse/vip/>. R package version 0.2.1. [p]
- B. M. Greenwell. pdp: An r package for constructing partial dependence plots. *The R Journal*, 9(1):421–436, 2017. URL <https://journal.r-project.org/archive/2017/RJ-2017-016/index.html>. [p]

- B. M. Greenwell, B. C. Boehmke, and A. J. McCarthy. A simple and effective model-based variable importance measure. *arXiv preprint arXiv:1805.04755*, 2018. [p]
- T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition*. Springer Series in Statistics. Springer-Verlag, 2009. [p]
- G. Hooker. Generalized functional anova diagnostics for high-dimensional functions of dependent variables. *Journal of Computational and Graphical Statistics*, 16(3):709–732, 2007. URL <https://doi.org/10.1198/106186007X237892>. [p]
- G. Hooker and L. Mentch. Please stop permuting features: An explanation and alternatives, 2019. [p]
- T. Hothorn and A. Zeileis. *partykit: A Toolkit for Recursive Partitioning*, 2019. URL <https://CRAN.R-project.org/package=partykit>. R package version 1.2-5. [p]
- T. Hothorn, K. Hornik, C. Strobl, and A. Zeileis. *party: A Laboratory for Recursive Partitioning*, 2019. URL <https://CRAN.R-project.org/package=party>. R package version 1.3-3. [p]
- D. Janzing, L. Minorics, and P. Blöbaum. Feature relevance quantification in explainable ai: A causal problem, 2019. [p]
- Z. Jones. *mmpf: Monte-Carlo Methods for Prediction Functions*, 2018. URL <https://CRAN.R-project.org/package=mmpf>. R package version 0.0.5. [p]
- A. Karatzoglou, A. Smola, and K. Hornik. *kernlab: Kernel-Based Machine Learning Lab*, 2019. URL <https://CRAN.R-project.org/package=kernlab>. R package version 0.9-29. [p]
- A. Kozak and P. Biecek. *vivo: Local Variable Importance via Oscillations of Ceteris Paribus Profiles*, 2019. URL <https://CRAN.R-project.org/package=vivo>. R package version 0.1.1. [p]
- M. Kuhn. *AmesHousing: The Ames Iowa Housing Data*, 2017. URL <https://CRAN.R-project.org/package=AmesHousing>. R package version 0.0.3. [p]
- M. Kuhn. *caret: Classification and Regression Training*, 2020. URL <https://CRAN.R-project.org/package=caret>. R package version 6.0-85. [p]
- M. Kuhn and K. Johnson. *Applied Predictive Modeling*. SpringerLink : Bücher. Springer New York, 2013. ISBN 9781461468493. [p]
- M. Lang, B. Bischl, J. Richter, P. Schratz, and M. Binder. *mlr3: Machine Learning in R - Next Generation*, 2019. URL <https://CRAN.R-project.org/package=mlr3>. R package version 0.1.6. [p]
- M. Loecher. *rfVarImpOOB: Unbiased Variable Importance for Random Forests*, 2019. URL <https://CRAN.R-project.org/package=rfVarImpOOB>. R package version 1.0. [p]
- S. M. Lundberg and S.-I. Lee. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems 30*, pages 4765–4774. Curran Associates, Inc., 2017. URL <http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf>. [p]
- S. M. Lundberg, G. Erion, H. Chen, A. DeGrave, J. M. Prutkin, B. Nair, R. Katz, J. Himmelfarb, N. Bansal, and S.-I. Lee. Explainable ai for trees: From local explanations to global understanding. *arXiv preprint arXiv:1905.04610*, 2019. [p]
- O. Mersmann. *microbenchmark: Accurate Timing Functions*, 2019. URL <https://CRAN.R-project.org/package=microbenchmark>. R package version 1.4-7. [p]
- S. Milborrow. *earth: Multivariate Adaptive Regression Splines*, 2019. URL <https://CRAN.R-project.org/package=earth>. R package version 5.1.2. [p]
- C. Molnar. *iml: Interpretable Machine Learning*, 2019a. URL <https://CRAN.R-project.org/package=iml>. R package version 0.9.0. [p]
- C. Molnar. *Interpretable Machine Learning*. 2019b. <https://christophm.github.io/interpretable-ml-book/>. [p]

- G. Montavon, W. Samek, and K.-R. Müller. Methods for interpreting and understanding deep neural networks. *Digital Signal Processing*, 73:1–15, 2018. URL <https://doi.org/10.1016/j.dsp.2017.10.011>. [p]
- K. Müller and H. Wickham. *tibble: Simple Data Frames*, 2019. URL <https://CRAN.R-project.org/package=tibble>. R package version 2.1.3. [p]
- J. D. Olden, M. K. Joy, and R. G. Death. An accurate comparison of methods for quantifying variable importance in artificial neural networks using simulated data. *Ecological Modelling*, 178(3):389–397, 2004. URL <https://dx.doi.org/10.1016/j.ecolmodel.2004.03.013>. [p]
- A. Paluszynska, P. Biecek, and Y. Jiang. *randomForestExplainer: Explaining and Visualizing Random Forests in Terms of Variable Importance*, 2019. URL <https://CRAN.R-project.org/package=randomForestExplainer>. R package version 0.10.0. [p]
- T. Parr and J. D. Wilson. Technical report: A stratification approach to partial dependence for codependent variables, 2019. [p]
- E. Polley, E. LeDell, C. Kennedy, and M. van der Laan. *SuperLearner: Super Learner Prediction*, 2019. URL <https://CRAN.R-project.org/package=SuperLearner>. R package version 2.0-26. [p]
- B. Poulin, R. Eisner, D. Szafron, P. Lu, R. Greiner, D. S. Wishart, A. Fyshe, B. Pearcy, C. MacDonell, and J. Anvik. Visual explanation of evidence in additive classifiers. In *Proceedings of the 18th Conference on Innovative Applications of Artificial Intelligence - Volume 2*, IAAI’ 06, pages 1822–1829. AAAI Press, 2006. [p]
- P. Probst. *measures: Performance Measures for Statistical Learning*, 2018. URL <https://CRAN.R-project.org/package=measures>. R package version 0.2. [p]
- P. Probst. *varImp: RF Variable Importance for Arbitrary Measures*, 2019. URL <https://CRAN.R-project.org/package=varImp>. R package version 0.3. [p]
- Revolution Analytics and S. Weston. *foreach: Provides Foreach Looping Construct*. [p]
- B. Ripley. *nnet: Feed-Forward Neural Networks and Multinomial Log-Linear Models*, 2016. URL <https://CRAN.R-project.org/package=nnet>. R package version 7.3-12. [p]
- C. A. Scholbeck, C. Molnar, C. Heumann, B. Bischl, and G. Casalicchio. Sampling, intervention, prediction, aggregation: A generalized framework for model agnostic interpretations. *CoRR*, abs/1904.03959, 2019. URL <http://arxiv.org/abs/1904.03959>. [p]
- C. Strobl, A.-L. Boulesteix, A. Zeileis, and T. Hothorn. Bias in random forest variable importance measures: Illustrations, sources and a solution. *BMC Bioinformatics*, 8(25), 2007. URL <http://www.biomedcentral.com/1471-2105/8/25>. [p]
- C. Strobl, A.-L. Boulesteix, T. Kneib, T. Augustin, and A. Zeileis. Conditional variable importance for random forests. *BMC Bioinformatics*, 9(1):307, 2008. URL <https://doi.org/10.1186/1471-2105-9-307>. [p]
- Q. Sun. *tree.interpreter: Random Forest Prediction Decomposition and Feature Importance Measure*, 2019. URL <https://CRAN.R-project.org/package=tree.interpreter>. R package version 0.1.0. [p]
- M. van der Laan. Statistical inference for variable importance. *The International Journal of Biostatistics*, 2(1), 2006. URL <https://doi.org/10.2202/1557-4679.1008>. [p]
- H. Wickham. *plyr: Tools for Splitting, Applying and Combining Data*, 2019. URL <https://CRAN.R-project.org/package=plyr>. R package version 1.8.5. [p]
- H. Wickham, W. Chang, L. Henry, T. L. Pedersen, K. Takahashi, C. Wilke, K. Woo, and H. Yutani. *ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics*, 2019. URL <https://CRAN.R-project.org/package=ggplot2>. R package version 3.2.1. [p]
- M. N. Wright, S. Wager, and P. Probst. *ranger: A Fast Implementation of Random Forests*, 2020. URL <https://CRAN.R-project.org/package=ranger>. R package version 0.12.1. [p]
- Y. Xie, J. Cheng, and X. Tan. *DT: A Wrapper of the JavaScript Library 'DataTables'*, 2019. URL <https://CRAN.R-project.org/package=DT>. R package version 0.11. [p]

- A. Zien, N. Kraemer, S. Sonnenburg, and G. Raetsch. The feature importance ranking measure, 2009. [p]
- E. Štrumbelj and I. Kononenko. Explaining prediction models and individual predictions with feature contributions. *Knowledge and Information Systems*, 31(3):647–665, 2014. URL <https://doi.org/10.1007/s10115-013-0679-x>. [p]

Brandon M. Greenwell
University of Cincinnati
2925 Campus Green Dr
Cincinnati, OH 45221
United States of America
ORCID—0000-0002-8120-0084
greenwell.brandon@gmail.com

Bradley C. Boehmke
University of Cincinnati
2925 Campus Green Dr
Cincinnati, OH 45221
United States of America
ORCID—0000-0002-3611-8516
bradleyboehmke@gmail.com

Individual-Level Modelling of Infectious Disease Data: EpiILM

by Vineetha Warriyar K. V., Waleed Almutiry and Rob Deardon

Abstract In this article we introduce the R package **EpiILM**, which provides tools for simulation from, and inference for, discrete-time individual-level models of infectious disease transmission proposed by Deardon et al. (2010). The inference is set in a Bayesian framework and is carried out via Metropolis-Hastings Markov chain Monte Carlo (MCMC). For its fast implementation, key functions are coded in Fortran. Both spatial and contact network models are implemented in the package and can be set in either susceptible-infected (SI) or susceptible-infected-removed (SIR) compartmental frameworks. Use of the package is demonstrated through examples involving both simulated and real data.

Introduction

The task of modelling infectious disease transmission through a population poses a number of challenges. One challenge is that successfully modelling many, if not most, infectious disease systems requires accounting for complex heterogeneities within the population. These heterogeneities may be characterized by individual-level covariates, spatial clustering, or the existence of complex contact networks through which the disease may propagate. A second challenge is that there are inherent dependencies in infection (or event) times.

To model such scenarios, Deardon et al. (2010) introduced a class of discrete time individual-level models (ILMs), fitting the models to data in a Bayesian Markov chain Monte Carlo (MCMC) framework. They applied spatial ILMs to the UK foot-and-mouth disease (FMD) epidemic of 2001, which accounted for farm-level covariates such as the number and type of animals on each farm. However, the ILM class also allows for the incorporation of contact networks through which disease can spread. Once fitted, such models can be used to predict the course of an epidemic (e.g., O'Reilly et al., 2018) or test the effectiveness of various control strategies (e.g., Tildesley et al., 2006) that can be imposed upon epidemics simulated from the fitted model.

A third challenge when modelling disease systems is that very little software so far has been made available that allows for simulation from, and especially inference for, individual-level models of disease transmission. Most inference for such models is carried out in fast, low-level languages such as Fortran or variants of C, which makes it difficult for researchers (e.g., public health epidemiologists) without a strong background in computational statistics and programming to make use of the models.

A number of R packages have recently been developed for modelling infectious disease systems (e.g., **R0** (Boelle and Obadia, 2015), **EpiEstim** (Cori, 2019), **EpiModel** (Jenness et al., 2018), and **epinet** (Groendyke and Welch, 2016)). Most of these packages can be used to carry out epidemic simulation from given models; in addition, **R0** or **EpiEstim**, for example, can be used to calculate the (basic) reproduction number under various scenarios. The **EpiModel** package allows for the simulation of epidemics from stochastic models, primarily exponential-family random graph models (ERGMs), and provides tools for analyzing simulation output. Functions for carrying out some limited forms of inference are also provided. Another widely used package for monitoring and modelling infectious disease spread through surveillance data is **surveillance** (Meyer et al., 2017). This package provides for a highly flexible modelling framework for such data. However, the package does not cover mechanistic, individual-level disease transmission models such as those of Deardon et al. (2010).

Here, we detail a novel R statistical software package **EpiILM** (Warriyar. K. V. et al., 2020) for simulating from, and carrying out Bayesian MCMC-based statistical inference for spatial and/or network-based models in the Deardon et al. (2010) individual-level modelling framework. The package allows for the incorporation of individual-level susceptibility and transmissibility covariates in models, provides various methods of summarizing epidemic data sets, and permits reasonably involved scenarios to be coded up by the user due to its setting in an R framework. The main functions, including for likelihood calculation are coded in Fortran in order to achieve the goal of agile implementation.

The type of spatial and network-based transmission models that **EpiILM** facilitates can be used to model a wide range of disease systems, as well as other transmissible processes. Human diseases such as influenza, measles or HIV, tend to be transmitted via interactions which can be captured by contact networks. For example, Malik et al. (2014) used a network representing whether two

people shared the same household for modelling influenza spread in Hong Kong. Networks can also be used to characterize social or sexual relationships.

In the livestock industries, diseases are often transmitted from farm to farm via supply trucks or animal movements from farm to farm, or from farm to market. For example, ILM's were used by Kwong et al. (2013) to model the spread of porcine reproductive and respiratory syndrome (PRRS) through Ontario swine farms via such mechanisms. Spatial mechanisms are also often important in livestock industries (e.g., Jewell et al., 2009; Deardon et al., 2010; Kwong et al., 2013), as well as for modelling crop diseases (e.g., Pokharel and Deardon, 2016), since airborne spread is often a key factor.

Further, these types of models can also be used to model transmissible processes other than infectious disease spread. For example, Cook et al. (2007) used similar models to model the transmission of alien species through a landscape; specifically, giant hogweed in the UK. In addition, Vrbik et al. (2012) used spatial ILM's to model fire spread. They looked at fire spread under controlled conditions, but such models would likely be useful for modelling the spread of forest fires since important covariates such as vegetation-type could be incorporated into the models.

Data from infectious disease systems are generally 'time-to-event', typically involving multiple states. However, standard survival models (e.g., Cox (1972), Therneau (2015)) or multi-state time-to-event models (e.g., see Jackson (2011)) are not applicable here, because in an infectious disease system individual event times cannot be assumed independent even after conditioning on covariates. That is, my risk of contracting and infectious disease generally depends upon the disease state of other individuals in the population; this is not typically the case for most cancers, for example to which more standard models can be applied.

The remainder of this paper is structured as follows: Section 2 explains the relevant models involved in the package; Section 3 describes the contents of the package along with some illustrative examples; and Section 4 concludes the paper with a brief discussion on future development.

Model

In our **EpiILM** package, we consider two compartmental frameworks: susceptible-infectious (SI) and susceptible-infectious-removed (SIR). In the former framework, individuals begin in the susceptible state (S) and if/when infected become immediately infectious (I) and remain in that state indefinitely. In the latter framework, individuals once infected remain infectious for some time interval before entering the removed state (R). This final state might represent death, quarantine, or recovery accompanied by immunity. We consider discrete time scenarios so a complete epidemic history is represented by $t = 1, 2, \dots, t_{end}$, where (typically) $t = 1$ is the time when the first infection is observed and t_{end} is the time when the epidemic ends. Hence, for a given time point t , an individual i belongs to one, and only one, of the sets $S(t)$ or $I(t)$ if the compartmental framework is SI, and i belongs to one, and only one, of the sets $S(t)$, $I(t)$, or $R(t)$ if the compartmental framework is SIR.

Under either framework, the probability that a susceptible individual i is infected at time point t is given by $\text{IP}(i, t)$ as follows:

$$\text{IP}(i, t) = 1 - \exp\{-\Omega_S(i) \sum_{j \in I(t)} \Omega_T(j) \kappa(ij) - \varepsilon\}, \quad \Omega_S(i) > 0, \quad \Omega_T(j) > 0, \quad \varepsilon > 0 \quad (1)$$

where: $\Omega_S(i)$ is a susceptibility function that accommodates potential risk factors associated with susceptible individual i contracting the disease; $\Omega_T(j)$ is a transmissibility function that accommodates potential risk factors associated with infectious individual j contracting the disease; ε is a sparks term which represents infections originating from outside the population being observed or some other unobserved infection mechanism; and $\kappa(i, j)$ is an infection kernel function that represents the shared risk factors between pairs of infectious and susceptible individuals.

The susceptibility function can incorporate any individual-level covariates of interest, such as age, genetic factors, vaccination status, and so on. In Equation (1), $\Omega_S(i)$ is treated as a linear function of the covariates, i.e., $\Omega_S(i) = \alpha_0 + \alpha_1 X_1(i) + \alpha_2 X_2(i) + \dots + \alpha_{n_s} X_{n_s}(i)$, where $X_1(i), \dots, X_{n_s}(i)$ denote n_s covariates associated with susceptible individual i , along with susceptibility parameters $\alpha_0, \dots, \alpha_{n_s} > 0$. Note that, if the model does not contain any susceptibility covariates then $\Omega_S(i) = \alpha_0$ is used. In a similar way, the transmissibility function in Equation (1) can incorporate any individual-level covariates of interest associated with infectious individual. $\Omega_T(j)$ is also treated as a linear function of the covariates, but without the intercept term, i.e., $\Omega_T(j) = \phi_1 X_1(j) + \phi_2 X_2(j) + \dots + \phi_{n_t} X_{n_t}(j)$, where $X_1(j), \dots, X_{n_t}(j)$ denote the n_t covariates associated with infectious individual j , along with transmissibility parameters $\phi_1, \dots, \phi_{n_t} > 0$. Also

note that if the model does not contain any transmissibility covariates then $\Omega_T(j) = 1$ is used.

In this package, we also consider two broad types of ILM models based on the type of the kernel function $\kappa(i, j)$: spatial and network-based ILMs. In the spatial-based ILMs, the infection kernel function is represented by the power-law function as

$$\kappa(ij) = d_{ij}^{-\beta},$$

where β is the spatial parameter that accounts for the varying risk of transmitting disease over the Euclidean distance between individuals i and j , d_{ij} . Whereas in the network-based ILMs, $\kappa(i, j)$ can be represented by one or more contact network matrices and is written as

$$\kappa(ij) = \beta_1 C_{ij}^{(1)} + \cdots + \beta_n C_{ij}^{(n)},$$

where $C_{ij}^{(.)}$ denotes the $(i, j)^{th}$ element of what we term the contact matrix of a given contact network; in graph theory this is more typically referred to as a (weighted) adjacency matrix. The corresponding $\beta_{(.)}$'s represent the effect of each of the n networks on transmission risk. In each contact network, each individual in the population is denoted by a node and is connected by lines or edges. These connections represent potential transmission routes through which disease can spread between individuals in the population. If the network is unweighted, the contact matrix is treated as binary (0 or 1). If the edges have weights assigned to them, then $C_{ij}^{(.)} \in R^+$ or $C_{ij}^{(.)} \in [0, 1]$ are typically used. These weights can be used to allow for different infection potential between different pairs of individuals. If the network is undirected, the contact matrix will be symmetric; if directed, it can be non-symmetric. Finally, the C_{ii} (diagonal terms) are not used in the models and are typically set to $C_{ii} = 0, \forall i$.

Note that $\mathbb{P}(i, t)$ gives the probability that susceptible individual i is infected at time point t , representing some interval in continuous time (e.g., a day or week), but they actually become infectious at time $t + 1$.

Following Deardon et al. (2015), the likelihood function for the ILMs (1) is given by

$$f(S, I, R|\theta) = \prod_{t=1}^{t_{max}} f_t(S, I, R|\theta) \quad (2)$$

where

$$f_t(S, I, R|\theta) = \left[\prod_{i \in I(t+1) \setminus I(t)} \mathbb{P}(i, t) \right] \left[\prod_{i \in S(t+1)} (1 - \mathbb{P}(i, t)) \right] \quad (3)$$

and where, θ is the vector of unknown parameters, $I(t+1) \setminus I(t)$ denotes all new infections observed at $t + 1$ in the infectious state at time t , and $t_{max} \leq t_{end}$ is the last time point at which data are observed or being simulated.

Contents of EpiILM

The **EpiILM** package makes use of Fortran code that is called from within R. This package can be used to carry out simulation of epidemics, calculate the basic reproduction number, plot various epidemic summary graphics, calculate the log-likelihood, and carry out Bayesian inference using Metropolis-Hastings MCMC for a given data set and model. The functions involved in the package are summarized in Table 1.

Simulation of epidemics

The function `epidata()` allows the user to simulate epidemics under different models and scenarios. One can use the argument `type` to select the compartmental framework (SI or SIR) and population size through the argument `n`. If the compartmental framework is SIR, the infectious period is passed through the argument `infperiod`. Depending on whether a spatial or network model is being considered, the user can pass the arguments: `x`, `y` for location and `contact` for contact networks. Users can also control the susceptibility function $\Omega_S(i)$ through the `Sformula` argument, with individual-level covariate information passable through this argument. If there is no covariate information, `Sformula` is null. An expression of the form `Sformula = ~ model` is used to specify the covariate information, separated by `+` and `-` operators similar to the R generic function `formula()`. For example, $\Omega_S(i) = \alpha_0 + \alpha_1 X(i)$, $i = 1, \dots, n$ can be passed through the argument `Sformula` as `Sformula = ~ 1 + X`. In a similar way, the user can control the transmissibility function $\Omega_T(i)$ through the

Function	Output
<code>epiB0</code>	Calculates the basic reproduction number for a specified SIR model
<code>epidata</code>	Simulates epidemic for the specified model type and parameters
<code>plot.epidata</code>	Produces spatial plots of epidemic progression over time as well as various epidemic curves of epidata object
<code>epidic</code>	Computes the deviance information criterion for a specified individual-level model
<code>epilike</code>	Calculates the log-likelihood for the specified model and data set
<code>epimcmc</code>	Runs an MCMC algorithm for the estimation of specified model parameters
<code>summary.mcmc</code>	Produces the summary of epimcmc object
<code>plot.mcmc</code>	Plots epimcmc object
<code>pred.epi</code>	Computes posterior predictions for a specified epidemic model
<code>plot.pred.epi</code>	Plot posterior predictions

Table 1: Description of functions and their output in the [EpiILM](#) package

`Tformula` argument. Note that, the `Tformula` must not include the intercept term to avoid model identifiability issues, i.e., for a model with one transmissibility covariate (`X`), the `Tformula` becomes `Tformula = ~ -1 + X`. The spatial (or network), susceptibility, transmissibility, and spark (if any) parameters are passed through arguments `beta`, `alpha`, `phi`, and `spark`, respectively.

The argument `tmin` helps to fix the initial infection time while generating an epidemic. By default, `tmin` is set as time $t = 1$. We can also specify the initial infective or infectives using the argument `inftime`. For example, in a population of 10 individuals, we could choose, say, the third individual to become infected at time point 1, using the option `inftime = c(0,0,1,0,0,0,0,0,0,0)`. We could also infect more than one individual and they could be infected at different time points. This allows simulation from a model conditional on, say, data already observed, if we set the `tmin` option at the maximum value of `inftime`.

The output of the function `epidata()` is formed as class of `epidata` object. This `epidata` object contains a list that consist of `type` (the compartmental framework), `XYcoordinates`(the XY coordinates of individual for spatial model) or `contact` (the contact network matrix for the network model), `inftime` (the infection times) and `remtime` (the removal times). Other functions such as `plot.epidata` and `epimcmc` involved in the package use this `object` class as an input argument.

Descriptive analyses

We introduce an S3 method `plot` function to graphically summarize, and allow for a descriptive analyses of, epidemic data. The function `plot.epidata()` illustrate the spread of the epidemic over time. One of the key input arguments (`x`) of this function has to be an `epidata` object. The other key argument `plottype` has two options: `curve` and `spatial`. Specifying the first option produces various epidemic curves, while the latter show the epidemic propagation over time and space when the model is set to spatial-based. When the `plottype = curve`, an additional argument needs to be passed through the function through `curvetype`. This has four options: `curvetype = "complete"` produces curves of the number of susceptible, infected, and removed individuals over time (when `type = "SIR"`); `"susceptible"` gives a single curve for the susceptible individuals over time; `"totalinfect"` gives the cumulative number of infected individuals over time; and `"newinfect"` produces a curve of the number of newly infected individuals at each time point.

The plot functions `plot.epimcmc()` and `plot.pred.epi()` can be used to illustrate inference results (see Bayesian inference section). Detailed explanation is provided in the corresponding subsections.

Example: spatial model

Suppose we want to simulate an epidemic from the model (1) using a spatial kernel with type SI, $\Omega_s(i) = \alpha$, and no transmissibility covariates, $\Omega_T(j) = 1$. Choosing the infectivity parameter $\alpha = 0.3$, spatial parameter $\beta = 5.0$, sparks parameter $\varepsilon = 0$, and $t_{max} = 15$, the model is given by:

$$\mathbb{P}(i, t) = 1 - \exp\{-0.3 \sum_{j \in I(t)} d_{ij}^{-5}\}, \quad t = 1, \dots, 15 \quad (4)$$

where d_{ij} is the Euclidean distance between individuals i and j , with their locations specified through x and y .

First, we install the [EpiILM](#) package and call the library.

```
R> install.packages("EpiILM")
R> library("EpiILM")
```

Then, let us simulate (x, y) coordinates uniformly across a 10×10 unit square.

```
R> x <- runif(100, 0, 10)
R> y <- runif(100, 0, 10)
```

One could now use the following syntax to simulate an epidemic from spatial model (4) and summarize the output as an epidemic curve and spatial plot.

```
R> SI.dis <- epidata(type = "SI", n = 100, tmax = 15, sus.par = 0.3, beta = 5.0,
+                      x = x, y = y)
R> SI.dis$inftime
[1] 0 0 10 4 7 0 0 0 4 5 0 0 9 2 6 6 3 0 0 0 11
[22] 0 0 9 12 3 0 15 9 0 8 0 0 0 11 9 0 2 0 5 7 0
[43] 2 15 7 0 0 5 0 0 0 14 8 2 0 15 9 10 10 4 1 5 4
[64] 6 11 3 0 8 0 6 8 5 12 4 2 13 0 9 3 6 3 4 9 13
[85] 0 0 0 0 0 0 0 0 10 0 9 11 9 9 0 0
```

Here, the epidemic is generated across the uniformly distributed population of 100 individuals with a default first infection time $t = 1$ and last observed time point of $t_{max} = 15$. The declaration of the spatial locations of individuals through x and y specifies that we are simulating from a spatial model. (See later for network-based models). The output `SI.dis$inftime` provides the times at which individuals enter the infectious state, with 0 representing individuals who are still susceptible at time t_{max} . Figures 1 and 2 show the summary graphics for the simulated epidemic, which are produced using the S3 method `plot.epidata()` as follows:

```
R> plot(SI.dis, plottype = "curve", curvetype = "complete")
R> plot(SI.dis, plottype = "spatial")
```

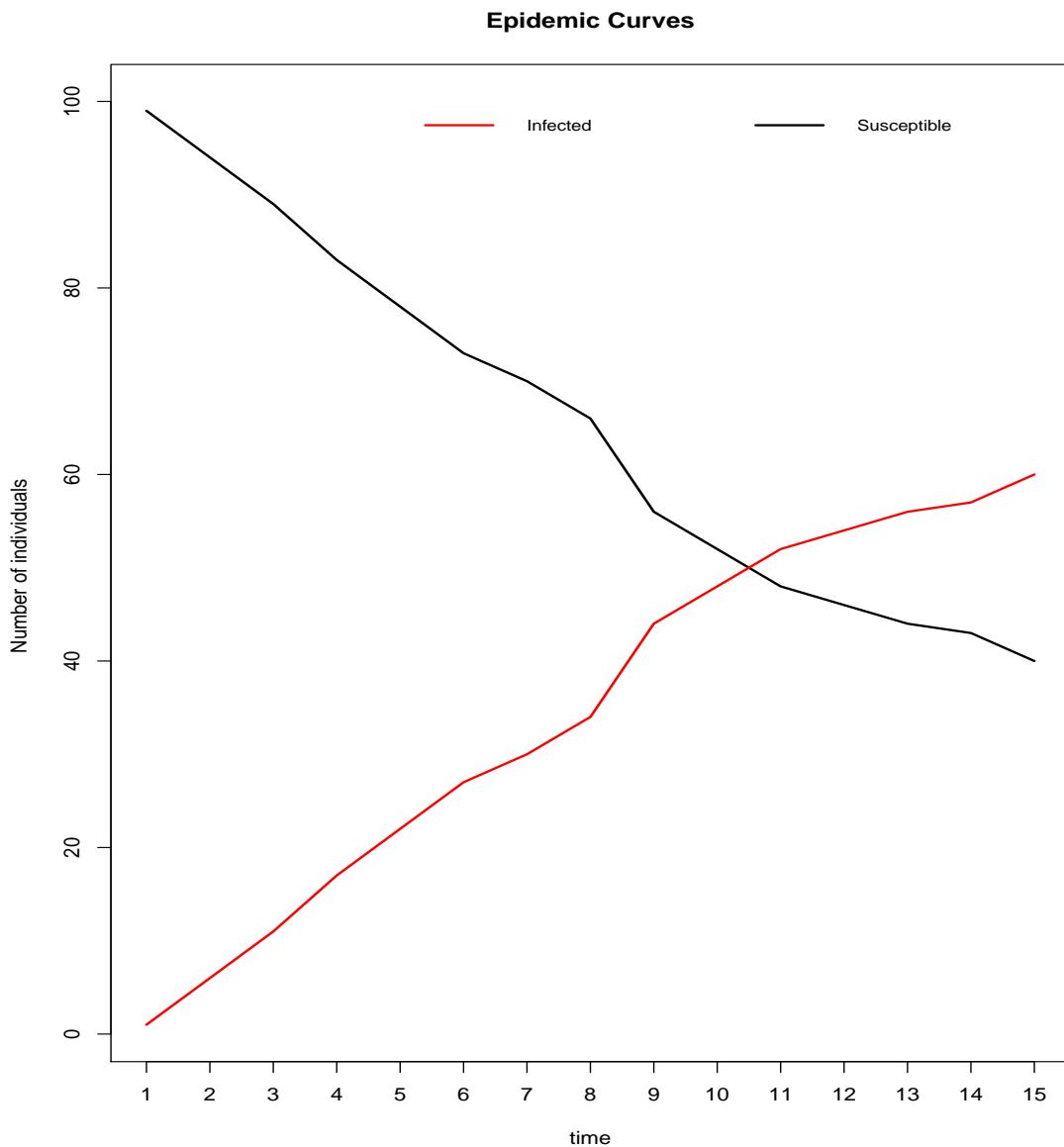


Figure 1: Epidemic curves of simulated epidemic from (4) for 100 individuals at $t = 1, \dots, 15$

Example: network model

To illustrate simulation from a contact network-based model, we consider a disease system in which disease transmission can occur through a single, directed, binary network over a population of $n = 100$ individuals. The elements in the contact matrix represent the existence or non-existence of a directed connection through which disease can transmit between two individuals in the population. Each individual within the population is represented by a row and column within the matrix. Specifically, an element C_{ij} in the contact matrix is given by:

$$C_{ij} = \begin{cases} 1 & \text{if a directed edge exists between } i \text{ and } j \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

We also consider the inclusion of a binary susceptibility covariate Z in the model. This can be thought to represent, say, treatment or vaccination status. The infection model is then given by

$$\mathbb{P}(i, t) = 1 - \exp\{-(\alpha_0 + \alpha_1 Z_i) \sum_{j \in I(t)} C_{ij}\}, \quad t = 1, \dots, t_{max} \quad (6)$$

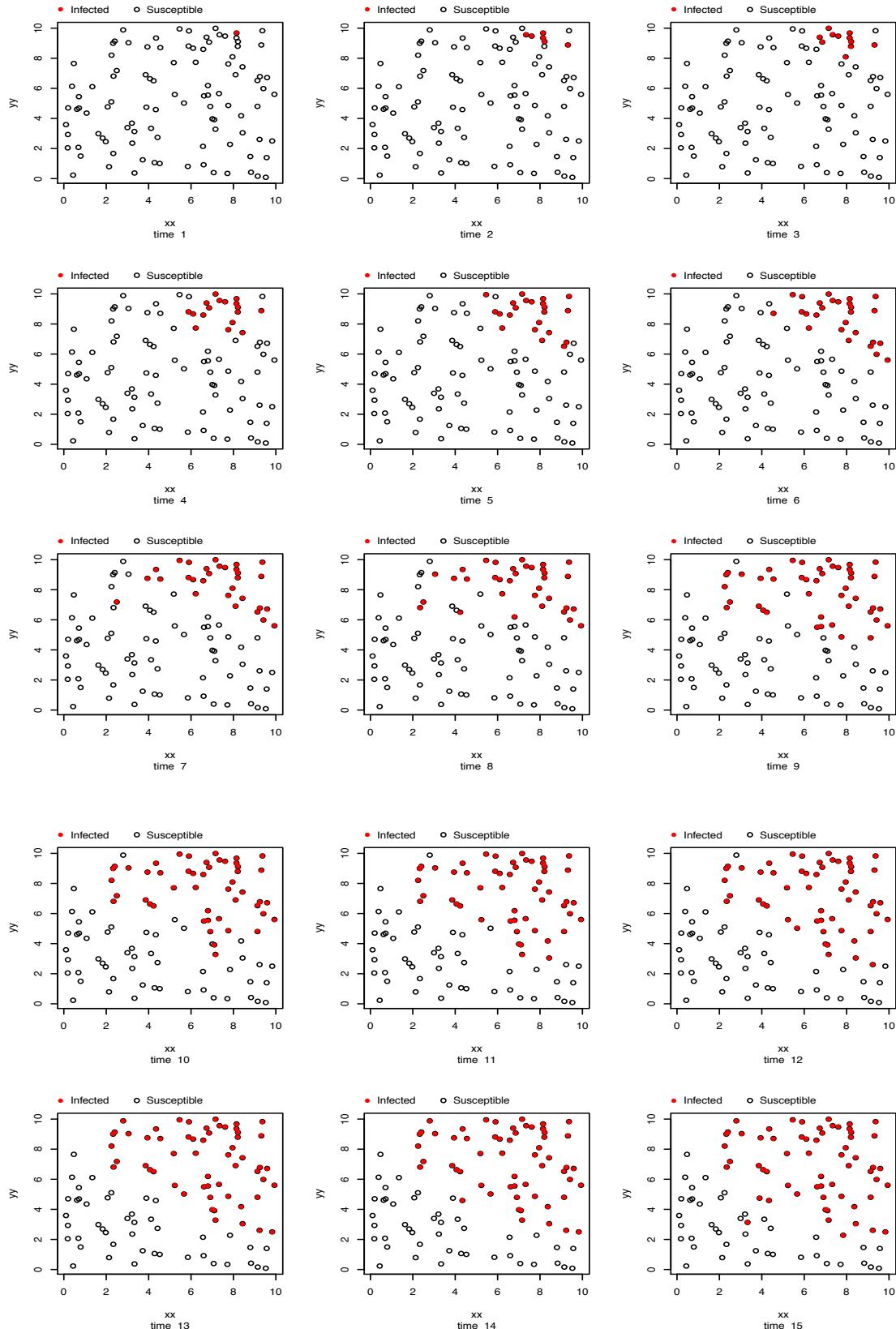


Figure 2: Simulated epidemic from (4) for 100 individuals, where open circles represent susceptible individuals and filled red circles represent infected individuals

where α_0 is the baseline susceptibility and α_1 is the binary treatment effect. The parameters in the model are set to be $(\alpha_0, \alpha_1) = (0.1, 0.05)$ and $\varepsilon = 0$, and we also set $t_{max} = 15$. We simulate a directed network using the following code:

```
R> contact <- matrix(rbinom(10000, 1, 0.1), nrow = 100, ncol = 100)
R> diag(contact[, ]) <- 0
```

Various packages are available in R for network visualization, such as `igraph`, `ergm`, etc and as an example, we use the `igraph` package for the network display as shown in Figure 3.

```
R> require("igraph")
R> net1 <- graph_from_adjacency_matrix(contact)
R> plot(net1, vertex.size = 10, vertex.label.cex = 0.5, edge.arrow.mode = "-")
```

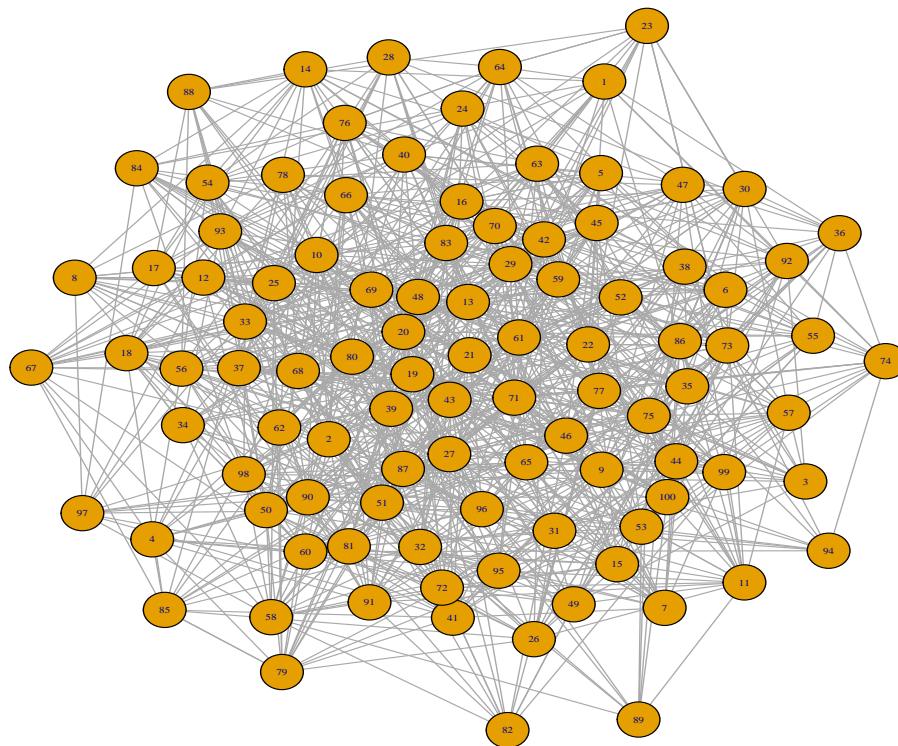


Figure 3: Contact network generated for model (6)

The epidemic is generated using the function `epidata()`. The arguments `Sformula = ~ Z` and `sus.par = c(0.1, 0.05)` define the susceptibility function $0.1 + 0.05Z_i$. As there is one contact network matrix in the model, the effect of the network C is set to one by default. The use of the `contact` argument informs the package that we are dealing with a contact network-based model.

```
R> Z <- round(runif(100, 0, 2))
R> SI.contact <- epidata(type = "SI", Sformula = ~Z, n = 100, tmax = 15,
+           sus.par = c(0.1, 0.05), contact = contact)
R> SI.contact$inftime
[1] 0 9 11 10 9 5 8 9 8 9 9 10 1 7 12 11 7 11 8 4 8
```

```
[22] 10 8 11 7 9 11 5 7 6 6 8 9 8 9 6 6 6 7 9 8 9 7 9 9 7 4
[43] 7 10 8 3 10 9 10 11 6 7 9 6 5 11 4 7 8 8 10 8 8
[64] 7 7 6 11 7 8 7 6 8 8 10 7 6 10 11 9 6 10 7 4 7
[85] 8 7 9 9 10 9 8 10 5 8 7 7 8 9 9 9 8
```

Figure 4 shows the epidemic curve, obtained via the following code.

```
R> plot(SI.contact, plottype = "curve", curvetype = "complete")
```

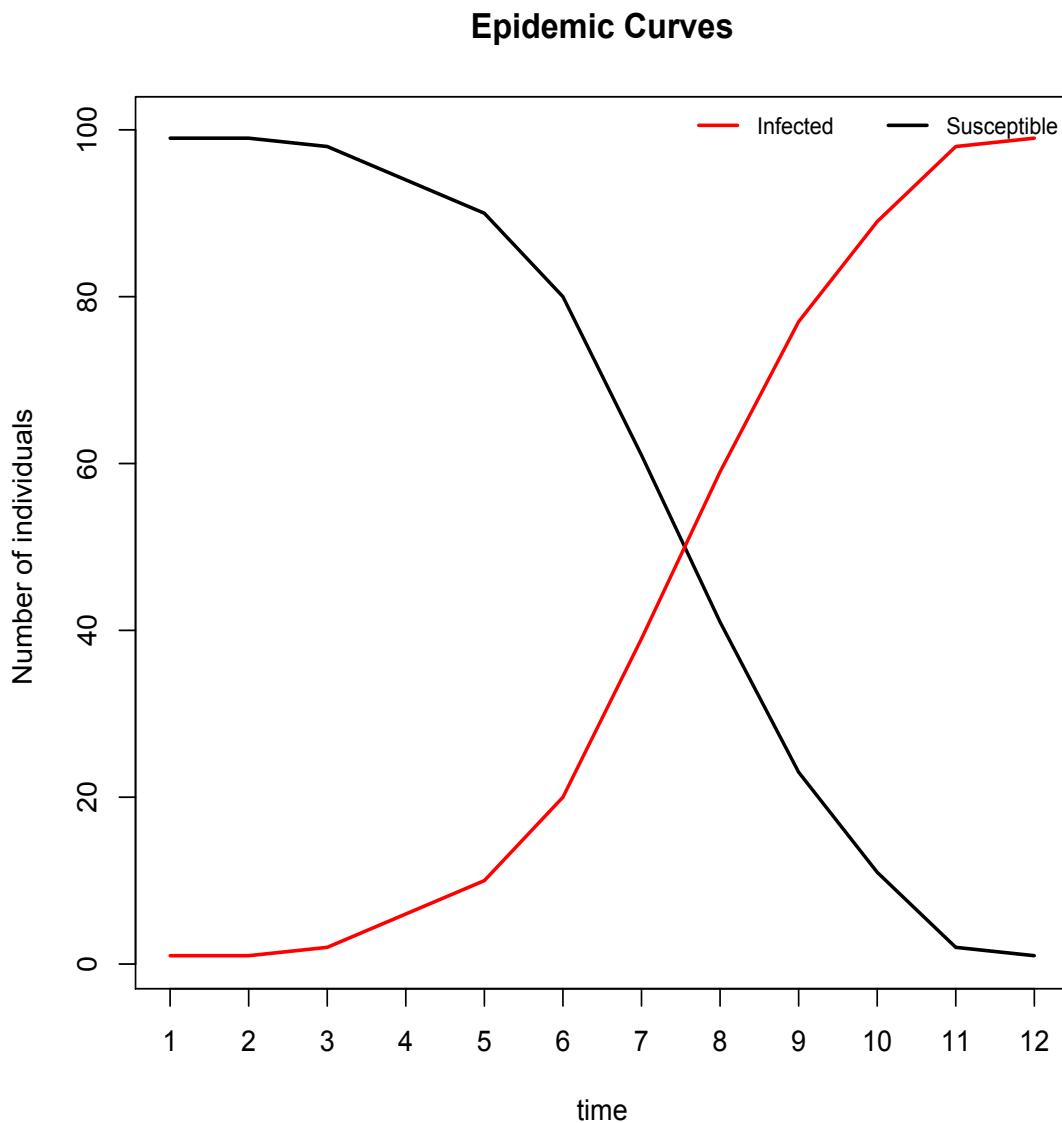


Figure 4: Epidemic curve of contact network model (6) for 100 individuals

Bayesian Inference

In **EpiILM**, ILMs can be fitted to observed data within a Bayesian framework. A Metropolis-Hastings MCMC algorithm is provided which can be used to estimate the posterior distribution of the parameters. The function `epimcmc()` provides three choices for the marginal prior distribution of each parameter: the gamma, half-normal, and uniform distributions. The parameters are assumed to be *a priori* independent. The proposal used is a Gaussian random walk. Again, users can use the `Sformula` and `Tformula` arguments to specify any individual-level susceptibility and transmissibility covariates. Users can also control the number of MCMC simulations, initial values, and proposal variances of the parameters to be estimated. Note that in case of fixing one

parameter and updating other parameters, users can do it by setting the proposal variance of fixed parameter to zero. This is usually the case to avoid identifiability issue when the model has both susceptibility and transmissibility covariates without intercept terms. Again, spatial/network, susceptibility parameters and transmissibility parameters are passed through arguments `beta`, `sus.par` and `trans.par`, respectively. One can specify the spark parameter using the `spark` argument, but by default its value is 0. `epimcmc()` can also call the adaptive MCMC method of inference facilitated by the `adaptMCMC` package. Specifically, users can pass the argument `adapt = TRUE` along with `acc.rate` to run the adapt MCMC algorithm.

We can use the S3 method functions `summary.epimcmc()` and `plot.epimcmc()` available in the package for output analysis and diagnostics. Both are dependent upon the `coda` package. The argument `plottype` in the function `plot.epimcmc`, has two options to specify which samples are to be plotted: (1) "parameter" is used to produce trace plots (time series plots) of the posterior distributions of the model parameters, and (2) "loglik" to produce trace plots of the log likelihood values of the model parameter samples. Other options that are used in the `coda` package can be used in the `plot.epimcmc()` as well; e.g., `start`, `end`, `thin`, `density`, etc.

Spatial or network-based ILMs can be fitted to data as shown in the following examples.

Example: spatial model

Suppose we are interested in modelling the spread of a highly transmissible disease through a series of farms, say $n = 100$, along with an assumption that the spatial locations of the farms and the number of animals on each farm are known. In this situation, it is reasonable to treat the farms themselves as individual units. If we treat the extent of infection from outside the observed population of farms as negligible ($\varepsilon = 0$), we can write the ILM model as

$$\text{IP}(i, t) = 1 - \exp\{-(\alpha_0 + \alpha_1 A(i)) \sum_{j \in I(t)} d_{ij}^{-\beta}\}, \quad t = 1, \dots, t_{max}, \quad (7)$$

where the susceptibility covariate A represents the number of animals on each farm, α_0 is the baseline susceptibility, α_1 is the number of animals effect, and β is the spatial parameter. Let us use the same (simulated) spatial locations from the previous spatial model example and set the parameters $(\alpha_0, \alpha_1) = (0.2, 0.1)$ and $\beta = 5$. We also set $t_{max} = 50$. Considering an SI compartmental framework for this situation, the epidemic is simulated using the following command:

```
R> A <- round(rexp(100, 1/50))
R> SI.dis.cov <- epidata(type = "SI", n = 100, tmax = 50, x = x, y = y,
+                           Sformula = ~A, sus.par = c(0.2, 0.1), beta = 5)
```

We can now refit the generating model to this simulated data and consider the posterior estimates of the model parameters. We can do this using the following code:

```
R> t_end <- max(SI.dis.cov$infetime)
R> unif_range <- matrix(c(0, 0, 10000, 10000), nrow = 2, ncol = 2)
R> mcmcout_Model7 <- epimcmc(SI.dis.cov, Sformula = ~A, tmax = t_end, niter = 50000,
+                               sus.par.ini = c(0.001, 0.001), beta.ini = 0.01,
+                               pro.sus.var = c(0.01, 0.01), pro.beta.var = 0.5,
+                               prior.sus.dist = c("uniform", "uniform"), prior.sus.par = unif_range,
+                               prior.beta.dist = "uniform", prior.beta.par = c(0, 10000))
```

where `niter` denotes the number of MCMC iterations and `sus.par.ini` and `beta.ini` are the initial values of the parameters to be estimated. The proposal variances for (α_0, α_1) and β are set to $(0.01, 0.01)$ and 0.5 , respectively (after tuning). Vague uniform prior distributions are used for all three parameters, i.e., we choose $U(0, 10000)$ for α_0, α_1 , and β . As the locations x and y are specified, a spatial ILM is fitted rather than a network-based ILM. Note that the full data set to which the model is being fitted consists of the spatial locations (x, y) and the infection times. Figure 5 displays the MCMC traceplot after 10000 burn-in using the command

```
R> plot(mcmcout_Model7, partype = "parameter", start = 10001, density = FALSE)
```

The posterior means and 95% credible intervals (CI) of the parameters, calculated as the 2.5% and 97.5% percentiles of 50000 MCMC draws after a burn-in of 10000 iterations has been removed, can be obtained using the following code:

```
R> summary(mcmcout_Model7, start = 10001)
```

Model: SI distance-based discrete-time ILM
 Method: Markov chain Monte Carlo (MCMC)

```
Iterations = 10001:50000
Thinning interval = 1
Number of chains = 1
Sample size per chain = 40000
```

1. Empirical mean and standard deviation for each variable,
 plus standard error of the mean:

	Mean	SD	Naive SE	Time-series SE
alpha.1	0.2868	0.23516	0.0011758	0.012449
alpha.2	0.1997	0.07544	0.0003772	0.002393
beta.1	5.6790	0.48188	0.0024094	0.014631

2. Quantiles for each variable:

	2.5%	25%	50%	75%	97.5%
alpha.1	0.02018	0.1141	0.2226	0.3917	0.9072
alpha.2	0.08680	0.1486	0.1875	0.2395	0.3769
beta.1	4.81361	5.3377	5.6704	5.9872	6.6588

Example: network model

Now consider modelling the spread of an animal infectious disease through a series of farms ($n = 500$) in a region. We once again consider individuals in the population to be the farms, rather than animals themselves, and a single binary contact network to represent connections between farms. We assume that there are two species of animals in farms that play an important role in spreading the disease; let us say, cows and sheep. Thus, we include the number of cows and sheep on the farm as susceptibility and transmissibility covariates in the model. Thus, the probability of susceptible farm i to be infected at time t becomes:

$$\mathbb{P}(i, t) = 1 - \exp\{-(\alpha_1 X_1(i) + \alpha_2 X_2(i)) \sum_{j \in I(t)} [\phi_1 X_1(j) + \phi_2 X_2(j)] C_{ij}\}, \quad t = 1, \dots, t_{max} \quad (8)$$

with susceptibility parameters, (α_1, α_2) , transmissibility parameters, (ϕ_1, ϕ_2) , and X_1 and X_2 represent the number of sheep and cows in farms, respectively. Note, when we have a single network, there is no network parameter to estimate; this is to avoid problems of non-identifiability.

We can generate such a directed contact network using the following commands:

```
R> n <- 500
R> contact <- matrix(rbinom(n*n, size = 1, prob = 0.1), nrow = n, ncol = n)
R> diag(contact) <- 0
```

We also sample the number of sheep and cows on each farm using the following code

```
R> X1 <- round(rexp(n, 1/100))
R> X2 <- round(rgamma(n, 50, 0.5))
```

Assuming each infected farm to be infectious for an infectious period of 3 days and by setting $t_{max} = 25$, $\alpha_1 = 0.003$, $\alpha_2 = 0.01$, $\phi_1 = 0.0003$, and $\phi_2 = 0.0002$, we can simulate the epidemic from the SIR network-based ILMs (8) as follows and the epidemic curves are shown in Figure 6.

```
R> infp <- rep(3, n)
R> SIR.net <- epidata(type = "SIR", n = 500, tmax = 15,
+                     sus.par = c(0.003, 0.01), trans.par = c(0.0003, 0.0002),
+                     contact = contact, infperiod = infp,
+                     Sformula = ~ -1 + X1 + X2, Tformula = ~ -1 + X1 + X2)
R> plot(SIR.net, plottype = "curve", curvetype = "complete")
```

To estimate the unknown parameters $(\alpha_1, \alpha_2, \phi_1, \phi_2)$, we use the function `epimcmc()` assuming the event times (infection and removal times) and contact network are observed. Note that the specification of the contact argument means a network-based (rather than spatial) ILM will be assumed. We run the `epimcmc()` function to produce an MCMC chain of 50000 iterations assigning

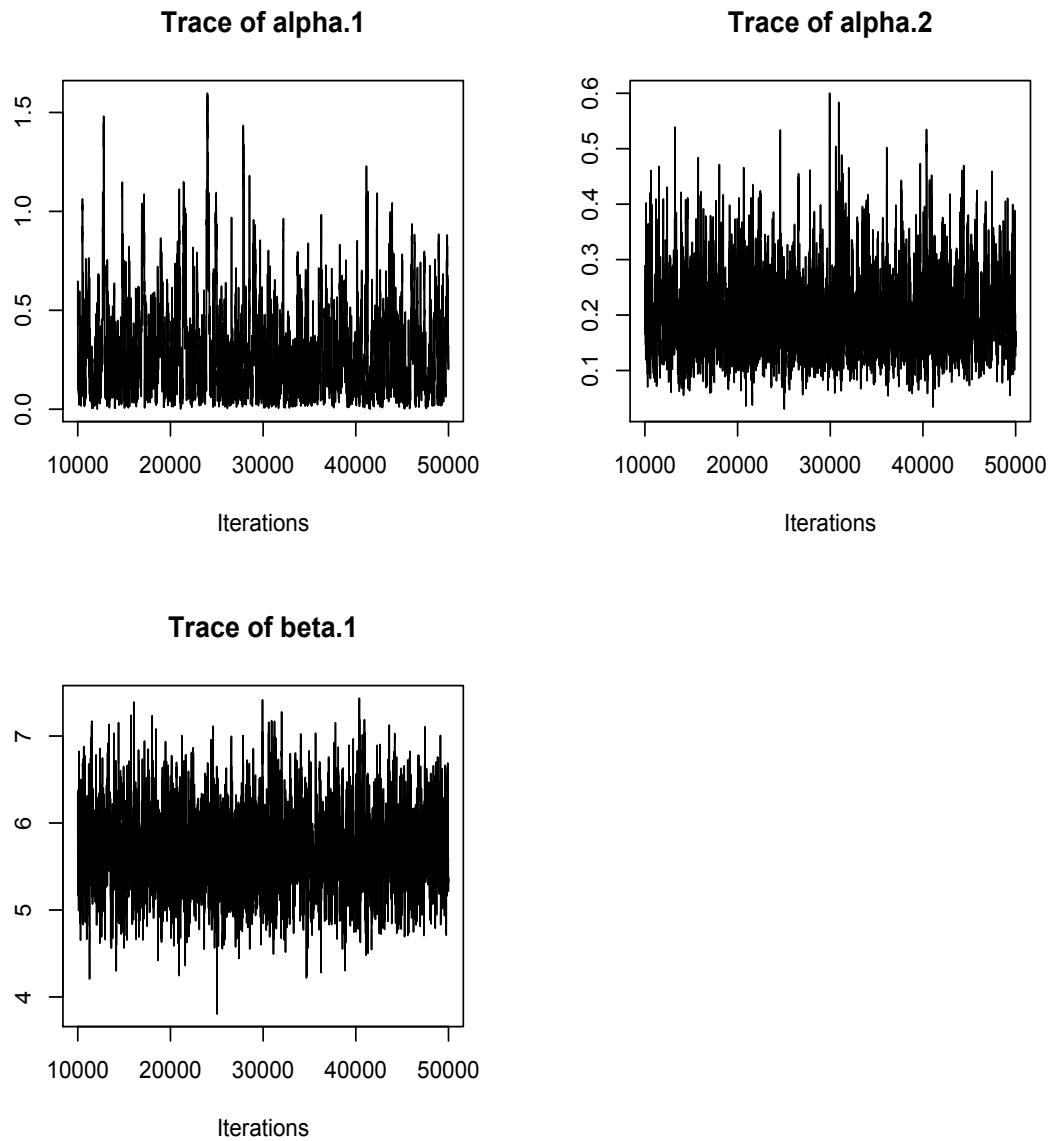


Figure 5: MCMC traceplot for the estimation of model (7) parameters

gamma prior distribution for the model parameters (α_2, ϕ_1, ϕ_2) while fixing α_1 to avoid the non-identifiability issue in the model. This is done by setting the proposal variance of this parameter to zero in the `pro.sus.var` argument. To illustrate the use of adaptMCMC method, we set `adapt = TRUE` and set the acceptance rate as 0.5 via `acc.rate = 0.5`. It should also be noted that we need to provide initial values and proposal variances of the parameters when we use the adaptive MCMC option in the `epimcmc()` function. The syntax is as follows:

```
R> t_end <- max(SIR.net$inftime)
R> prior_par <- matrix(rep(1, 4), ncol = 2, nrow = 2)
R> mcmcout_SIR.net <- epimcmc(SIR.net, tmax = t_end, niter = 50000,
+   Sformula = ~-1 + X1 + X2, Tformula = ~-1 + X1 + X2,
+   sus.par.ini = c(0.003, 0.001), trans.par.ini = c(0.01, 0.01),
+   pro.sus.var = c(0.0, 0.1), pro.trans.var = c(0.05, 0.05),
+   prior.sus.dist = c("gamma", "gamma"), prior.trans.dist = c("gamma", "gamma"),
+   prior.sus.par = prior_par, prior.trans.par = prior_par,
+   adapt = TRUE, acc.rate = 0.5)
```

Figure 7 shows the MCMC traceplot for the 50000 iterations. The estimate of the posterior mean of the model parameters (α_2, ϕ_1, ϕ_2) and their 95% credible intervals, after 10000 iterations of burn-in have been removed are: $\hat{\alpha}_2 = 0.0094$ (0.0051, 0.0168), $\hat{\phi}_1 = 0.0004$ (0.0002, 0.0007), and

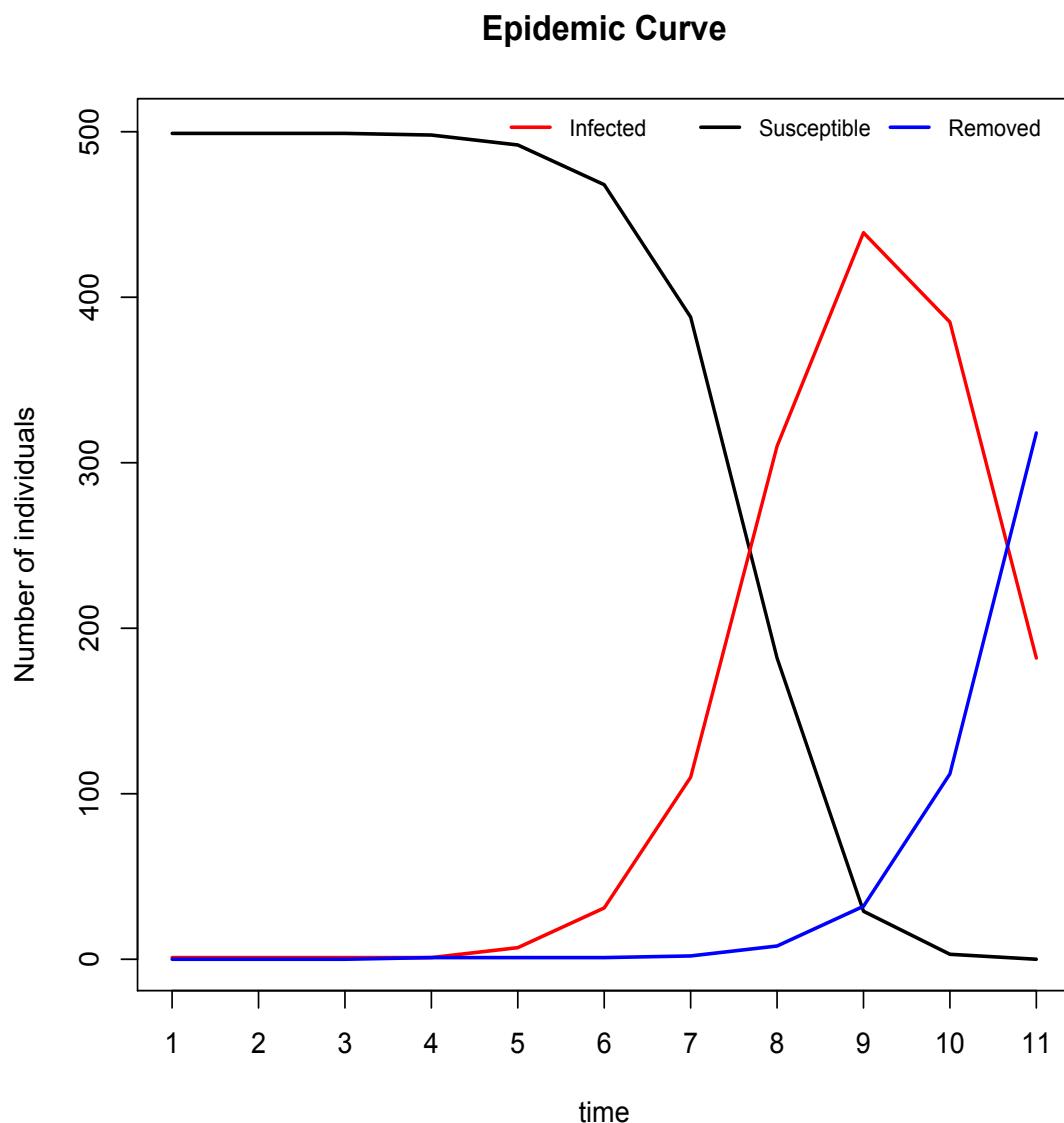


Figure 6: Epidemic curve of the contact network model (8) for 500 farms

$\hat{\phi}_1 = 0.0002$ (0.00006, 0.0003).

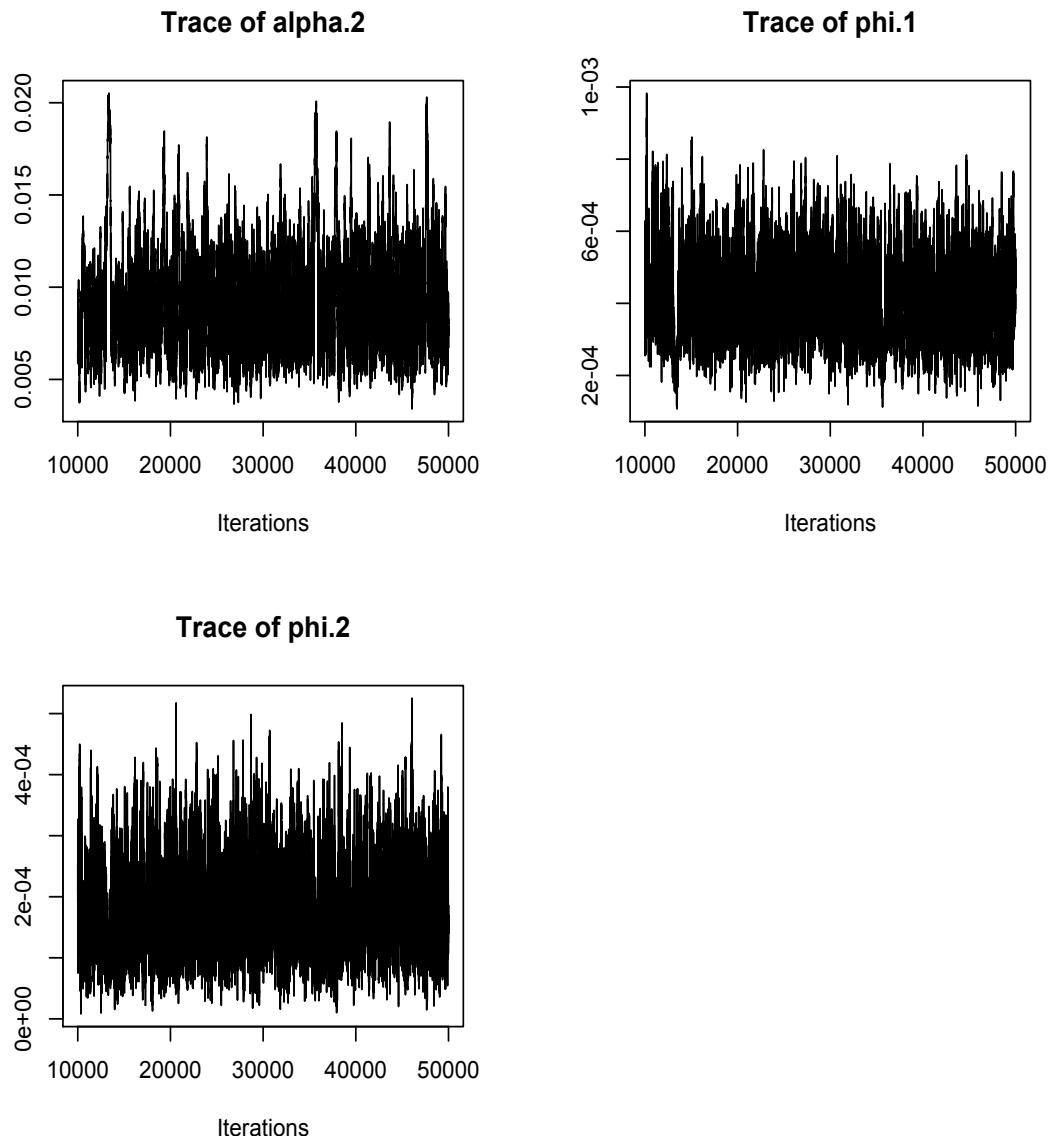


Figure 7: MCMC traceplot for the estimation of model (8) parameters.

```
R> summary(mcmc.out_SIR.net, start = 10001)

Model: SIR network-based discrete-time ILM
Method: Markov chain Monte Carlo (MCMC)

Iterations = 10001:50000
Thinning interval = 1
Number of chains = 1
Sample size per chain = 40000

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

      Mean        SD  Naive SE Time-series SE
alpha.2 0.0093899 0.0030070 1.504e-05    1.866e-04
phi.1   0.0004123 0.0001218 6.089e-07    4.483e-06
phi.2   0.0001851 0.0000732 3.660e-07    1.911e-06

2. Quantiles for each variable:
```

	2.5%	25%	50%	75%	97.5%
alpha.2	5.130e-03	0.0071826	0.0088076	0.0110550	0.0168404
phi.1	2.123e-04	0.0003226	0.0004020	0.0004879	0.0006800
phi.2	6.646e-05	0.0001322	0.0001763	0.0002299	0.0003463

Case Study: Tomato spotted wilt virus (TSWV) data

Here, we consider data from a field trial on TSWV as described in [Hughes et al. \(1997\)](#). The experiment was conducted on 520 pepper plants grown inside a greenhouse and the spread of the disease caused by TSWV was recorded at regular time intervals. Plants were placed in 26 rows spaced one metre apart, with 20 plants spaced half a metre apart in each row. The experiment started on May 26, 1993 and lasted until August 16, 1993. During this period, assessments were made every 14 days. We set the initial infection time to be $t = 2$ as the epidemic starts at time point 2 and the last observation made is set to be $t = 7$. The TSWV data in this package contain ID number, locations (x and y) and infectious and removal time of each individual. The data set was reconstructed from a spatial plot in [Hughes et al. \(1997\)](#). The infectious period for each tomato plant is assumed to span three time points ([Brown et al., 2005](#), [Pokharel and Deardon, 2016](#)). Thus, an SIR model is fitted using the following code:

```
R> data(tswv)
R> x <- tswv$x
R> y <- tswv$y
R> infime <- tswv$infime
R> removaltime <- tswv$removaltime
R> infperiod <- rep(3, length(x))
```

In order to see the spatial dispersion of the TSWV data, we use the function `epispatial()`. As no new infections occurred at the second time point, we display spatial plots from that second observation by setting `tmin = 2` (see Figure 8). The required code is given by:

```
R> epidat.tswv <- as.epidata(type = "SIR", n = 520, x = x, y = y,
+                               infime = infime, infperiod = infperiod)
R> plot(epidat.tswv, plottype = "spatial", tmin = 2)
```

The function `epimcmc()` is used to fit our model to the data. We ran 50000 MCMC iterations. The computing time taken for the MCMC is about 10 min on a 16 GB MacBook Pro with a 2.9 GHz Intel Core i5 processor. Proposal variances are chosen to be 0.000005 and 0.005 for α and β , respectively (after tuning). We use vague independent marginal prior distributions for the parameters. Here, a gamma distribution with shape parameter 1 and rate parameter 10^{-3} is used for both α and β . We choose to fit our model to data starting at the second time point because no infections occur between the first and second time points. Thus, we again set `tmin = 2`. The code for fitting our model is:

```
R> mcmc.tswv <- epimcmc(epidat.tswv, tmin = 2, tmax = 10,
+                           niter = 50000, sus.par.ini = 0.0, beta.ini = 0.01,
+                           pro.sus.var = 0.000005, pro.beta.var = 0.005, prior.sus.dist = "gamma",
+                           prior.sus.par = c(1,10**(-3)), prior.beta.dist = "gamma",
+                           prior.beta.par = c(1,10**(-3)))
```

The posterior mean estimates for this model are $\hat{\alpha} = 0.014$ and $\hat{\beta} = 1.351$. The 95% credible interval of the posterior mean of α and β are (0.009, 0.0190) and (1.041, 1.629), respectively. Once again, these intervals are calculated as the 2.5% and 97.5% percentiles of 50000 draws after 10000 iterations of burn-in have been removed. The MCMC traceplots shown in Figure 9.

Conclusion

This paper discusses the implementation of the R software package [EpiILM](#). Other than this package, there does not appear to be any R software that offers spatial and network-based individual-level modelling for infectious disease systems. Thus, this package will be helpful to many researchers and students in epidemiology as well as in statistics. These models can be used to model disease systems of humans (e.g., [Malik et al. \(2014\)](#)), animals (e.g., [Kwong et al. \(2013\)](#)), or plants (e.g., [Pokharel and Deardon \(2016\)](#)), as well as other transmission-based systems such as invasive species (e.g., [Cook et al. \(2007\)](#)) or fire spread ([Vrbik et al., 2012](#)).

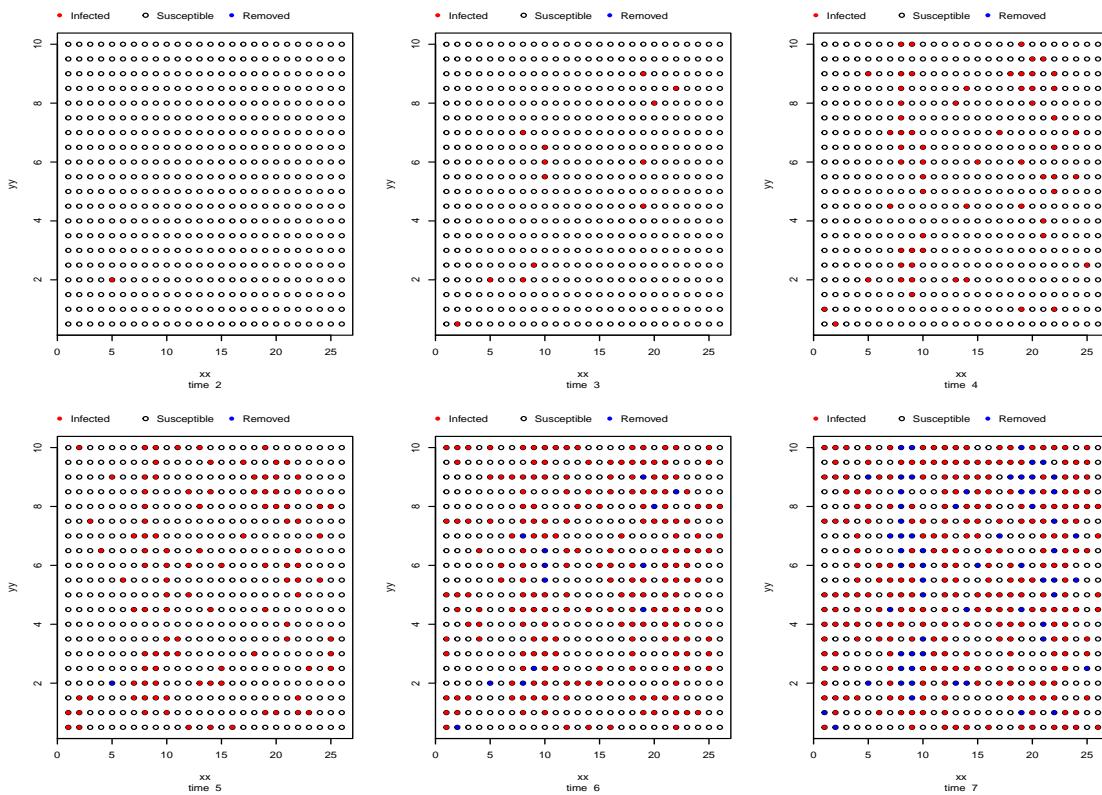


Figure 8: Epidemic dispersion of TSWV data across a grid of 520 individuals, where open circles represent susceptible individuals, filled red circles represent infected individuals, and blue crosses represent the removed individuals.

The [EpiILM](#) package continues to exist as a work in progress. We hope to implement additional models and options in the future, that might be useful for other researchers in their research and teaching. Such additions may include the incorporation of time-varying networks, covariates and/or spatial kernels (e.g., [Vrbik et al. \(2012\)](#)), models that allow for a joint spatial and network-based infection kernel, uncertainty in the times of transitions between disease states (e.g., [Malik et al. \(2016\)](#)), unknown covariates (e.g., [Deeth and Deardon \(2013\)](#)), and extensions to other compartmental frameworks such as SEIR and SIRS. Finally, we hope to extend the package to allow for the modelling of disease systems with multiple interacting strains or pathogens ([Romanescu and Deardon, 2016](#)).

Acknowledgments

Warriyar was funded by a University of Calgary Eyes High Postdoctoral Scholarship. Both Deardon, and equipment used to carry out this work, were funded by a Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery Grant.

Bibliography

- P.-Y. Boelle and T. Obadia. *R0: Estimation of R0 and Real-Time Reproduction Number from Epidemics*, 2015. URL <https://CRAN.R-project.org/package=R0>. R package version 1.2-6. [p]
- S. Brown, A. Csinos, J. Díaz-Pérez, R. Gitaitis, S. LaHue, J. Lewis, N. Martinez, R. McPherson, S. Mullis, C. Nischwitz, et al. Tospoviruses in solanaceae and other crops in the coastal plain of georgia. *The University of Georgia College of Agriculture and Environmental Sciences, Research Report*, 704:19, 2005. [p]

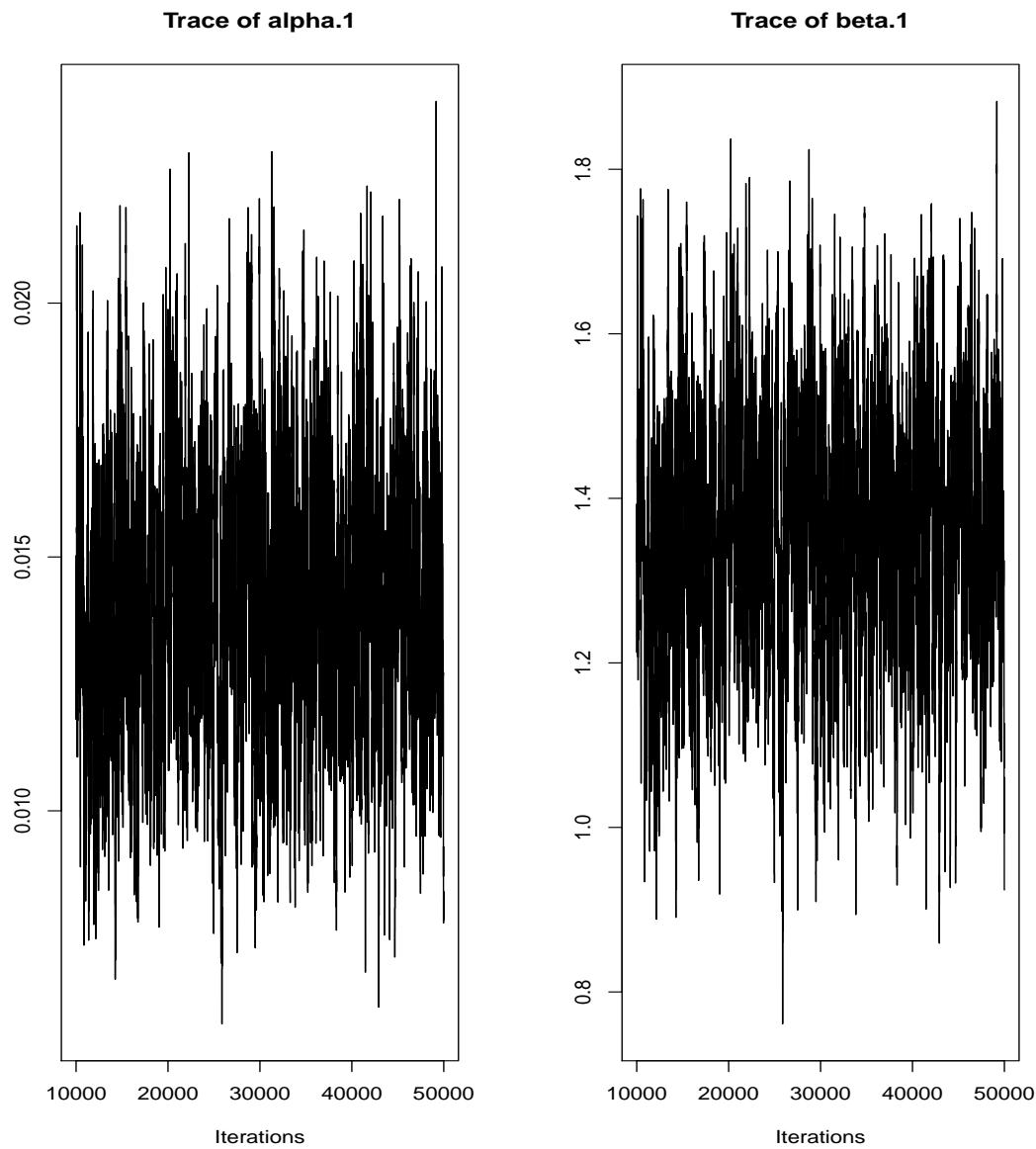


Figure 9: MCMC traceplots of the posterior samples for TSWV data

- A. R. Cook, G. Marion, A. Butler, and G. J. Gibson. Bayesian inference for the spatio-temporal invasion of alien species. *Bulletin of Mathematical Biology*, 69:2005–2025, 2007. [p]
- A. Cori. *EpiEstim: Estimate Time Varying Reproduction Numbers from Epidemic Curves*, 2019. URL <https://CRAN.R-project.org/package=EpiEstim>. R package version 2.2-1. [p]
- D. R. Cox. Regression models and life-tables. *Journal of the Royal Statistical Society*, 34(2):187–220, 1972. [p]
- R. Deardon, S. P. Brooks, B. T. Grenfell, M. J. Keeling, M. J. Tildesley, N. J. Savill, D. J. Shaw, and M. E. Woolhouse. Inference for individual-level models of infectious diseases in large populations. *Statistica Sinica*, 20(1):239, 2010. [p]
- R. Deardon, X. Fang, and G. Kwong. Statistical modeling of spatiotemporal infectious disease transmission. *Analyzing and Modeling Spatial and Temporal Dynamics of Infectious Diseases* (eds. Chen D, Moulin B, Wu J), pages 211–232, 2015. [p]
- L. Deeth and R. Deardon. Latent conditional individual-level models for infectious disease modeling. *The International Journal of Biostatistics*, 9(1):75–93, 2013. [p]
- C. Groendyke and D. Welch. *epinet: Epidemic/Network-Related Tools*, 2016. URL <https://CRAN.R-project.org/package=epinet>. R package version 2.1.7. [p]

- G. Hughes, N. McRoberts, L. V. Madden, and S. C. Nelson. Validating mathematical models of plant-disease progress in space and time. *Mathematical Medicine and Biology*, 14(2):85–112, 1997. [p]
- 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/>. [p]
- S. Jenness, S. M. Goodreau, and M. Morris. *EpiModel: Mathematical Modeling of Infectious Disease Dynamics*, 2018. URL <https://CRAN.R-project.org/package=EpiModel>. R package version 1.6.1. [p]
- C. P. Jewell, T. Kypraios, P. Neal, and G. O. Roberts. Bayesian analysis for emerging infectious diseases. *Bayesian Analysis*, 4(4):465–496, 2009. [p]
- G. Kwong, Z. Poljak, R. Deardon, and C. Dewey. Bayesian analysis of risk factors for infection with a genotype of porcine reproductive and respiratory syndrome virus in ontario swine herds using monitoring data. *Preventive Veterinary Medicine*, 110(3-4):405–17, 2013. [p]
- R. Malik, R. Deardon, G. Kwong, and B. J. Cowling. Individual-level modeling of the spread of influenza within households. *Journal of Applied Statistics*, 41(7):1578–1592, 2014. [p]
- R. Malik, R. Deardon, and G. Kwong. Parameterizing spatial models of infectious disease transmission that incorporate infection time uncertainty using sampling- based likelihood approximations. *PLoS One*, 11(1), 2016. URL <https://doi.org/10.1371/journal.pone.0146253>. [p]
- S. Meyer, L. Held, and M. Hohle. Spatio-temporal analysis of epidemic phenomena using the R package surveillance. *Journal of Statistical Software*, 77(11):1–55, 2017. URL <https://doi.org/10.18637/jss.v077.i11>. [p]
- K. M. O'Reilly, R. Lowe, W. J. Edmunds, P. Mayaud, A. Kucharski, R. M. Eggo, S. Funk, D. Bhatia, K. Khan, M. U. G. Kraemer, A. Wilder-Smith, L. C. Rodrigues, P. Brasil, E. Massad, T. Jaenisch, S. Cauchemez, O. J. Brady, and L. Yakob. Projecting the end of the zika virus epidemic in latin america: a modelling analysis. *BMC Medicine*, 16(1):180, 2018. [p]
- G. Pokharel and R. Deardon. Gaussian process emulators for spatial individual-level models of infectious disease. *Canadian Journal of Statistics*, 44(4):480–501, 2016. [p]
- R. Romanescu and R. Deardon. Modelling two strains of disease via aggregate-level infectivity curves. *Journal of Mathematical Biology*, 72(5):1195–1224, 2016. [p]
- T. M. Therneau. *A Package for Survival Analysis in S*, 2015. URL <https://CRAN.R-project.org/package=survival>. version 2.38. [p]
- M. J. Tildesley, N. J. Savill, D. J. Shaw, R. Deardon, S. P. Brooks, M. E. J. Woolhouse, B. T. Grenfell, and M. J. Keeling. Optimal reactive vaccination strategies for an outbreak of foot-and-mouth disease in great britain. *Nature*, 440(7080):83–86, 2006. [p]
- I. Vrbik, R. Deardon, Z. Feng, A. Gardner, and J. Braun. Using individual-level models to model the spatio-temporal dynamics of combustion. *Bayesian Analysis*, 7(3):615–638, 2012. [p]
- V. Warriyar. K. V., W. Almutiry, and R. Deardon. *EpiILM: Spatial and Network Based Individual Level Models for Epidemics*, 2020. URL <https://CRAN.R-project.org/package=EpiILM>. R package version 1.5. [p]

Vineetha Warriyar. K. V.
Faculty of Veterinary Medicine
University of Calgary
Canada
vineethawarriyar.kod@ucalgary.ca

Waleed Almutiry
Department of Mathematics
College of Science and Arts in Ar Rass
Qassim University
Saudi Arabia
wkmtierie@qu.edu.sa

Rob Deardon

*Faculty of Veterinary Medicine and Department of Mathematics and Statistics
University of Calgary
Canada
robert.deardon@ucalgary.ca*

SurvBoost: An R Package for High-Dimensional Variable Selection in the Stratified Proportional Hazards Model via Gradient Boosting

by Emily Morris, Kevin He, Yanming Li, Yi Li, and Jian Kang

Abstract High-dimensional variable selection in the proportional hazards (PH) model has many successful applications in different areas. In practice, data may involve confounding variables that do not satisfy the PH assumption, in which case the stratified proportional hazards (SPH) model can be adopted to control the confounding effects by stratification without directly modeling the confounding effects. However, there is a lack of computationally efficient statistical software for high-dimensional variable selection in the SPH model. In this work an R package, **SurvBoost**, is developed to implement the gradient boosting algorithm for fitting the SPH model with high-dimensional covariate variables. Simulation studies demonstrate that in many scenarios **SurvBoost** can achieve better selection accuracy and reduce computational time substantially compared to the existing R package that implements boosting algorithms without stratification. The proposed R package is also illustrated by an analysis of gene expression data with survival outcome in The Cancer Genome Atlas study. In addition, a detailed hands-on tutorial for **SurvBoost** is provided.

Introduction

Variable selection for high-dimensional survival data has become increasingly important in a variety of research areas. One of the most popular methods is based on the proportional hazards (PH) model. Many penalized regression methods including adaptive lasso and elastic net have been proposed for the PH model (Tibshirani, 1997; Simon et al., 2011; Goeman, 2010). Alternatively, boosting described by Bühlmann and Yu (2010) has been adopted for variable selection in regression models and the PH model via gradient descent techniques. It can have a better variable selection accuracy compared with other methods in many scenarios. The R package **mboost** has been developed and become a powerful tool for variable selection and parameter estimation in complex parametric and nonparametric models via the boosting methods (Hothorn et al., 2017). It has been widely used in many applications.

However, in many biomedical studies, the collected data may involve confounding variables that do not satisfy the PH assumption. For example, in cancer research you may argue that gender effects are not proportional, but we are more interested in selecting genes as the important risk factors for cancer survival. The PH assumption can reasonably be imposed on modeling the gene effects but not for gender effects. In this case the stratified proportional hazards (SPH) models are needed. In particular, the data are often grouped into multiple strata according to confounding variables. The SPH model adjusts those confounding effects by fitting the Cox regression with different baseline hazards for different strata, while still assuming that the covariate effects of interest are the same across different strata and satisfy the proportional hazard assumption.

The SPH model has a wide range of applications for survival analysis, but no computationally efficient statistical software are available for high-dimensional variable selection in the SPH model. To fill this gap, we develop an R package, **SurvBoost**, to implement the gradient boosting algorithm for fitting the SPH model with high-dimensional covariates with adjusting confounding variables. **SurvBoost** implements the gradient descent algorithm for fitting both PH and SPH model. The algorithm for the PH model has been used for the additive Cox model in the **mboost** package, which cannot fit the SPH model to perform variable selection. The **survival** package is capable of performing model fitting for the SPH model, but does not implement variable selection desired in the high-dimensional setting. In our **SurvBoost** package, we optimize the implementations which can reduce 30%–50% computational time. Additional options are available in the **SurvBoost** package to determine an appropriate stopping criteria for the algorithm. Another useful function assists in selecting stratification variables, which may improve model fitting results.

The rest of the paper is organized as follows: In Section 2, we will provide a brief overview of the gradient boosting method for the SPH model along with the algorithm stopping criteria. In Section 3, we show that **SurvBoost** can achieve a better selection accuracy and reduce computational time substantially compared with **mboost**. In Section 4, we provide a detailed hands-on tutorial for

SurvBoost. In Section 5, we illustrate the proposed R package on an analysis of the gene expression data with survival outcome in The Cancer Genome Atlas (TCGA) study.

Methods

Stratified proportional hazards model

The Cox proportional hazards model is effective for modeling survival outcomes in many applications. An important assumption underlying this model is a constant hazard ratio, meaning that the hazard for one individual is proportional to that of any other individual. This is a strong assumption for many applications. Thus, one useful adaptation to this model is relaxing the strict proportional hazards assumption; one approach is to allow the baseline hazard to differ by group across the observations. This is known as the stratified proportional hazards (SPH) model.

Suppose the dataset consists of n subjects. For $i = 1, \dots, n$, denote by T_i the observed time of event or censoring for subject i , and δ_i indicates whether or not an event occurred for subject i . Denote by G the total number of strata and by n_g the number of subjects in stratum g . Let g_i be the strata indicator for subject i . Suppose there are p potential covariate variables of our interest to select. For $j = 1, \dots, p$, let x_{ij} be the covariate j for subject i . For stratum $g = 1, \dots, G$, the hazard of subject i at time t in stratum g becomes

$$h_g(t, X_{i,g}) = h_{0,g}(t) \exp \left\{ X_{i,g}^T \beta \right\},$$

where $h_{0,g}(t)$ is the baseline hazard function, $X_{i,g}$ is a vector of covariates and β is the regression coefficients of interest.

Allowing the baseline hazard to differ across strata allows flexibility often desired when proportional hazards is too strong. The SPH model can control effects of confounding variables through this stratification. The estimates of the effect of covariates remain constant across strata, so the model is still interpretable across all subjects.

Gradient boosting for SPH

The log partial likelihood of the SPH model is

$$\ell(\beta) = \sum_{i=1}^n \delta_i \left\{ X_{i,g}^T \beta - \log \left(\sum_{\ell \in R_{i,g}} \exp \{X_{\ell,g}^T \beta\} \right) \right\},$$

where $\beta = (\beta_1, \dots, \beta_p)^\top$, $X_i = (X_{i1}, \dots, X_{ip})^\top$ and $R_{ig} = \{\ell : T_\ell \geq T_i, g_\ell = g\}$ for all i with $g_i = g$ representing the set of at risk subjects in group g . We adopt the following gradient boosting algorithm to find the maximum partial likelihood estimate (MPLE). Let $S_{kg}(i, j) = \sum_{\ell \in R_{ig}} X_{\ell,j}^k \exp \{X_{\ell,g}^T \beta\}$ for $k = 0, 1, 2$.

Data: $\{T_i, \delta_i, g_i, X_i\}_{i=1}^n$; Number of iterations M ; Updating rate v
Result: β .

```

1 begin
2   Initialize  $\beta_j = 0$  ( $j = 1, \dots, p$ ).
3   for  $m = 1, \dots, M$  do
4     for  $j = 1, \dots, p$  do
5       Compute the first partial derivative with respect to  $j$ :
6        $L_1(j) = \sum_{i=1}^n \sum_{g=1}^G I_{[g_i=g]} \delta_i \{X_{ij} - S_{1g}(i, j)/S_{0g}(i, i)\}.$ 
7     end
8     Find  $j^* = \text{argmax}_j |L_1(j)|$ .
9     Calculate the second partial derivative with respect to  $j^*$ :
10     $L_2(j^*) = \sum_{i=1}^n \sum_{g=1}^G I_{[g_i=g]} \delta_i \left[ \frac{S_{2g}(i, i)}{S_{0g}(i, i)} - \left\{ \frac{S_{1g}(i, j^*)}{S_{0g}(i, i)} \right\}^2 \right]$ 
11    Update  $\beta_{j^*} = \beta_{j^*} + v L_2(j^*)^{-1} L_1(j^*)$ 
12  end
13 end

```

Algorithm 2: Boosting gradient descent algorithm

This algorithm updates variables one at a time, by selecting the variable which maximizes the first partial derivative. The number of iterations is important for ensuring a sufficient number of updates to the β estimates, in addition to selecting the true signals (He et al., 2016). Note that this algorithm is slightly different than the one implemented in the **mboost** package even in the unstratified case, which we will use for comparison in the simulation setting. Bühlmann and Hothorn's algorithm uses only the first derivative to update the estimated β values (Bühlmann and Hothorn, 2007).

Stopping criteria

Selection of the number of boosting iterations is important. Over-fitting can occur if the number of iterations is too large (Jiang, 2004). Additionally the stopping criteria is important for accuracy of the coefficient estimates, since each iteration contributes to updating the estimate of one coefficient. The algorithm is less sensitive to the step size (Bühlmann and Hothorn, 2007).

SurvBoost provides several options for optimizing the number of iterations including: k -fold cross validation, Bayesian information criteria, change in likelihood, or specifying the number of variables to select.

The Bayesian Information Criteria (BIC) is one approach for selecting the optimal number of boosting iterations.

$$BIC = -2 \{l_j(\hat{\theta}_j) - l_0(\hat{\theta}_0)\} + (p_j - p_0) \log(d), \quad (1)$$

where $l_j(\hat{\theta}_j)$ is the maximized likelihood for a model with p_j selected variables and $l_0(\hat{\theta}_0)$ is the maximized likelihood for the reference model with p_0 selected variables. The number of uncensored events is d . Volinsky and Raftery (2000) argue that replacing the sample size, n , with d in the BIC calculation has better properties when dealing with censored survival models.

The extended BIC is also useful in high dimensional cases; this approach penalizes for greater complexity

$$EBIC = -2 l_j(\hat{\theta}_j) + p_j \log(d) + 2\gamma \log \binom{p}{p_j}, \quad (2)$$

where $\binom{p}{p_j}$ is the size of the class of models that model j belongs to, p is the total number of variables. The value of γ is fixed between 0 and 1, selected to penalize at the appropriate rate. Selecting 0 will reduce this to the standard BIC; EBIC and BIC are implemented jointly in the package using this connection to reduce from EBIC to BIC. AIC is available as well as a stopping criteria, although this information might not be as effective in the high dimensional setting.

Cross validation is another approach which may be used to determine the stopping point. The goodness of fit function is calculated as suggested by Simon et al. (2011). It is the log-partial likelihood of all the data using the optimal β determined with data excluding fold k (β_{-k}) minus the log-partial likelihood excluding fold k (ℓ_{-k}) of the data with the same β .

$$CV_k(m) = -[\ell\{\beta_{-k}(m)\} - \ell_{-k}\{\beta_{-k}(m)\}], \quad (3)$$

Where m is the current number of iterations and k indicates the subset of data being excluded.

Change in likelihood is another approach incorporated in the package. This method stops iterating once a small change in likelihood, specified in the function, is reached.

$$\Delta\ell = -[\ell(\beta(m)) - \ell(\beta(m+1))] < \alpha, \quad (4)$$

Where α is a small constant. Default change in likelihood, used in simulations, is a change of 0.001.

Simulation studies

This section compares the variable selection performance to a competing R package, **mboost** (Hofner et al., 2014).

Stratified Data Stratified data was simulated such that censoring rates were relatively constant across groups and the expected survival time differed by group. These assumptions mimic realistic settings such as those encountered with data grouped by hospital or facility.

For this simulation 1,500 observations were generated into ten strata; each strata had a different baseline hazard following a Weibull distribution. The Weibull distribution shape parameter was 3 for all strata, and the scale parameter varied across strata from e^{-1} to e^{-15} with ten evenly spaced

intervals. There were 100 true signals among 4,000 variables with true magnitude of 2 or -2. There was uniform censoring from time 0 to 200. Fifty of these data sets were generated.

The following example demonstrates the importance of the stopping criteria. **SurvBoost** has five options for specifying the number of iterations as described in the methods section. Selecting an appropriate number of iterations depends on the goals of the analysis. For example, if the goal is to achieve high sensitivity cross validation or extended BIC may be the best approach.

The performance of different stopping criteria are compared based on several selection and estimation measures: sensitivity (Se), specificity (Sp), false discovery rate (FDR), and mean squared error (MSE). FDR is calculated as the ratio of false positives over the total number of selected variables. Sensitivity, specificity, and FDR aim to address the performance of the variable selection, while MSE aims to address the accuracy of the coefficient estimates.

This simulation presents the performance of **SurvBoost** compared to the R package **mboost**. The boosting algorithm implemented in **mboost** is very similar to that of **SurvBoost** but does not allow stratification. We will compare results between the two packages using only a fixed number of iterations as the stopping rule; **mboost** has K-fold cross validation available for some settings but no longer provides it for Cox PH models. All of the stopping methods implemented in **SurvBoost** are not available in **mboost**. The performance can be compared by measures such as sensitivity and mean squared error. Table 1 presents the results of 50 simulated data sets, comparing the boosting algorithm using several different stopping procedures and to results from **mboost**. In this simulation, **mboost** selects fewer variables on average resulting in fewer false positives and more false negatives. Additionally the mean squared error is slightly higher than that of all the **SurvBoost** options.

Runtime is also an important factor with this algorithm. Stratification speeds up the algorithm as seen in the first simulation. All runtimes were generated on a MacBook with 2.9GHz Intel Core i5 and 16GB memory.

	stopping method	number selected	Se	Sp	FDR	MSE	number of iterations	runtime (seconds)
SurvBoost	fixed	111 (5)	0.90 (.02)	0.99 (.00)	0.19 (.04)	382 (1)	500 (0)	163 (20)
mboost		99 (4)	0.82 (.03)	1.00 (.00)	0.17 (.04)	387 (1)	500 (0)	396 (254)
SurvBoost	cv	379 (24)	1.00 (.00)	0.93 (.01)	0.74 (.02)	333 (3)	3896 (275)	5742 (595)
SurvBoost	# selected	101 (0)	0.84 (.03)	1.00 (.00)	0.17 (.03)	385 (2)	385 (40)	163 (24)
SurvBoost	likelihood	121 (6)	0.95 (.02)	0.99 (.00)	0.21 (.04)	378 (1)	632 (15)	242 (33)
SurvBoost	EBIC	140 (7)	0.99 (.01)	0.99 (.00)	0.29 (.04)	370 (1)	993 (10)	624 (60)

Table 1: Results from simulation with approximately 1,500 observations in 10 strata and 4,000 variables to be selected. The table presents averages with the standard deviation, in parentheses, from 50 simulated datasets. Sensitivity (Se) is calculated as the proportion of true positives out of the total number of true signals. Specificity (Sp) is calculated as the proportion of true negatives out of the total number of variables that are not true signals. The false discovery rate (FDR) is the proportion of false positives in the total number of selected variables.

To further demonstrate the importance of using the SPH model, we compared results of modeling with and without stratification for data simulated with ten strata. In this setting the true signal for all 100 variables is 0.75 to illustrate the performance with a smaller effect size.

From this example we can evaluate the importance of using the stratified model. This case demonstrates that when not stratifying by group the model is not as sensitive to the true signals, resulting in lower sensitivity. We also observe a larger or similar number of variables selected, meaning that there are a larger number of false positives when ignoring the stratification. Depending on the context, a larger number of false positives may be very undesirable. The algorithm implemented for the Cox PH model is slightly different than the one used in **SurvBoost**, which can be seen here by the difference in the two unstratified rows.

Unstratified Data Another simulation was used to compare performance of our method to **mboost** when stratification is not necessary for appropriate modeling. Similarly to the stratified case, four thousand variables were generated for 1,500 observations but without stratification. The baseline hazard followed a Weibull distribution, with shape parameter equal to 5 and scale equal to \exp^{-5} . The true β contained 100 true signals of magnitude 2 or -2 out of 4,000 variables.

We can observe in Table 2 that **SurvBoost** performs similarly to **mboost** under these conditions. **mboost** tends to select fewer variables than **SurvBoost**, so in this simulation **mboost** has fewer false positives and more false negatives compared to **SurvBoost**.

	Stopping Method	number selected	Se	Sp	FDR	MSE	number of iterations	runtime (seconds)
Stratified	fixed	116 (7)	0.71 (.04)	0.95 (.01)	0.39 (.04)	49 (0)	500 (0)	14 (1)
	cv	198 (16)	0.90 (.04)	0.88 (.02)	0.54 (.03)	50 (2)	1585 (316)	263 (60)
	# selected	100 (0)	0.65 (.04)	0.96 (.00)	0.36 (.04)	50 (1)	391 (42)	11 (2)
	likelihood	112 (8)	0.69 (.04)	0.95 (.01)	0.38 (.03)	49 (1)	472 (33)	14 (2)
	EBIC	161 (9)	0.84 (.03)	0.91 (.01)	0.48 (.04)	44 (1)	41 (45)	29 (3)
Unstratified	SB fixed	122 (7)	0.67 (.04)	0.94 (.01)	0.45 (.04)	49 (1)	500 (0)	13 (1)
	mboost	58 (5)	0.41 (.03)	0.98 (.00)	0.30 (.06)	53 (0)	500 (0)	13 (1)

Table 2: Results from simulation with approximately 1,000 observations and 1,000 possible variables for selection with 100 true signals. The table presents averages with the standard deviation from 50 simulated datasets. All methods in this table were run using the **SurvBoost** package except for the unstratified mboost row.

	stopping method	number selected	Se	Sp	FDR	MSE	number of iterations	runtime (seconds)
SurvBoost mboost	fixed	104 (3)	0.94 (.02)	1.00 (.00)	0.10 (.02)	381 (.45)	500 (0)	138 (11)
		98 (4)	0.89 (.03)	1.00 (.00)	0.10 (.03)	384 (.36)	500 (0)	220 (198)
SurvBoost	cv	141 (7)	1.00 (.00)	0.99 (.00)	0.29 (.04)	298 (1)	5010 (0)	6531 (18)
SurvBoost	# selected	100 (0)	0.91 (.02)	1.00 (.00)	0.10 (.02)	382 (2)	452 (62)	151 (18)
SurvBoost	likelihood	109 (3)	0.98 (.01)	1.00 (.00)	0.10 (.03)	382 (.54)	668 (14)	216 (18)
SurvBoost	EBIC	112 (4)	0.99 (.01)	1.00 (.00)	0.11 (.03)	366 (1)	999 (.25)	530 (27)

Table 3: Results from simulation with approximately 1,500 observations and 4,000 possible variables for selection with 100 true signals. The table presents averages with the standard deviation from 50 simulated datasets.

Illustration of package

This section provides a brief tutorial on how to use this package based on simulated data. In order to install the package, several other R packages must be installed. The code relies on **Rcpp**, **RcppArmadillo**, and **RcppParallel** in order to improve computational speed (Eddelbuettel et al., 2018a,b; Allaire et al., 2018). Additionally the **survival** package is used for simulation and post selection refitting for inference and will be required for installation of **SurvBoost** (Therneau, 2017). If working on a Windows machine, installing Rtools is also necessary. The following line of R code installs the package from CRAN.

```
R > install.packages("SurvBoost")
```

Model fitting

The **boosting_core()** function requires similar inputs to the familiar **coxph()** function from the package **survival**.

```
boosting_core(formula, data = matrix(), rate = 0.01, num_iter = 500, ...)
```

The input **formula** has the form **Surv(time, death) ~ variable1 + variable2**. The input **data** is in matrix form or a data frame. Two additional parameters must be specified for the boosting algorithm: **rate** and **num_iter**. **Rate** is the step size in the algorithm, although choice of this may not impact the performance too significantly (Bühlmann and Hothorn, 2007), default value is set to 0.01. Selecting an appropriate number of iterations to run the algorithm will, however, have a greater impact on the results. The last input **num_iter** is used to determine the number of iterations to run the algorithm, default value is 500.

Simple example

We present a simple example demonstrating the convenience of using the package for stratified data. We simulate survival data for five strata with different constant baseline hazards.

Call	Method
boosting_core(formula, data)	fixed mstop = 500
boosting_core(formula, data, num_iter=1000)	fixed mstop = specified value
boosting_core(formula, data, control_method="cv")	10-fold cross validation
boosting_core(formula, data, control_method="num_selected", control_parameter = 5)	number selected, need to specify number of variables
boosting_core(formula, data, control_method="likelihood")	change in likelihood
boosting_core(formula, data, control_method="BIC")	minimum BIC or EBIC
boosting_core(formula, data, control_method="AIC")	minimum AIC

Table 4: Stopping criteria options for boosting_core function.

Function	Result
summary.boosting()	prints summary of variable selection and estimation
plot.boosting()	plots variable selection frequency
predict.boosting()	generates predicted hazard ratio for each observation or new data
post.selection.fitting.boosting()	refits model with only subset of selected covariates

Table 5: Functions available in **SurvBoost** package. Every function accepts a boosting object input to generate the corresponding result.

```
R > TrueBeta
[1] 0.5 0.5 0.0 0.0 0.0 -0.5 0.5 0.5 0.0 0.0
R > set.seed(123)
R > data_small <- simulate_survival_cox(true_beta=TrueBeta,
   base_hazard="auto",
   num_strata=5,
   input_strata_size=100, cov_structure="ar",
   block_size=5, rho=0.6, censor_dist="unif",
   censor_const=2, tau=Inf, normalized=F)
```

We have $p = 10$ and $|\beta_j|$ ranges from 0 to 0.5. There are five “facilities” with average size of 100 each representing one stratum, and n is approximately 500. The covariance structure within the blocks is AR(1) with correlation 0.6. The censoring rate is about 33%. In this case the variable *strata_idx* indicates the variable to stratify on in the survival model; each “facility” in this simulated data has a different baseline hazard function.

Another feature of the package assists with determining variables to stratify on if this information is unknown. The function *strata.boosting* will print box plots and a summary table of the survival time grouped by splits in the specified variable. The variable can be categorical or continuous; if continuous, the function will split on the median value to demonstrate whether there appears to be a difference in the survival time distribution for the two groups. This information alone does not suggest that stratification on this variable is necessary. It is intended to be a tool to confirm if there are differences seen across groups, when stratification is anticipated to be necessary.

```
R > strata.boosting(data_small$strata_idx, data_small$time)

  as.factor(x)      Min       Q1     Median       Q3      Max
1          1 0.0046772744 0.1163388 0.3108169 1.096236 1.693283
2          2 0.0005600448 0.1422992 0.5849665 1.270754 1.951286
3          3 0.0057943145 0.1371938 0.9125127 1.314191 1.989180
4          4 0.0042511208 0.1998902 0.5797646 1.437124 1.960646
5          5 0.0015349222 0.1283325 0.5896426 1.325094 1.873137
```

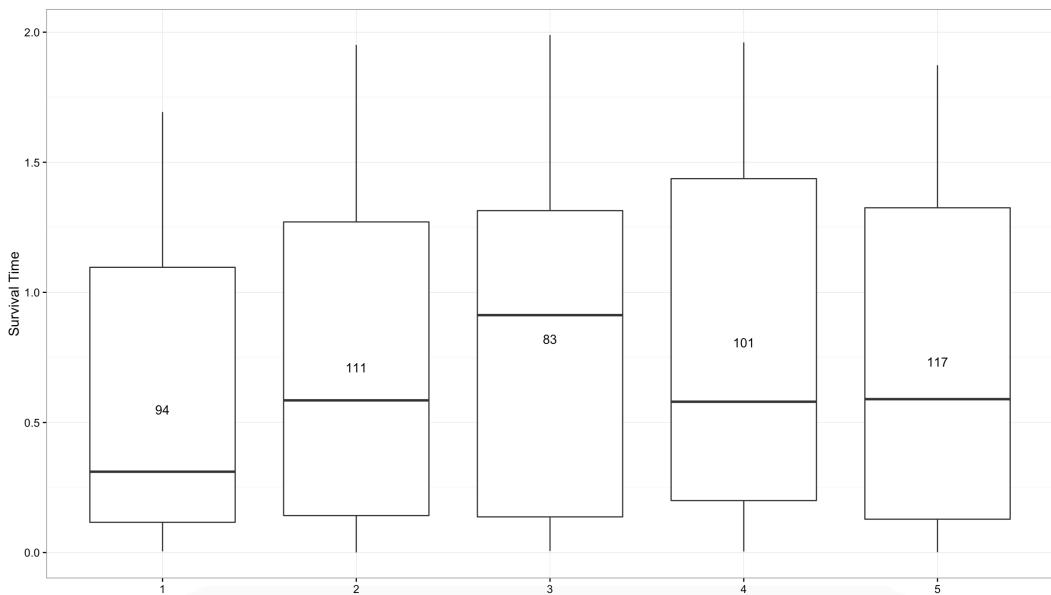


Figure 1: Box plots of survival time by strata index in simulated data generated by the function `strata.boosting`.

Simulated data includes a vector of survival or censoring time, `time`, indicator of an event, `delta`, and matrix of covariates, `Z`. Then generate the formula including all possible variables for selection.

```
R > time <- data_small$time
R > delta <- data_small$delta
R > Z <- as.matrix(data_small[,-c(1,2,3)])

R > covariates <- paste("strata(strata_idx)+", paste(colnames(Z),
  collapse = "+"))
R > formula <- as.formula(paste("Surv(time,delta)~", covariates))
```

Run the `boosting_core()` function to obtain the variables selected. This example uses the number of iterations control as a fixed input of 75 and update rate of 0.1.

```
R > test1 <- boosting_core(formula,
+                           data=data_small,
+                           rate=0.1,
+                           num_iter=75)
R > summary.boosting(test1)

Call:
boosting_core(formula = formula, data = data_small, rate = 0.1,
  num_iter = 75)

data: data_small

n = 506
Number of events = 371
Number of boosting iterations: mstop = 75
Step size = 0.1

Coefficients:
      V1        V2        V6        V7        V8 
0.4486589 0.3676104 -0.2150421  0.2384806  0.4728502
```

Function `summary.boosting()` displays the variables which are selected as well as the coefficient estimates and the number of boosting iterations performed. Set the argument `all_beta = TRUE` to see all the variables, not just those selected.

To use a different method for the number of boosting iterations use the arguments `control_method` and `control_parameter`. The value of `control_parameter` should be a list containing the value of the parameter(s) corresponding to the method specified by `control_method`. For example,

```
R > test2 <- boosting_core(formula, data=data_small, rate=0.1,
  control_method="num_selected", control_parameter=list(num_select=5))
R > summary.boosting(test2)
Call:
boosting_core(formula = formula, data = data_small, rate = 0.1,
  control_method = "num_selected", control_parameter = list(num_select = 5))

data: data_small

n = 506
Number of events = 371
Number of boosting iterations: mstop = 104
Step size = 0.1

Coefficients:
      V1        V2        V6        V7        V8
0.11828718 0.11021464 -0.05292158 0.25561965 0.05199151

Number of iterations: 10
```

This option iterates until the specified number of variables, 5 in this example, are selected. See methods for other stopping criteria. Note that in the package BIC and EBIC are available jointly in one option when *control_method* is set to "BIC". Setting the parameter $\gamma = 0$ will reduce the EBIC penalty to the BIC penalty. In order to implement EBIC, a nonzero value of gamma should be specified in *control_parameter*.

The `plot.boosting()` function displays a plot of the selection frequency by the number of iterations. Another option of the `plot.boosting()` function is to plot the coefficient paths of each variable by the number of boosting iterations. See Figures 2 and 3.

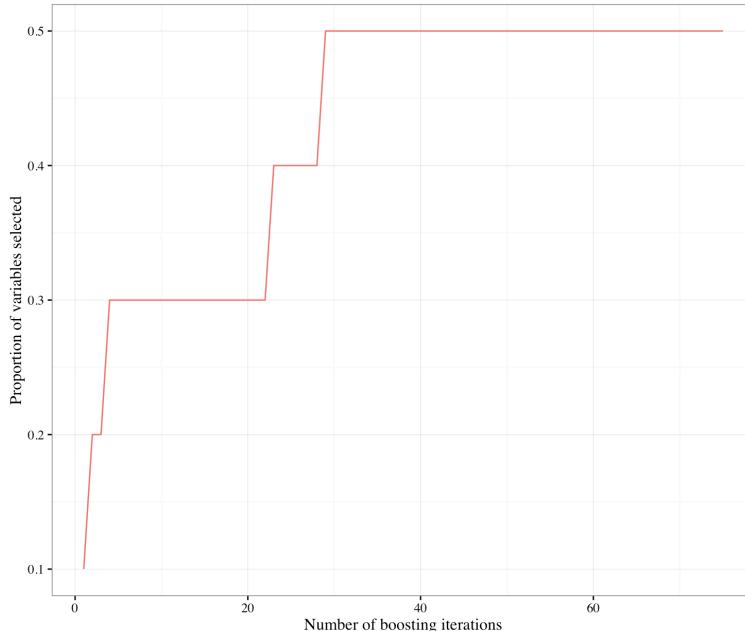


Figure 2: Plot generated by `plot.boosting` function, variable selection frequency by number of boosting iterations.

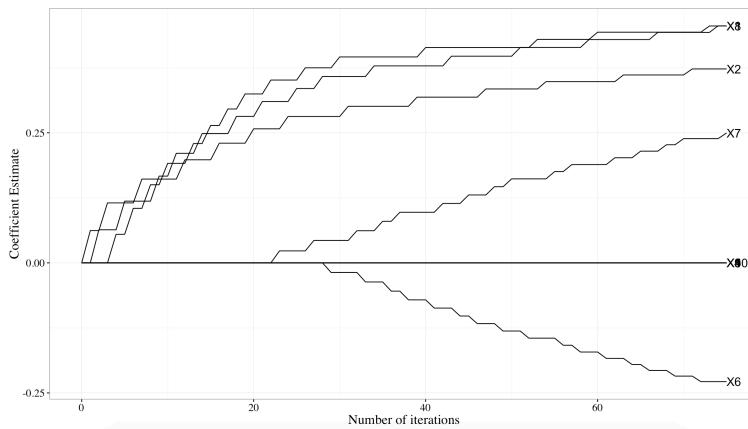


Figure 3: Plot generated by `plot.boosting` function with option “coefficients,” coefficient paths for variables selected by number of boosting iterations.

The function `predict.boosting()` provides an estimate of the hazard ratio for each observation in the dataset provided, relative to the average of p predictors.

```
R > predict.boosting(test1)[1:6]
46.385476 1.823920 42.049932 16.427860 4.013200 2.243711
```

The model selected using boosting can be refit with `coxph()` for post selection inference. The function `post.selection.fitting.boosting()` will perform this refitting and output the coefficient estimates with corresponding standard errors and p-values. Note that the statistical inferences performed here are conditional on the variable selection results. The interpretation of p-values and standard errors are fundamentally different from the regular unconditional statistical inferences. The performance of conditional statistical inferences is highly dependent on the variable selection accuracy. In our case, it depends on the choice of stopping rule.

```
R > summary.test1 <- summary.boosting(test1)
R > fmla <- summary.test1$formula
R > post.selection.fitting.boosting(fmla, data=data_small)
Call:
coxph(formula = fmla, data = data)

n = 506, number of events = 371

      coef exp(coef) se(coef)     z Pr(>|z|)
V1  0.59181  1.80726  0.07454  7.940 2.00e-15 ***
V2  0.48079  1.61736  0.06948  6.920 4.53e-12 ***
V6 -0.51830  0.59553  0.07145 -7.254 4.05e-13 ***
V7  0.51108  1.66709  0.08479  6.028 1.66e-09 ***
V8  0.54758  1.72907  0.07116  7.695 1.42e-14 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

      exp(coef) exp(-coef) lower .95 upper .95
V1    1.8073    0.5533   1.5616   2.0915
V2    1.6174    0.6183   1.4114   1.8533
V6    0.5955    1.6792   0.5177   0.6851
V7    1.6671    0.5998   1.4118   1.9685
V8    1.7291    0.5783   1.5040   1.9879

Concordance= 0.762  (se = 0.036 )
Rsquare= 0.487  (max possible= 0.997 )
Likelihood ratio test= 338.1 on 5 df,  p=0
Wald test            = 287.8 on 5 df,  p=0
Score (logrank) test = 299.1 on 5 df,  p=0
```

TCGA data example

Data from three breast cancer cohorts was used to demonstrate this method on data outside of the simulation framework. There were 578 patients included in the combined data, with 8,864 variables measured for each patient: 8,859 genes and 5 phenotypic variables. The phenotype variables included age at diagnosis, tumor size, cancer stage, progesterone-receptor status, and estrogen-receptor status. The data can be downloaded from The Cancer Genome Atlas (TCGA) (Naderi et al., 2006; Chin et al., 2006; Miller et al., 2005) or from GitHub using the following R code.

```
R > library(piggyback)
R > pb_download("data.tcga.tsv.gz",
                 repo = "EmilyLMorris/survBoost")
R > data <- read_tsv("data.tcga.tsv")
```

The patients were split into two cohorts depending on their cancer stage and tumor size. One cohort contained patients with a less severe prognosis, cancer stage of one and tumor size less than the median; the other cohort contained those with cancer stage greater than one and/or with a tumor larger than the median size.

```
R > fit.plot <- survfit(Surv(survival_time, survival_ind) ~ as.factor(severity), data=data)
R > ggsurvplot(fit.plot,
                conf.int = TRUE,
                risk.table = TRUE,
                risk.table.col="strata",
                ggtheme = theme_bw(), palette = "grey")
```

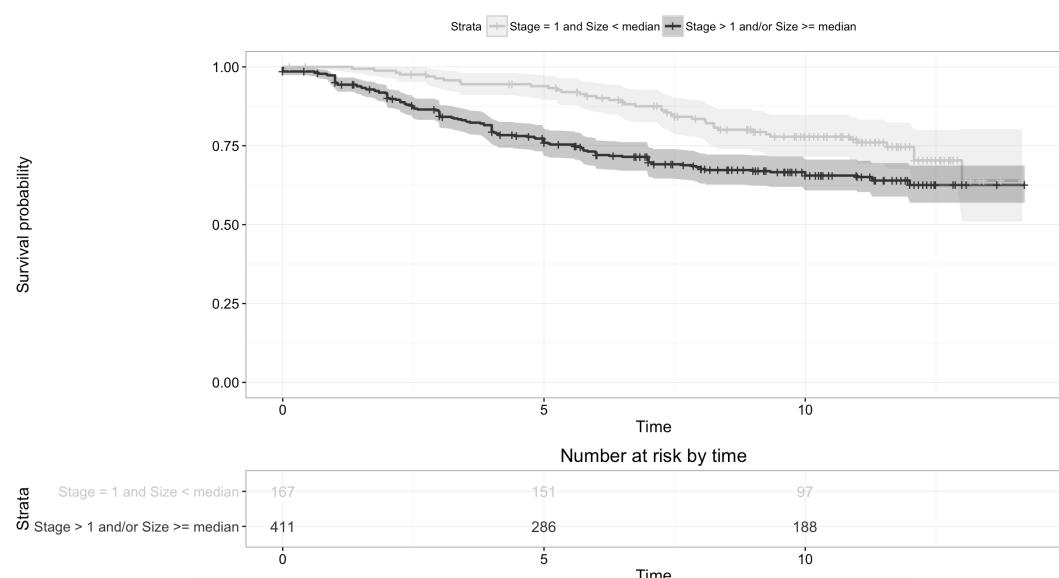


Figure 4: Survival curves for the two strata based on cancer stage and tumor size. This plot demonstrates that the proportional hazards assumption may not hold in this case. Stratifying based on this criteria generates the following results.

Using stability selection (Meinshausen and Bühlmann, 2010), 14 variables were identified with selection frequencies greater than 50% from 50 iterations of subsampling. Age and progesterone-receptor status were selected in addition to 12 genes. The boosting algorithm was performed with the number of iterations fixed at the sample size of 578.

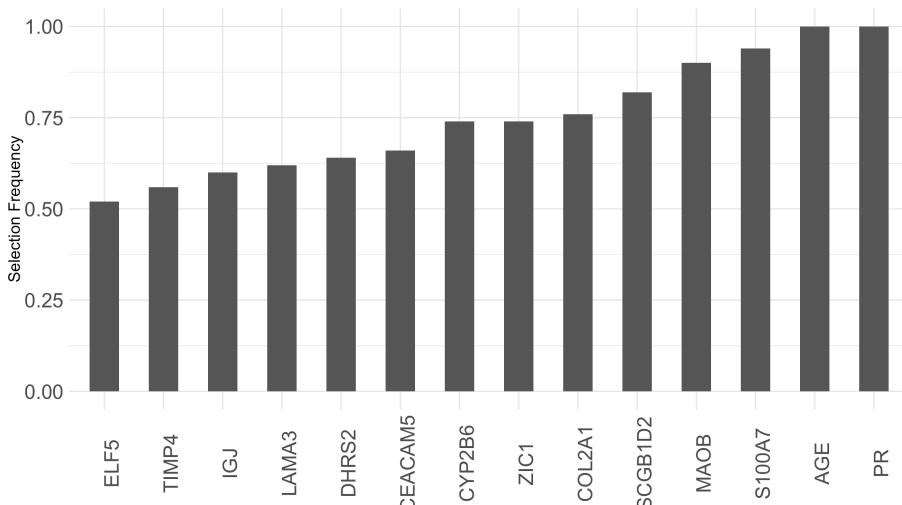


Figure 5: Selection frequencies for genes or phenotype variables that were selected at least 50% of the time with stability selection.

Several of the genes selected in this analysis have been previously identified as having an association with breast cancer. Psoriasin (S100A7) has been associated with breast cancer (Al-Haddad et al., 1999). Several studies have found COL2A1 to be part of gene signatures for predicting tumor recurrence (Yu et al., 2007; Wang et al., 2005). Other genes selected that have been identified as part of a gene signature or association with breast cancer tumor progression risk include: ZIC1 (Boersma et al., 2008), CYP2B6 (Tozlu et al., 2006), ELF5 (Chakrabarti et al., 2012), IGJ (Boersma et al., 2008), DHRS2 (Krijgsman et al., 2012), and CEACAM5 (Blumenthal et al., 2007). **Mboost** using the same criteria but without a stratified model only identifies one gene of importance, MC2R, demonstrating the utility of the SPH model in this context.

Conclusion

In this article, we introduce a new R package **SurvBoost** which implements the gradient boosting algorithm for high-dimensional variable selection in the stratified proportional hazards (SPH) model, while most existing R packages, such as **mboost** only focus on the proportional hazards model. In the simulation studies, we show that **SurvBoost** can improve the model fitting and achieve better variable selection accuracy for the data with stratified structures. In addition, we optimize the implementations of the gradient boosting in both the SPH and the PH models. For the PH model fitting, **SurvBoost** can reduce about 30%-50% computational time compared to **mboost**. In the future, we plan to extend the package to handle more complex survival data such as left-truncation data and interval censoring data.

Acknowledgments

The authors would like to thank the editor and reviewers for suggestions that led to an improved manuscript. This work was partially supported by the NIH grants R01MH105561 (Kang) and R01GM124061 (Kang).

Bibliography

- S. Al-Haddad, Z. Zhang, E. Leygue, L. Snell, A. Huang, Y. Niu, T. Hiller-Hitchcock, K. Hole, L. C. Murphy, and P. H. Watson. Psoriasin (s100a7) expression and invasive breast cancer. *The American journal of pathology*, 155(6):2057–2066, 1999. [p]
- J. Allaire, R. Francois, K. Ushey, G. Vandenbrouck, M. Geelnard, and Intel. *RcppParallel: Parallel Programming Tools for 'Rcpp'*, 2018. URL <https://CRAN.R-project.org/package=RcppParallel>. R package version 4.4.0. [p]
- R. D. Blumenthal, E. Leon, H. J. Hansen, and D. M. Goldenberg. Expression patterns of ceacam5 and ceacam6 in primary and metastatic cancers. *BMC Cancer*, 7(1):2, Jan 2007. doi: 10.1186/1471-2407-7-2. URL <https://doi.org/10.1186/1471-2407-7-2>. [p]

- B. J. Boersma, M. Reimers, M. Yi, J. A. Ludwig, B. T. Luke, R. M. Stephens, H. G. Yfantis, D. H. Lee, J. N. Weinstein, and S. Ambs. A stromal gene signature associated with inflammatory breast cancer. *International Journal of Cancer*, 122(6):1324–1332, 2008. doi: 10.1002/ijc.23237. URL <https://doi.org/10.1002/ijc.23237>. [p]
- P. Bühlmann and T. Hothorn. Boosting algorithms: Regularization, prediction and model fitting (with discussion). *Statistical Science*, 22(4):477–505, 2007. [p]
- P. Bühlmann and B. Yu. Boosting. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2(1):69–74, 2010. doi: 10.1002/wics.55. URL <https://doi.org/10.1002/wics.55>. [p]
- R. Chakrabarti, J. Hwang, M. A. Blanco, Y. Wei, M. Lukacisin, R.-A. Romano, K. Smalley, S. Liu, Q. Yang, T. Ibrahim, L. Mercatali, D. Amadori, B. G. Haffty, S. Sinha, and Y. Kang. Elf5 inhibits the epithelial-mesenchymal transition in mammary gland development and breast cancer metastasis by transcriptionally repressing snail2. *Nature Cell Biology*, 14:1212–1222, 2018/2/7/2012. [p]
- K. Chin, S. DeVries, J. Fridlyand, P. T. Spellman, R. Roydasgupta, W.-L. Kuo, A. Lapuk, R. M. Neve, Z. Qian, T. Ryder, F. Chen, H. Feiler, T. Tokuyasu, C. Kingsley, S. Dairkee, Z. Meng, K. Chew, D. Pinkel, A. Jain, B. M. Ljung, L. Esserman, D. G. Albertson, F. M. Waldman, and J. W. Gray. Genomic and transcriptional aberrations linked to breast cancer pathophysiosities. *Cancer Cell*, 10(6):529–541, 2017/12/18 2006. doi: 10.1016/j.ccr.2006.10.009. URL <https://doi.org/10.1016/j.ccr.2006.10.009>. [p]
- D. Eddelbuettel, R. Francois, J. Allaire, K. Ushey, Q. Kou, N. Russell, D. Bates, and J. Chambers. *Rcpp: Seamless R and C++ Integration*, 2018a. URL <https://CRAN.R-project.org/package=Rcpp>. R package version 0.12.15. [p]
- D. Eddelbuettel, R. Francois, D. Bates, and B. Ni. *RcppArmadillo: 'Rcpp' Integration for the 'Armadillo' Templated Linear Algebra Library*, 2018b. URL <https://CRAN.R-project.org/package=RcppArmadillo>. R package version 0.8.400.0.0. [p]
- J. J. Goeman. L1 penalized estimation in the cox proportional hazards model. *Biometrical Journal*, 52(1):70–84, 2010. doi: 10.1002/bimj.200900028. [p]
- K. He, Y. Li, J. Zhu, H. Liu, J. E. Lee, C. I. Amos, T. Hyslop, J. Jin, H. Lin, Q. Wei, and Y. Li. Component-wise gradient boosting and false discovery control in survival analysis with high-dimensional covariates. *Bioinformatics*, 32(1):50–57, 2016. doi: 10.1093/bioinformatics/btv517. URL <https://doi.org/10.1093/bioinformatics/btv517>. [p]
- B. Hofner, A. Mayr, N. Robinzonov, and M. Schmid. Model-based boosting in R: A hands-on tutorial using the R package mboost. *Computational Statistics*, 29:3–35, 2014. [p]
- T. Hothorn, P. Buehlmann, T. Kneib, M. Schmid, and B. Hofner. *mboost: Model-Based Boosting*, 2017. URL <https://CRAN.R-project.org/package=mboost>. R package version 2.8-1. [p]
- W. Jiang. Process consistency for adaboost. *Ann. Statist.*, 32(1):13–29, 02 2004. doi: 10.1214/aos/1079120128. URL <https://doi.org/10.1214/aos/1079120128>. [p]
- O. Krijgsman, P. Roepman, W. Zwart, J. S. Carroll, S. Tian, F. A. de Snoo, R. A. Bender, R. Bernards, and A. M. Glas. A diagnostic gene profile for molecular subtyping of breast cancer associated with treatment response. *Breast Cancer Research and Treatment*, 133(1):37–47, May 2012. doi: 10.1007/s10549-011-1683-z. URL <https://doi.org/10.1007/s10549-011-1683-z>. [p]
- N. Meinshausen and P. Bühlmann. Stability selection. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 72(4):417–473, 2010. doi: 10.1111/j.1467-9868.2010.00740.x. URL <http://dx.doi.org/10.1111/j.1467-9868.2010.00740.x>. [p]
- L. D. Miller, J. Smeds, J. George, V. B. Vega, L. Vergara, A. Ploner, Y. Pawitan, P. Hall, S. Klaar, E. T. Liu, and J. Bergh. An expression signature for p53 status in human breast cancer predicts mutation status, transcriptional effects, and patient survival. *Proceedings of the National Academy of Sciences of the United States of America*, 102(38):13550–13555, 2005. doi: 10.1073/pnas.0506230102. URL <https://doi.org/10.1073/pnas.0506230102>. [p]
- A. Naderi, A. E. Teschendorff, N. L. Barbosa-Morais, S. E. Pinder, A. R. Green, D. G. Powe, J. F. R. Robertson, S. Aparicio, I. O. Ellis, J. D. Brenton, and C. Caldas. A gene-expression signature to predict survival in breast cancer across independent data sets. *Oncogene*, 26:1507–1516, 08 2006. [p]

- N. Simon, J. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for cox's proportional hazards model via coordinate descent. *Journal of Statistical Software, Articles*, 39(5):1–13, 2011. doi: 10.18637/jss.v039.i05. URL <https://doi.org/10.18637/jss.v039.i05>. [p]
- T. M. Therneau. *survival: Survival Analysis*, 2017. URL <https://CRAN.R-project.org/package=survival>. R package version 2.41-3. [p]
- R. Tibshirani. The lasso method for variable selection in the cox model. *Statistics in medicine*, 16(4):385–395, 1997. [p]
- S. Tozlu, I. Girault, S. Vacher, J. Vendrell, C. Andrieu, F. Spyros, P. Cohen, R. Lidereau, and I. Bieche. Identification of novel genes that co-cluster with estrogen receptor alpha in breast tumor biopsy specimens, using a large-scale real-time reverse transcription-pcr approach. *Endocrine-Related Cancer*, 13(4):1109–1120, 2006. doi: 10.1677/erc.1.01120. URL <https://doi.org/10.1677/erc.1.01120>. [p]
- C. T. Volinsky and A. E. Raftery. Bayesian information criterion for censored survival models. *Biometrics*, 56(1):256–262, 2000. doi: 10.1111/j.0006-341X.2000.00256.x. URL <https://doi.org/10.1111/j.0006-341X.2000.00256.x>. [p]
- Y. Wang, J. G. Klijn, Y. Zhang, A. M. Sieuwerts, M. P. Look, F. Yang, D. Talantov, M. Timmermans, M. E. Meijer-van Gelder, J. Yu, et al. Gene-expression profiles to predict distant metastasis of lymph-node-negative primary breast cancer. *The Lancet*, 365(9460):671–679, 2005. [p]
- J. X. Yu, A. M. Sieuwerts, Y. Zhang, J. W. Martens, M. Smid, J. G. Klijn, Y. Wang, and J. A. Foekens. Pathway analysis of gene signatures predicting metastasis of node-negative primary breast cancer. *BMC Cancer*, 7(1):182, 2007. doi: 10.1186/1471-2407-7-182. URL <https://doi.org/10.1186/1471-2407-7-182>. [p]

Emily Morris

Department of Biostatistics

University of Michigan

1415 Washington Heights, Ann Arbor, MI 48109

E-mail: emorrisl@umich.edu

Kevin He

Department of Biostatistics

University of Michigan

1415 Washington Heights, Ann Arbor, MI 48109

E-mail: kevinhe@umich.edu

Yanming Li

Department of Biostatistics

University of Michigan

1415 Washington Heights, Ann Arbor, MI 48109

E-mail: liyanmin@umich.edu

Yi Li

Department of Biostatistics

University of Michigan

1415 Washington Heights, Ann Arbor, MI 48109

E-mail: yili@umich.edu

Jian Kang

Department of Biostatistics

University of Michigan

1415 Washington Heights, Ann Arbor, MI 48109

E-mail: jiankang@umich.edu

CoxPhLb: An R Package for Analyzing Length Biased Data under Cox Model

by Chi Hyun Lee, Heng Zhou, Jing Ning, Diane D. Liu and Yu Shen

Abstract Data subject to length-biased sampling are frequently encountered in various applications including prevalent cohort studies and are considered as a special case of left-truncated data under the stationarity assumption. Many semiparametric regression methods have been proposed for length-biased data to model the association between covariates and the survival outcome of interest. In this paper, we present a brief review of the statistical methodologies established for the analysis of length-biased data under the Cox model, which is the most commonly adopted semiparametric model, and introduce an R package **CoxPhLb** that implements these methods. Specifically, the package includes features such as fitting the Cox model to explore covariate effects on survival times and checking the proportional hazards model assumptions and the stationarity assumption. We illustrate usage of the package with a simulated data example and a real dataset, the Channing House data, which are publicly available.

Introduction

In prevalent cohort studies, subjects who have experienced an initiating event (e.g., disease diagnosis) but have not yet experienced a failure event (e.g., death) are sampled from the target population and followed until a failure or censoring event occurs. Data collected from such sampling designs are subject to left truncation since subjects who experienced a failure event prior to study enrollment are selectively excluded and are not observed in the data. When the occurrence of the initiating event follows a stationary Poisson process, the data are called “length-biased data”, which is a special case of left-truncated data. These data are encountered in a variety of fields such as cancer screening trials (Zelen and Feinleib, 1969), studies of unemployment (Lancaster, 1979; de Una-Alvarez et al., 2003), epidemiologic cohort studies (Gail and Benichou, 2000; Scheike and Keiding, 2006), and genome-wide linkage studies (Terwilliger et al., 1997). The failure times observed in such data tend to be longer than those in the target population since subjects with longer failure times are more likely to be included in the length-biased data. Figure 1 depicts the occurrence of length-biased sampling. The underlying length-biased sampling assumption (i.e., the stationarity assumption) can be analytically examined (Addona and Wolfson, 2006).

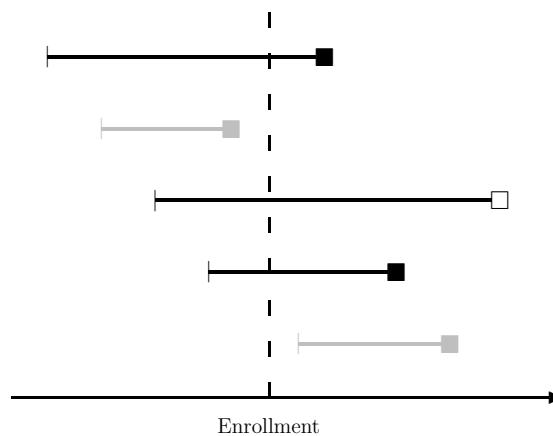


Figure 1: Illustration of length-biased sampling. Black horizontal lines represent observations that are included in the observed data cohort; gray horizontal lines represent observations excluded; black squares (■) represent uncensored failure events; and empty squares (□) represent censored failure events.

Provided that the observed data are not random samples of the target population, statistical methods for conventional survival data cannot be directly applied to length-biased data. Extensive studies have been conducted on statistical methodologies that account for length bias. In particular, a number of semiparametric regression methods have been proposed in the literature to model the association between covariates and the survival outcome of interest. Among the semiparametric regression models, the Cox proportional hazards model (Cox, 1972) has been the most commonly

adopted. Under the Cox model, Wang (1996) proposed a pseudo-partial likelihood approach to assess the covariate effects. However, her estimation method is limited to length-biased data with no right censoring. Tsai (2009) generalized the method to handle potential right censoring. Qin and Shen (2010) constructed estimating equations based on risk sets that are adjusted for length-biased sampling through inverse weights. A thorough review of the existing nonparametric and semiparametric regression methods can be found in Shen et al. (2017).

Although there is a substantial amount of literature on statistical methods for length-biased data, publicly available computational tools for analyzing such data are limited. In this paper, we introduce a new package, **CoxPhLb** (Lee et al., 2019a), in R that provides tools to analyze length-biased data under the Cox model. The package includes functions that fit the Cox model using the estimation method proposed by Qin and Shen (2010), check the proportional hazards model assumptions based on methods developed by Lee et al. (2019b) and check the underlying stationarity assumption. **CoxPhLb** is available from the Comprehensive R Archive Network (CRAN) at <http://CRAN.R-project.org/package=CoxPhLb>. To the best of our knowledge, this is the first and only publicly available R package for analyzing length-biased data under the Cox model.

The remainder of this paper is organized as follows. The following section provides a brief review of the semiparametric estimation method under the Cox model to assess the covariate effects on the survival outcome. Then, we outline how the Cox proportional hazards model assumptions can be checked both graphically and analytically, and describe two approaches to test the stationarity of the underlying incidence process. We illustrate the R package **CoxPhLb** using a simulated data example and a real dataset, the Channing House data. Finally, we conclude this paper with summarizing remarks.

Fitting the Cox model

Notation and model

Let \tilde{T} , \tilde{A} , and \mathbf{Z} be the duration from an initiating event to failure, the duration from the initiating event to enrollment in the study, and the $p \times 1$ baseline covariate vector, respectively. Assume that the failure time \tilde{T} follows the Cox model,

$$\lambda(t | \mathbf{z}) = \lambda_0(t) \exp(\boldsymbol{\beta}_0^\top \mathbf{z}), \quad (1)$$

where $\boldsymbol{\beta}_0$ is a $p \times 1$ vector of unknown regression coefficients and $\lambda_0(t)$ is an unspecified baseline hazard function. Under length-biased sampling, we only observe failure times that satisfy $\tilde{A} < \tilde{T}$. We denote the length-biased failure time by $T = A + V$, where A is the observed truncation variable (i.e., backward recurrence time) and V is the duration from study enrollment to failure (i.e., residual survival time or forward recurrence time). Since V is subject to right censoring, the observed failure time is $Y = \min(T, A + C)$ and the censoring indicator is $\delta = I(T \leq A + C)$, where C is the duration from study enrollment to censoring (i.e., residual censoring). The data structure is illustrated in Figure 2. We assume that C is independent of A and V given \mathbf{Z} , and the distribution of C is independent of \mathbf{Z} .

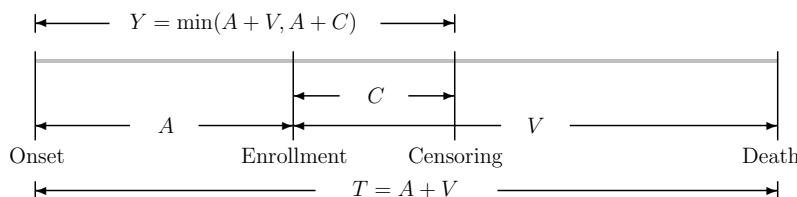


Figure 2: A diagram of the right-censored length-biased data.

The length-biased data consist of $\{(Y_i, A_i, \delta_i, \mathbf{Z}_i), i = 1, \dots, n\}$, for n independent subjects. We note that the observed data are not representative of the target population, and the observed biased data do not follow Model (1). Thus, conventional Cox regression methods cannot be used when evaluating the covariate effects on the duration from the initiating event to failure for the target population. Furthermore, even under the independent censoring assumption on C with A and V , the sampling mechanism induces dependent censoring because $\text{Cov}(T, A + C | \mathbf{Z}) = \text{Var}(A | \mathbf{Z}) + \text{Cov}(A, V | \mathbf{Z}) > 0$.

Estimation of the covariate effects

Among many estimation methods established for length-biased data under the Cox model, we provide the estimation function based on the inversely weighted estimating equation of Qin and Shen (2010). While this estimating equation approach may not be the most efficient method, the estimation procedure is easy to implement and provides a mean zero stochastic process that forms the basis of model checking. We adopt this estimation method for model fitting and checking in the R package **CoxPhLb**.

For subject i , we denote $N_i(t) = I\{Y_i \leq t, \delta_i = 1\}$ and $R_i(t) = I\{Y_i \geq t, \delta_i = 1\}$ following the counting process notation. Let $\mathbf{a}^0 = 1$, $\mathbf{a}^1 = \mathbf{a}$, and $\mathbf{a}^2 = \mathbf{a}\mathbf{a}^\top$ for any vector \mathbf{a} . Define

$$S^{(k)}(\boldsymbol{\beta}, t) = n^{-1} \sum_{i=1}^n w_C(t) R_i(t) \{w_C(Y_i)\}^{-1} \mathbf{Z}_i^k \exp(\boldsymbol{\beta}^\top \mathbf{Z}_i),$$

where the weight function $w_C(y) = \int_0^y S_C(u) du$, in which $S_C(y) = \Pr(C > y)$ is the survival function of the residual censoring variable C for $k = 0, 1$, and 2 . By replacing $w_C(y)$ with its consistent estimator, $\widehat{w}_C(y) = \int_0^y \widehat{S}_C(u) du$, where $\widehat{S}_C(\cdot)$ is the Kaplan–Meier estimator of the residual censoring survival function, we have

$$\widehat{S}^{(k)}(\boldsymbol{\beta}, t) = n^{-1} \sum_{i=1}^n \widehat{w}_C(t) R_i(t) \{\widehat{w}_C(Y_i)\}^{-1} \mathbf{Z}_i^k \exp(\boldsymbol{\beta}^\top \mathbf{Z}_i)$$

for $k = 0, 1$, and 2 . The regression parameter β_0 can be estimated by solving the following unbiased estimating equation:

$$\widehat{U}(\boldsymbol{\beta}) = \sum_{i=1}^n \int_0^\tau \left\{ \mathbf{Z}_i - \widehat{E}(\boldsymbol{\beta}, u) \right\} dN_i(u) = 0, \quad (2)$$

where τ satisfies $\Pr(Y \geq \tau) > 0$ and $\widehat{E}(\boldsymbol{\beta}, t) = \widehat{S}^{(1)}(\boldsymbol{\beta}, t)/\widehat{S}^{(0)}(\boldsymbol{\beta}, t)$. The solution to Equation (2), which is denoted by $\widehat{\boldsymbol{\beta}}$, is unique and a consistent estimator to β_0 . Using the Taylor series expansion, it can be shown that the distribution of $\widehat{\boldsymbol{\beta}}$ converges weakly to a normal distribution with variance $\Gamma^{-1}\Sigma\Gamma^{-1}$, where $\Gamma = -\lim_{n \rightarrow \infty} n^{-1}\partial\widehat{U}(\boldsymbol{\beta})/\partial\boldsymbol{\beta}$ and Σ is the covariance matrix of $\lim_{n \rightarrow \infty} n^{-1/2}\widehat{U}(\boldsymbol{\beta}_0)$.

This estimation procedure can be implemented using the **coxphlb** function in the **CoxPhLb** package as follows:

```
coxphlb(formula,data,method = c("Bootstrap","EE"),boot.iter = 500,
        seed.n = round(runif(1,1,1e09)),digits = 3L)
```

where **formula** has the same syntax as the formula used in **coxph** from the **survival** package (Therneau, 2020). The response needs to be a **survival** object such as **Surv(a,y,delta)** where **a**, **y**, and **delta** are the truncation times, the observed failure times, and the censoring indicators, respectively. The argument **data** is a data frame that includes variables named in the formula. We can choose either the bootstrap variance estimates ("Bootstrap"), or the model-based variance estimates ("EE") to be returned in the fitted model object through the argument **method**. When bootstrap resampling is chosen for variance estimation, the bootstrap sample size is controlled by **boot.iter** with the default set as 500, and a seed number can be fixed by **seed.n**. A summary table is returned with values rounded by the integer set through **digits**.

Alternatively, one can implement the estimation procedure by using the **coxph** function with the subset of the data that consist of uncensored failure times only and an **offset** term to add $\log\{\widehat{w}_C(Y_i)\}^{-1}$ to the linear predictor with a fixed coefficient of one as discussed in Qin and Shen (2010). The **coxph** function will return the same point estimates as the **coxphlb** function. To compute the corresponding standard errors using **coxph**, we need to use the bootstrap approach. Later, in the simulated data example, we further evaluate the computational efficiency of the **coxphlb** function with the model-based variance estimation (i.e., **method = EE**) opposed to the bootstrap resampling method (i.e., **method = Bootstrap**).

Checking the Cox model assumptions

Two primary components of checking the Cox proportional hazards model assumptions are examining (i) the functional form of a covariate and (ii) the proportional hazards assumption. To detect violations of these model assumptions, the general form of the cumulative sums of multiparametric stochastic processes is considered.

Under Model (1), we can construct a mean zero stochastic process,

$$M_i(t) = N_i(t) - \int_0^t w_C(u) R_i(u) \{w_C(Y_i)\}^{-1} \exp(\boldsymbol{\beta}_0^\top \mathbf{Z}_i) d\Lambda_0(u),$$

for $i = 1, \dots, n$, where $\Lambda_0(t) = \int_0^t \lambda_0(s) ds$ is the cumulative baseline hazard function. The stochastic process can be estimated by

$$\widehat{M}_i(t) = N_i(t) - \int_0^t \widehat{w}_C(u) R_i(u) \{\widehat{w}_C(Y_i)\}^{-1} \exp(\widehat{\boldsymbol{\beta}}^\top \mathbf{Z}_i) d\widehat{\Lambda}_0(\widehat{\boldsymbol{\beta}}, u),$$

where

$$\widehat{\Lambda}_0(\widehat{\boldsymbol{\beta}}, t) = \int_0^t \frac{\sum_{i=1}^n dN_i(u)}{n \widehat{S}^{(0)}(\widehat{\boldsymbol{\beta}}, u)}.$$

The stochastic process can be considered as the difference between the observed and the expected number of events, which mimics the ordinary martingale residuals. When the estimated processes $\widehat{M}_i(t)$, $i = 1, \dots, n$ deviate from zero systematically, it may be a sign of model misspecification.

Let

$$\mathbf{G}(t, \mathbf{z}) = \sum_{i=1}^n f(\mathbf{Z}_i) I(\mathbf{Z}_i \leq \mathbf{z}) \widehat{M}_i(t), \quad (3)$$

where $f(\cdot)$ is a prespecified smooth and bounded function, and $I(\mathbf{Z}_i \leq \mathbf{z}) = I(Z_{i1} \leq z_1, \dots, Z_{ip} \leq z_p)$ with $\mathbf{Z}_i = (Z_{i1}, \dots, Z_{ip})^\top$ and $\mathbf{z} = (z_1, \dots, z_p)^\top$. When the model assumptions are satisfied, the process (3) will fluctuate randomly around zero. We can adjust the general form (3) to examine the specific model assumptions. To assess the functional form of the j th covariate, we choose $f(\cdot) = 1$, $t = \tau$, and $z_k = \infty$ for all $k \neq j$. The proportional hazards assumption for the j th covariate can be evaluated by setting $f(Z_{ij}) = Z_{ij}$ and $\mathbf{z} = \infty$. To develop analytical test procedures, test statistics can be constructed using the supremum test, $\sup_{t, \mathbf{z}} |\mathbf{G}(t, \mathbf{z})|$. Let $T_1^j = \sup_z |\mathbf{G}_1^j(z)|$ be the test statistics for checking the functional form of the j th covariate, where $\mathbf{G}_1^j(z) = \sum_{i=1}^n I(Z_{ij} \leq z) \widehat{M}_i(\tau)$; and $T_2^j = \sup_t |\mathbf{G}_2^j(t)|$, where $\mathbf{G}_2^j(t) = \sum_{i=1}^n Z_{ij} \widehat{M}_i(t)$ for checking the proportional hazards assumption for the j th covariate. We can also consider the global test statistic $T_2 = \sup_t \sum_{j=1}^p |\mathbf{G}_2^j(t)|$ when the overall proportionality of hazards for all covariates is of interest.

The null distribution of the general form (3) under Model (1) has been studied in Lee et al. (2019b) to derive the critical values for test statistics T_1^j , T_2^j , and T_2 . We can approximate the null distribution by adopting the resampling technique used in Lin et al. (1993). Let

$$\begin{aligned} \widehat{\mathbf{G}}_i^*(t, \mathbf{z}) &= \int_0^t \left\{ f(\mathbf{Z}_i) I(\mathbf{Z}_i \leq \mathbf{z}) - \widehat{E}_Z(\widehat{\boldsymbol{\beta}}, u, \mathbf{z}) \right\} d\widehat{M}_i(u) + \int_0^t \widehat{H}(\widehat{\boldsymbol{\beta}}, u) \frac{d\widehat{M}_{C_i}(u)}{\widehat{\pi}(u)} \\ &\quad + \widehat{\Gamma}_Z(\widehat{\boldsymbol{\beta}}, t, \mathbf{z}) \{\widehat{\Gamma}(\widehat{\boldsymbol{\beta}})\}^{-1} \int_0^\tau \left\{ \mathbf{Z}_i - \widehat{E}(\widehat{\boldsymbol{\beta}}, u) \right\} d\widehat{M}_i(u), \end{aligned}$$

where

$$\widehat{S}_Z^{(l)}(\boldsymbol{\beta}, t, \mathbf{z}) = n^{-1} \sum_{i=1}^n f(\mathbf{Z}_i) I(\mathbf{Z}_i \leq \mathbf{z}) \widehat{w}_C(t) R_i(t) \{\widehat{w}_C(Y_i)\}^{-1} \mathbf{Z}_i^l \exp(\boldsymbol{\beta}^\top \mathbf{Z}_i)$$

for $l = 0, 1$, $\widehat{E}_Z(\boldsymbol{\beta}, u, \mathbf{z}) = \widehat{S}_Z^{(0)}(\boldsymbol{\beta}, t, \mathbf{z}) / \widehat{S}^{(0)}(\boldsymbol{\beta}, t)$,

$$\begin{aligned}\widehat{H}(\boldsymbol{\beta}, t) &= \sum_{i=1}^n \sum_{k=1}^n \frac{f(\mathbf{Z}_k) I(\mathbf{Z}_k \leq \mathbf{z}) \widehat{w}_C(Y_i) R_k(Y_i) \exp(\boldsymbol{\beta}^\top \mathbf{Z}_k) \{\widehat{w}_C(Y_k)\}^{-2} \widehat{h}_k(t)}{n^2 \widehat{S}^{(0)}(\boldsymbol{\beta}, Y_i)}, \\ \widehat{M}_{C_i}(t) &= I(V_i \leq t, \delta_i = 0) - \int_0^t I(V_i \geq u) d\widehat{\Lambda}_C(u), \\ \widehat{h}_k(t) &= I(Y_k \geq t) \int_t^{Y_k} \widehat{S}_C(u) du, \\ \widehat{\pi}(t) &= \widehat{S}_C(t) \widehat{S}_V(t),\end{aligned}$$

in which $\widehat{\Lambda}_C(\cdot)$ is the Nelson-Aalen estimator for the residual censoring time and $\widehat{S}_V(\cdot)$ is the Kaplan-Meier estimator of the residual survival time, and

$$\begin{aligned}\widehat{\Gamma}_Z(\boldsymbol{\beta}, t, \mathbf{z}) &= n^{-1} \sum_{i=1}^n \int_0^t \left[\frac{\widehat{S}_{\mathbf{Z}}^{(1)}(\boldsymbol{\beta}, u, \mathbf{z})}{\widehat{S}^{(0)}(\boldsymbol{\beta}, u)} - \frac{\widehat{S}_{\mathbf{Z}}^{(0)}(\boldsymbol{\beta}, u, \mathbf{z}) \widehat{S}^{(1)}(\boldsymbol{\beta}, u)}{\{\widehat{S}^{(0)}(\boldsymbol{\beta}, u)\}^2} \right] dN_i(u), \\ \widehat{\Gamma}(\boldsymbol{\beta}) &= -n^{-1} \sum_{i=1}^n \int_0^\tau \left[\frac{\widehat{S}^{(2)}(\boldsymbol{\beta}, u)}{\widehat{S}^{(0)}(\boldsymbol{\beta}, u)} - \left\{ \frac{\widehat{S}^{(1)}(\boldsymbol{\beta}, u)}{\widehat{S}^{(0)}(\boldsymbol{\beta}, u)} \right\}^2 \right] dN_i(u).\end{aligned}$$

Define $\widetilde{\mathbf{G}}_m(t, \mathbf{z}) = \sum_{i=1}^n \widetilde{\mathbf{G}}_i^*(t, \mathbf{z}) V_{mi}$, where $V_{mi}, i = 1, \dots, n$, are independent random variables sampled from a standard normal distribution for $m = 1, \dots, M$. The simulated realizations of $\widetilde{\mathbf{G}}_m(t, \mathbf{z})$ for a large M approximate the null distribution. For graphical assessment, we can plot a few randomly chosen realizations of $\widetilde{\mathbf{G}}_m(t, \mathbf{z})$ and compare the observed process based on the data with them. A departure of the observed process from the simulated realizations implies violations of model assumptions. For an analytical test, the critical values for the test statistics can be derived by simulating $\sup_{t,z} |\widetilde{\mathbf{G}}_m(t, \mathbf{z})|$ for $m = 1, \dots, M$. We can compute the p values by the proportion of critical values greater than the test statistic.

We can carry out model checking in R using the function `coxphlb.ftest` to examine the functional form of a continuous covariate as follows:

```
coxphlb.ftest(fit, data, spec.p = 1, n.sim = 1000, z0 = NULL, seed.n = round(runif(1, 1, 1e09)),
               digits = 3L)
```

where the argument `fit` is an object of the "coxphlb" class, which can be obtained by using the `coxphlb` function, and `data` is the data frame used in the fitted model. We specify the j th component of the covariates to be examined via `spec.p`, where the default is set as $j = 1$. To approximate the null distribution, we sample a large number of realizations. The argument `n.sim` controls the number of samples with the default set as 1000. When specific grid points over the support of the j th covariate are to be examined, we can plug them in as a vector in `z0`, which if `NULL`, 100 equally distributed grid points will be selected over the range of the j th covariate by default. The random seed number can be fixed through `seed.n`. The p value returned by the function is rounded by the integer set via `digits`. To test the proportional hazards assumption, we use `coxphlb.phtest` as follows:

```
coxphlb.phtest(fit, data, spec.p = NULL, n.sim = 1000, seed.n = round(runif(1, 1, 1e09)),
                digits = 3L)
```

where all arguments play the same role as in the `coxphlb.ftest` function, except for `spec.p`. The proportional hazards assumption can be tested for the j th covariate if we set `spec.p` equal to j . The function will conduct the global test by default if `spec.p` is left unspecified.

We can conduct graphical assessment by using the following functions:

```
coxphlb.ftest.plot(x, n.plot = 20, seed.n = round(runif(1, 1, 1e09)))
coxphlb.phtest.plot(x, n.plot = 20, seed.n = round(runif(1, 1, 1e09)))
```

where `x` are objects of the "coxphlb.ftest" class and the "coxphlb.phtest" class, respectively. These functions return a plot of the observed process along with a randomly sampled `n.plot` number

of realizations. We can fix the random seed number through `seed.n`.

Checking the stationarity assumption

When the underlying incidence process follows a stationary Poisson process, the distribution of the truncation variable is uniform and the data are considered length-biased. In the literature, two approaches have been proposed to check the stationarity assumption: a graphical assessment and an analytical test procedure. [Asgharian et al. \(2006\)](#) demonstrated that the stationarity assumption can be checked graphically by comparing the Kaplan–Meier estimators based on the current and residual survival times. A large discrepancy indicates that the stationarity assumption is invalid. [Addona and Wolfson \(2006\)](#) proposed an analytic test to check the assumption. They showed that testing the stationarity assumption is equivalent to testing whether the distributions of the backward and forward recurrence times are the same. Let $F(t) = \Pr(A \leq t)$ and $G(t) = \Pr(V^* \leq t, \delta = 1)$, where $V^* = \min(V, C)$. Following [Wei \(1980\)](#), the test statistic can be constructed as follows:

$$W = n^{-2} \sum_{i=1}^n \sum_{j=1}^n \{ \Phi(A_i, V_j^*, \delta_j) - p \},$$

where $\Phi(A_i, V_j^*, \delta_j) = I(A_i > V_j^*, \delta_j = 1) - I(A_i < V_j^*)$ and $p = \mathbb{E}\{G(A_i)\} - \mathbb{E}\{F(V_j^*)\}$. The limiting distribution of the test statistic W has been studied in [Addona and Wolfson \(2006\)](#), and the corresponding p value can be computed.

In R, we can explore the stationarity assumption graphically using function `station.test.plot` as follows:

```
station.test.plot(a,v,delta)
```

where `a` is the vector of backward recurrence times, `v` is the vector of forward recurrence times, and `delta` is the vector of censoring indicators. The function produces a plot of two Kaplan–Meier curves. To test the assumption analytically, we can use

```
station.test(a,v,delta,digits = 3L)
```

where the data input arguments are the same as those in the `station.test.plot` function. The test statistic and the corresponding p value based on the two-sided test will be returned with the values rounded by the integer set by `digits`.

Implementation of CoxPhLb

The three major components of **CoxPhLb** are (i) model fitting using function `coxphlb`, (ii) model checking using functions `coxphlb.ftest` and `coxphlb.phtest`, and (iii) stationarity assumption testing using function `station.test`. An overview of all functions in the **CoxPhLb** package is presented in Table 1. In the following sections, we provide R codes that illustrate how to use the functions with the simulated data that are available in the **CoxPhLb** package and a real dataset, the Channing House data, which is publicly available in the **KMsurv** package ([Klein et al., 2012](#)). The provided R codes can be implemented after installing and loading the **CoxPhLb** package, which will automatically load the **survival** package.

The simulated data example

We use the simulated dataset, `ExampleData1`, that is available in the **CoxPhLb** package for illustration. The data have 200 observations and consist of length-biased failure times (`y`), the truncation variable (`a`), the censoring indicator (`delta`), and two covariates, X_1 with binary values (`x1`) and X_2 with continuous values that range from 0 to 1 (`x2`). The vector of forward recurrence times (`v`) is the difference between the failure times and the backward recurrence times (`y-a`).

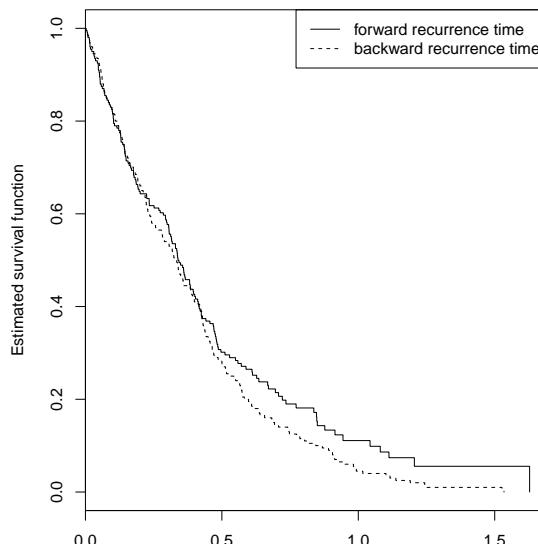
We begin by checking the stationarity assumption of the simulated dataset graphically as follows.

```
> data("ExampleData1", package = "CoxPhLb")
> dat1 <- ExampleData1
> station.test.plot(a = dat1$a, v = dat1$y - dat1$a, delta = dat1$delta)
```

Function	Description	S3 methods
<code>coxphlb</code>	Fits a Cox model to right-censored length-biased data	<code>print()</code> <code>summary()</code> <code>coef()</code> <code>vcov()</code>
<code>coxphlb.ftest</code>	Tests the functional form of covariates	<code>print()</code>
<code>coxphlb.phtest</code>	Tests the proportional hazards assumption	<code>print()</code>
<code>station.test</code>	Tests the stationarity assumption	<code>print()</code>
<code>coxphlb.ftest.plot</code>	Returns a graph for testing the functional form of covariates	
<code>coxphlb.phtest.plot</code>	Returns a graph for testing the proportional hazards assumption	
<code>station.test.plot</code>	Returns a graph for testing the stationarity assumption	

Table 1: Summary of functions in the **CoxPhLb** package.

The resulting plot is presented in Figure 3. We observe that the two Kaplan–Meier curves are very close to each other, especially at the early time points, which provides some evidence of the stationarity of incidence. However, we note some discrepancy in the tails of the curves. Thus, we conduct an analytical test to verify the underlying assumption as follows.

**Figure 3:** Testing the stationarity assumption for Example Data 1.

```
> station.test(a = dat1$a, v = dat1$y - dat1$a, delta = dat1$delta)
  test.statistic  p.value
  -0.375        0.707
```

To save the computed test statistic and the corresponding p value in the form of a list, we may assign the function outputs to an object. The analytical test provides a p value of 0.707, which indicates that the underlying stationarity of the incidence process is reasonable for the simulated dataset. Given that the data are subject to length bias, we evaluate the covariate effects on the failure time under the Cox model using the estimation method for length-biased data. First, we consider the model-based variance estimation.

```
> fit.ee1 <- coxphlb(Surv(a, y, delta) ~ x1 + x2, data = dat1, method = "EE")
  Call:
```

```
coxphlb(formula = Surv(a, y, delta) ~ x1 + x2, method = EE)
  coef  variance std.error z.score p.value lower.95 upper.95
x1 1.029 0.031    0.177   5.83    <0.001  0.683    1.375
x2 0.45  0.132    0.364   1.24    0.216   -0.263   1.163
```

The outputs include the estimated coefficients, the corresponding variance and the standard error estimates, the computed z scores and p values, and the 95% confidence intervals. In the above example R code, the list of outputs is saved by `fit.ee1` as an object of the "`coxphlb`" class. In the resulting table, we observe that the effect of X_1 is significant whereas that of X_2 is not. As an alternative approach for estimating the variance, we may use the bootstrap resampling method as follows.

```
> coxphlb(Surv(a, y, delta) ~ x1 + x2, data = dat1, method = "Bootstrap", seed.n = 1234)
Call:
coxphlb(formula = Surv(a, y, delta) ~ x1 + x2, method = Bootstrap)
      coef  variance std.error z.score p.value lower.95 upper.95
x1 1.029 0.032    0.178   5.78 <0.001   0.68    1.378
x2 0.45  0.133    0.365   1.23   0.218  -0.266   1.166
```

The outputs based on the bootstrap resampling method (i.e., `method = Bootstrap`) are close to the results from the model-based variance estimation (i.e., `method = EE`). To measure the average execution times of the two variance estimation methods, we ran the `coxphlb` function 100 times. The average execution times were 0.972s and 3.581s for `method = EE` and `Bootstrap`, respectively, on a desktop computer with Intel Core i5 CPU@3.40GHz and 8 GB 2400 MHz DDR4 of memory, which demonstrates the computational efficiency of `coxphlb` with the model-based variance estimation.

We note that the estimation results are only valid when the Cox proportional hazards model assumptions are correct. We thus verify the proportional hazards model assumptions. First, the linear functional form of the second covariate which has continuous values, can be checked as follows.

```
> ftest1 <- coxphlb.ftest(fit = fit.ee1, data = dat1, spec.p = 2, seed.n = 1234)
      p.value
x2 0.433
> coxphlb.ftest.plot(ftest1, n.plot = 50, seed.n = 1234)
```

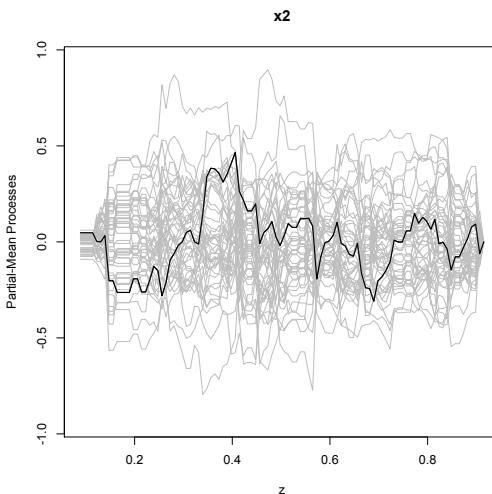


Figure 4: Checking the functional form of continuous covariate X_2 of Example Data 1.

To test the linear functional form, the object `fit.ee1` is specified as an input argument. We set `spec.p = 2` to conduct the test for X_2 and set `seed.n = 1234` for reproducible results. The `coxphlb.ftest` function returns a p value from the analytical test. For graphical assessment, we specify the object `ftest1` which is in the "`coxphlb.ftest`" class and set `n.plot = 50` to plot 50 realization lines. Based on Figure 4 and the p value, the functional form of X_2 satisfies the model assumption. Another important assumption of the Cox model is the proportional hazards assumption. We first test the assumption for each covariate and then conduct the global test for the overall proportionality.

```
> phtest11 <- coxphlb.phtest(fit = fit.ee1, data = dat1, spec.p = 1, seed.n = 1234)
      p.value
x1 0.59
> phtest12 <- coxphlb.phtest(fit = fit.ee1, data = dat1, spec.p = 2, seed.n = 1234)
      p.value
x2 0.833
> coxphlb.phtest.plot(phtest11, n.plot = 50, seed.n = 1234)
> coxphlb.phtest.plot(phtest12, n.plot = 50, seed.n = 1234)
```

The outputs consist of the p values derived from the analytical tests of checking the proportional hazards assumption and a plot of the stochastic processes. Figure 5 shows that the proportional

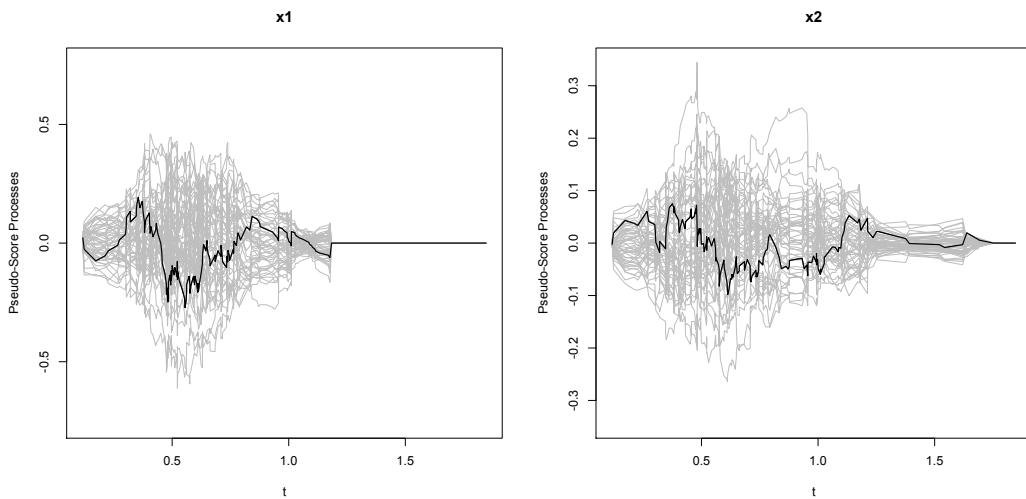


Figure 5: Checking the proportional hazards assumption for covariates X_1 and X_2 of Example Data 1.

hazards assumption is reasonable for both covariates. This is further confirmed by the computed p values, which are greater than 0.05.

```
> coxphlb.phtest(fit = fit.ee1, data = dat1, spec.p = NULL, seed.n = 1234)
      p.value
GLOBAL 0.762
```

The global test can be conducted by specifying `spec.p = NULL`. The result is consistent with the tests performed for each covariate. Note that the graphical assessment is unavailable for the global test.

The Channing House data

The Channing House data were collected from 462 individuals at a retirement center located in Palo Alto, California, from 1964 to 1975 (Klein and Moeschberger, 2003). We consider the elders aged 65 years or older as the target group of interest. Hence, we analyze the subset of the dataset composed of 450 individuals who entered the center at age 65 or older, in which 95 are males and 355 are females. The data include information of death indicator (`death`), age at entry (`ageentry`), age at death or censoring (`age`), and gender (`gender`). The observed survival times are left-truncated because only individuals who have lived long enough to enter the retirement center will be included in the data. We load the original dataset and select the subset of the dataset for illustration. Note that we convert age measured in months to years.

```
> install.packages("KMsurv")
> data("channing", package = "KMsurv")
> dat2 <- as.data.frame(cbind(ageentry = channing$ageentry/12, age = channing$age/12,
+                               death = channing$death, gender = channing$gender))
> dat2 <- dat2[dat2$ageentry >= 65, ]
```

First, we check the stationarity assumption as follows.

```
> station.test.plot(a = (dat2$ageentry - 65), v = (dat2$age - dat2$ageentry),
+                     delta = dat2$death)
```

The resulting plot in Figure 6 provides a strong sign that the stationarity assumption is satisfied. We further verify the assumption by conducting the analytical test.

```
> station.test(a = (dat2$ageentry - 65), v = (dat2$age - dat2$ageentry),
+               delta = dat2$death)
test.statistic p.value
0.261          0.794
```

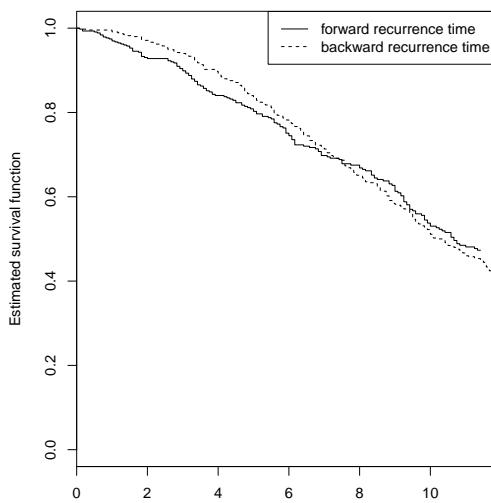


Figure 6: Testing the stationarity assumption for Channing House data.

Based on the results, we can conclude that the stationarity assumption is valid. Hence, we use the functions in **CoxPhLb** to assess the covariate effects on the survival outcome for the Channing House data. The model can be fitted as follows.

```
> fit.ee2 <- coxphlb(Surv((ageentry - 65), (age - 65), death) ~ gender, data = dat2,
+                      method = "EE")
Call:
coxphlb(formula = Surv((ageentry - 65), (age - 65), death) ~ gender, method = EE)
      coef  variance std.error z.score p.value lower.95 upper.95
gender -0.112 0.029     0.17    -0.66   0.509   -0.445   0.22
```

Using the model-based variance estimation, we find that gender is not a significant factor for survival. The bootstrap resampling approach provides consistent results as follows.

```
> coxphlb(Surv((ageentry - 65), (age - 65), death) ~ gender, data = dat2,
+           method = "Bootstrap", seed.n = 1234)
Call:
coxphlb(formula = Surv((ageentry - 65), (age - 65), death) ~ gender, method = Bootstrap)
      coef  variance std.error z.score p.value lower.95 upper.95
gender -0.112 0.028     0.168    -0.67   0.504   -0.441   0.217
```

The estimation of the covariate effects is only valid when the Cox model assumptions are not violated. By conducting the following proportional hazards assumption test, we verify that the assumption is reasonable based on the computed p value and Figure 7.

```
> phtest2 <- coxphlb.phtest(fit = fit.ee2, data = dat2, spec.p = 1, seed.n = 1234)
      p.value
gender 0.723
> coxphlb.phtest.plot(phtest2, seed.n = 1234)
```

Summary

Observational data subject to length-biased sampling have been widely recognized by epidemiologists, clinicians, and health service researchers. While statistical methodologies have been well established for analyzing such types of failure time data, the lack of readily available software has been a barrier to the implementation of proper methods. We introduce the R package **CoxPhLb** that allows practitioners to easily and properly analyze length-biased data under the Cox model, which is commonly used for conventional survival data. When the stationarity assumption is uncertain for the data, one can check the assumption graphically and analytically using tools provided in

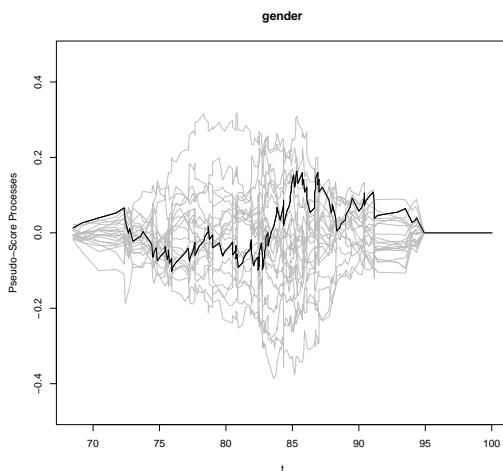


Figure 7: Checking the proportional hazards assumption for gender in the Channing House data.

the package prior to fitting the Cox model. In addition, the fundamental assumptions of the Cox model can be examined.

The **CoxPhLb** package may be further expanded by including other estimation approaches under the Cox model. For example, Qin et al. (2011) proposed an estimation method based on the full likelihood, which involves more intensive computations. Implementation of this method yields more efficient estimators, which is certainly desirable. In addition, when a violation of the proportional hazards assumption is detected by the `coxphlb.phtest` or `coxphlb.phtest.plot` functions, we may consider extending the regression method to handle covariates with non-proportionality such as the **coxphw** package (Dunkler et al., 2018) which implements the weighted Cox regression method for conventional survival data. We leave these possible extensions for future work.

Acknowledgements

This work was partially supported by the U.S. National Institutes of Health, grants CA193878 and CA016672.

Bibliography

- V. Addona and D. B. Wolfson. A formal test for the stationarity of the incidence rate using data from a prevalent cohort study with follow-up. *Lifetime Data Analysis*, 12:267–284, 2006. URL <https://doi.org/10.1007/s10985-006-9012-2>. [p]
- M. Asgharian, D. B. Wolfson, and X. Zhang. Checking stationarity of the incidence rate using prevalent cohort survival data. *Statistics in Medicine*, 25:1751–1767, 2006. URL <https://doi.org/10.1002/sim.2326>. [p]
- D. R. Cox. Regression models and life-tables (with discussion). *Journal of Royal Statistical Society B*, 34:187–220, 1972. URL <https://doi.org/10.1111/j.2517-6161.1972.tb00899.x>. [p]
- J. de Una-Alvarez, M. S. Otero-Giraldez, and G. Alvarez-Llorente. Estimation under length-bias and right-censoring: an application to unemployment duration analysis for married women. *Journal of Applied Statistics*, 30:283–291, 2003. URL <https://doi.org/10.1080/0266476022000030066>. [p]
- D. Dunkler, M. Ploner, M. Schemper, and G. Heinze. Weighted cox regression using the r package **coxphw**. *Journal of Statistical Software*, 84:1–26, 2018. URL <https://doi.org/10.18637/jss.v084.i02>. [p]
- M. H. Gail and J. Benichou. *Encyclopedia of Epidemiologic Methods*. Chichester: Wiley, 2000. [p]

- J. P. Klein and M. L. Moeschberger. *Survival analysis: techniques for censored and truncated data*. Springer-Verlag New York, Inc., 2nd edition, 2003. URL <https://doi.org/10.1007/978-1-4757-2728-9>. [p]
- J. P. Klein, M. L. Moeschberger, and J. modifications by Yan. *KMsurv: Data sets from Klein and Moeschberger (1997), Survival Analysis*, 2012. URL <https://CRAN.R-project.org/package=KMsurv>. R package version 0.1-5. [p]
- T. Lancaster. Econometric methods for the duration of unemployment. *Econometrica*, 47:939–956, 1979. URL <https://doi.org/10.2307/1914140>. [p]
- C. H. Lee, D. D. Liu, J. Ning, H. Zhou, and Y. Shen. *CoxPhLb: Analyzing Right-Censored Length-Biased Data*, 2019a. URL <https://CRAN.R-project.org/package=CoxPhLb>. R package version 1.2.0. [p]
- C. H. Lee, J. Ning, and Y. Shen. Model diagnostics for the proportional hazards model with length-biased data. *Lifetime Data Analysis*, 25:79–96, 2019b. URL <https://doi.org/10.1007/s10985-018-9422-y>. [p]
- D. Y. Lin, L. J. Wei, and Z. L. Ying. Checking the cox model with cumulative sums of martingale-based residuals. *Biometrika*, 80:557–572, 1993. URL <https://doi.org/10.2307/2337177>. [p]
- J. Qin and Y. Shen. Statistical methods for analyzing right-censored length-biased data under cox model. *Biometrics*, 66:382–392, 2010. URL <https://doi.org/10.1111/j.1541-0420.2009.01287.x>. [p]
- J. Qin, J. Ning, H. Liu, and Y. Shen. Maximum likelihood estimations and em algorithms with length-biased data. *Journal of American Statistical Association*, 106:1434–1449, 2011. URL <https://doi.org/10.1198/jasa.2011.tm10156>. [p]
- T. H. Scheike and N. Keiding. Design and analysis of time-to-pregnancy. *Statistical Methods in Medical Research*, 15:127–140, 2006. URL <https://doi.org/10.1191/0962280206sm435oa>. [p]
- Y. Shen, J. Ning, and J. Qin. Nonparametric and semiparametric regression estimation for length-biased survival data. *Lifetime Data Analysis*, 23:3–24, 2017. URL <https://doi.org/10.1007/s10985-016-9367-y>. [p]
- J. D. Terwilliger, W. D. Shannon, G. M. Lathrop, J. P. Nolan, L. R. Goldin, G. A. Chase, and D. E. Weeks. True and false positive peaks in genomewide scans: applications of length-biased sampling to linkage mapping. *American Journal of Human Genetics*, 61:430–438, 1997. URL <https://doi.org/10.1086/514855>. [p]
- T. M. Therneau. *A Package for Survival Analysis in R*, 2020. URL <https://CRAN.R-project.org/package=survival>. R package version 3.1-12. [p]
- W. Y. Tsai. Pseudo-partial likelihood for proportional hazards models with biased-sampling data. *Biometrika*, 96:601–615, 2009. URL <https://doi.org/10.1093/biomet/asp026>. [p]
- M. C. Wang. Hazards regression analysis for length-biased data. *Biometrika*, 83:343–354, 1996. URL <https://doi.org/10.1093/biomet/83.2.343>. [p]
- L. J. Wei. A generalized gehan and gilbert test for paired observations that are subject to arbitrary right censorship. *Journal of American Statistical Association*, 75:634–637, 1980. URL <https://doi.org/10.1080/01621459.1980.10477524>. [p]
- M. Zelen and M. Feinleib. On the theory of screening for chronic diseases. *Biometrika*, 56:601–614, 1969. URL <https://doi.org/10.1093/biomet/56.3.601>. [p]

Chi Hyun Lee

Department of Biostatistics and Epidemiology

University of Massachusetts Amherst

(<https://orcid.org/0000-0001-6340-2718>)

chihyunlee@umass.edu

Heng Zhou

Biostatistics and Research Decision Sciences

Merck & Co., Inc.

hengzhou89@gmail.com

Jing Ning
Department of Biostatistics
The University of Texas MD Anderson Cancer Center
jning@mdanderson.org

Diane D. Liu
Department of Biostatistics
The University of Texas MD Anderson Cancer Center
dianeliu@mdanderson.org

Yu Shen
Department of Biostatistics
The University of Texas MD Anderson Cancer Center
yshen@mdanderson.org

npordtests: An R Package of Nonparametric Tests for Equality of Location Against Ordered Alternatives

by Bulent Altunkaynak and Hamza Gamgam

Abstract Ordered alternatives are an important statistical problem in many situations such as increased risk of congenital malformation caused by excessive alcohol consumption during pregnancy life test experiments, drug-screening studies, dose-finding studies, the dose-response studies, age-related response. There are numerous other examples of this nature. In this paper, we present the **npordtests** package to test the equality of locations for ordered alternatives. The package includes the Jonckheere-Terpstra, Beier and Buning's Adaptive, Modified Jonckheere-Terpstra, Terpstra-Magel, Ferdiana-Terpstra-Magel, KTP, S and Gaur's Gc tests. A simulation study is conducted to determine which test is the most appropriate test for which scenario and to suggest it to the researchers.

Introduction

Ordered alternative tests are employed to evaluate if a quantitative feature is linked to an ordinal trait, as in the association between ammonia levels and the severity of hepatic encephalopathy (Ong et al., 2003), the association of abnormal MRI findings with bone-marrow-related disease (Bredella et al., 2006), and the association between single nucleotide polymorphisms in human genes and quantitative phenotypes (Hoffmeyer et al., 2000; Cheng et al., 2005; Kawaguchi et al., 2012; Uchiyama et al., 2012; Tan et al., 2014; Yorifuji et al., 2018)

There are parametric and nonparametric methods to test ordered alternatives. Nevertheless, the statistical validity of parametric methods depends upon distributional assumptions, such as normality or equality of variances. However, nonparametric tests do not necessitate assumptions about the distribution of the data and are robust to outliers and influential values (Lin et al., 2017b).

Several nonparametric tests were developed to test the equality of locations against ordered alternatives. These tests can be grouped under three headings such as linear combination of two sample statistics, linear rank statistics, and statistics based on k-tuplet.

The tests proposed by Terpstra (1952), Jonckheere (1954), Puri (1965), Govindarajulu and Haller (1971), Tryon and Hettmansperger (1973), Cuzick (1985), Le (1988), Neuhauser et al. (1998), Gaur (2014), Shan et al. (2014), Gaur (2017) are based on a linear combination of two sample statistics with pairs of samples of $k(k - 1)/2$. The problem of testing homogeneity against ordered alternatives was considered for the first time by Terpstra (1952) and Jonckheere (1954). They suggested the nonparametric test (JT) based on a sum of $k(k - 1)/2$ Mann-Whitney (MW) statistics for the ordered alternatives.

Linear rank statistics consist of a combination of the rank scores obtained from the combined data and the regression constants. These statistics were originally named as the Left Skewed (LS) and Right Skewed (RS) scores as proposed by Hogg et al. (1975). Gastwirth (1965), Buning and Kossler (1996), and Beier and Buning (1997) proposed Short-Tailed (ST), Long-Tailed (LT), and Wilcoxon (WS) scores, respectively. Beier and Buning (1997) proposed a nonparametric Adaptive Test (AT) for the choice of suitable scores based on the underlying distribution.

The k-tuplet tests are based on the information simultaneously obtained across all samples. These tests are determined by adding $N^* = n_1 \times n_2 \times \dots \times n_k$ functions. That is, k-tuplet includes one observation from each group. Terpstra and Magel (2003) proposed a test k-tuplet statistic (TM), which is based on the indicator function. Ferdiana et al. (2008) proposed a test statistic (FTM), which can be viewed as a generalization of the TM test. The FTM test uses Kendall correlation coefficient based on the following data: $(1, X_{1i_1}), (2, X_{2i_2}), \dots, (k, X_{ki_k})$, where $X_{ij} i = 1, 2, \dots, k, j = 1, 2, \dots, n_i$ is the sample data. Here, k is the number of groups and n_i denotes the number of observations in the i th group. Similarly, Terpstra et al. (2011) proposed KTP test, which uses Spearman correlation coefficient instead of Kendall correlation coefficient.

JT is the classical and the most common ordered test. It is included in some packages such as **clinfun** (Venkatraman, 2018), **jtGWAS** (Lin et al., 2017a), **fastJT** (Lin et al., 2017b), **kSamples** (Scholz and Zhu, 2018), **StatCharrms** (Swintek et al., 2018), **PMCMRplus** (Pohlert, 2018). However, the other ordered alternative tests considered in this study are not included in any CRAN package

other than **npordtests**.

However, there may be more efficient tests than JT for different data scenarios; nonetheless, a perusal of literature does not yield a comprehensive simulation study in which ordered alternative tests are compared for various scenarios. The nonparametric ordered alternative tests have recently been adapted for such big data structures as gene data and machine learning (Lin et al., 2017b), which clearly indicates the significance such a simulation study has.

Our study contributes significantly to the related literature in two ways: 1) This study includes most of the ordered alternative tests in the literature, introduced as an R package, **npordtests** (Altunkaynak and Gamgam, 2019) including the JT, Modified JT, LS, RS, ST, LT, WS, AT, TM, FTM, KTP, S, and Gaur's Gc tests, and presents open source codes. The **npordtests** package is publicly available on the CRAN. 2) This study presents a comprehensive simulation study that compares ordered alternative tests in terms of power, which helps researchers choose the most appropriate test for a given scenario.

The organization of this paper is presented as follows. After the introduction, firstly, we give the theoretical information about the nonparametric tests for ordered alternatives included in this study. Secondly, we introduce the **npordtests** package and demonstrate the applicability of the package using two benchmark datasets. Thirdly, a simulation study is conducted to determine which test is the most appropriate test for which scenario and to give some advice to the researchers. The results of this simulation study and general comments are given in the final section.

Ordered alternative tests

Let $X_{i1}, X_{i2}, \dots, X_{in_i}$, $i = 1, \dots, k$ be random independent samples with size n_i from k populations with continuous cumulative distribution function $F_i(x) = F((x - \theta_i)/\sigma_i)$, where $-\infty < \theta_i < +\infty$ and $\sigma_i > 0$ are location and scale parameters, respectively. The null hypothesis to identify whether the populations have common continuous cumulative distribution function can be expressed as

$$H_0 : F_1(x) = F_2(x) = \dots = F_k(x) \quad \forall x. \quad (1)$$

A number of test statistics have been proposed to test the null hypothesis in (1) under certain assumptions and for different forms of H_1 . The ordered alternative states that the distributions are stochastically ordered, i.e.,

$$H_1 : F_1(x) \geq F_2(x) \geq \dots \geq F_k(x) \quad \exists x : F_1(x) > F_k(x). \quad (2)$$

Under H_1 , X_i tends to be smaller than X_{i+1} , $i = 1, 2, \dots, k-1$, since $F_i(x) \geq F_{i+1}(x)$ implies that $P(X_i \leq X_{i+1}) \geq 1/2$. For the special case of the location model, (2) is equivalent to (Terpstra et al., 2011)

$$H_1 : \theta_1 \leq \theta_2 \leq \dots \leq \theta_k \quad (\theta_1 < \theta_k). \quad (3)$$

Similarly, the ordered alternative hypothesis

$$H_1 : F_1(x) \leq F_2(x) \leq \dots \leq F_k(x) \quad \exists x : F_1(x) < F_k(x) \quad (4)$$

states that X_i tends to be larger than X_{i+1} , $i = 1, 2, \dots, k-1$, since $F_i(x) \leq F_{i+1}(x)$ implies that $P(X_i \geq X_{i+1}) \geq 1/2$ under H_1 given in (4). For the location model, (4) is equivalent to

$$H_1 : \theta_1 \geq \theta_2 \geq \dots \geq \theta_k \quad (\theta_1 > \theta_k). \quad (5)$$

Jonckheere-Terpstra test

This classic nonparametric test is typically used for ordered alternatives and was proposed by Terpstra (1952) and Jonckheere (1954). It is known that the Mann-Whitney statistic defines as

$$U_{ij} = \sum_{l=1}^{n_i} \sum_{m=1}^{n_j} I(X_{il} < X_{jm});$$

where n_i and n_j are the sample sizes for the i th and j th populations, respectively, and $I(\psi) = 1$ if ψ is true and 0 otherwise. The test statistic JT corresponds to the sum of the $k(k-1)/2$

Mann-Whitney statistics, i.e.,

$$JT = \sum_{i=1}^{k-1} \sum_{j=i+1}^k U_{ij}. \quad (6)$$

The statistic JT is approximately normally distributed under H_0 . The mean and variance of this statistic are

$$E(JT) = \frac{N^2 - \sum_{i=1}^k n_i^2}{4}$$

and

$$V(JT) = \frac{N^2(2N+3) - \sum_{i=1}^k n_i^2(2n_i+3)}{72},$$

where $N = n_1 + n_2 + \dots + n_k$.

Beier and Buning's Adaptive test

This test is a two-step method based on the selection of the weight coefficients of the linear rank statistics according to the shape of the distribution (Beier and Buning, 1997). A linear rank statistics has the following form:

$$L_N = \sum_{i=1}^k \sum_{j=1}^{n_i} c_N(i) a_N(R_{ij}) \quad (7)$$

where N is the combined sample size; $c_N(\cdot)$ are the regression constants; $a_N(\cdot)$ are the scores; R_{ij} is the rank of X_{ij} in the combined data. For an ordered alternative, the following proposal is made:

$$c_N(i) = i, i = 1, 2, \dots, k.$$

Under H_0 , the mean and variance of linear rank statistics are

$$E(L_N) = N\bar{c}_N\bar{a}_N,$$

and

$$V(L_N) = \frac{1}{N-1} \sum_{i=1}^k n_i (c_N(i) - \bar{c}_N)^2 \sum_{r=1}^N (a_N(r) - \bar{a}_N)^2$$

where

$$\bar{c}_N = \frac{1}{N} \sum_{i=1}^k n_i c_N(i)$$

and

$$\bar{a}_N = \frac{1}{N} \sum_{r=1}^N a_N(r).$$

The distribution of a linear rank statistic converges to a normal distribution with mean $E(L_N)$ and variance $V(L_N)$ (Hogg and Craig, 2013; Beier and Buning, 1997).

There are some suggestions for the score $a_N(\cdot)$ according to the shape of the distribution in the literature as follows

$$a_{LS}(r) = \begin{cases} 0 & \text{if } r \leq (N+1)/2 \\ r - (N+1)/2 & \text{if } r > (N+1)/2 \end{cases}$$

These scores are efficient for detecting shifts in distributions that are skewed to the left (Beier and Buning, 1997).

$$a_{ST}(r) = \begin{cases} r - (N+1)/4 & \text{if } r \leq (N+1)/4 \\ 0 & \text{if } (N+1)/4 < r < 3(N+1)/4 \\ r - 3(N+1)/4 & \text{if } r \geq 3(N+1)/4 \end{cases}$$

These scores are particularly good for detecting shifts in short-tailed distributions and were proposed by Gastwirth (1965).

$$a_{WS}(r) = r, \quad r = 1, 2, \dots, N$$

These scores are efficient for detecting shifts in symmetric distributions with medium to heavy tails (Beier and Buning, 1997).

$$a_{LT}(r) = \begin{cases} -((N/4) + 1) & \text{if } r < (N/4) + 1 \\ r - (N + 1)/2 & \text{if } (N/4) + 1 \leq r \leq 3(N + 1)/4 \\ (N/4) + 1 & \text{if } r > 3(N + 1)/4 \end{cases}$$

These scores are efficient for detecting shifts in long-tail distributions and were proposed by Buning and Kossler (1996).

$$a_{RS}(r) = \begin{cases} r - (N + 1)/2 & \text{if } r \leq (N + 1)/2 \\ 0 & \text{if } r > (N + 1)/2 \end{cases}$$

These scores are efficient for detecting shifts in distributions that are skewed to the right (Hogg et al., 1975).

The adaptive test proposed by Beier and Buning (1997) is denoted by the index of their scores. For example, the distribution-free test based on the scores $a_{ST}(.)$ of Gastwirth (1965), which is particularly good for detecting a shift in short-tailed distributions, is denoted by ST. Now, the adaptive test AT is defined by

$$AT = \begin{cases} LS & \text{if } 0 \leq \hat{S}_1 \leq 0.6, \hat{S}_2 \geq 1 \\ ST & \text{if } 0.6 < \hat{S}_1 \leq 2, 1 \leq \hat{S}_2 \leq 1.5 \\ WS & \text{if } 0.6 < \hat{S}_1 \leq 2, 1.5 < \hat{S}_2 \leq 1.5 \\ LT & \text{if } 0.6 < \hat{S}_1 \leq 2, \hat{S}_2 \geq 2 \\ RS & \text{if } \hat{S}_1 \geq 2, \hat{S}_2 \geq 1 \end{cases} \quad (8)$$

where x_p is the quantile value of the combined data, and the estimation values of the skewness and tailweight of the distribution are

$$\hat{S}_1 = \frac{x_{0.975} - x_{0.5}}{x_{0.5} - x_{0.025}}$$

and

$$\hat{S}_2 = \frac{x_{0.975} - x_{0.025}}{x_{0.875} - x_{0.125}}.$$

Since the adaptive statistic is a linear rank statistic, the distribution of each of these statistics converges to a normal distribution with mean $E(L_N)$ and variance $V(L_N)$.

Modified Jonckheere-Terpstra test

Tryon and Hettmansperger (1973) proposed the modified JT statistic to test H_0 against the ordered alternatives,

$$MJT = \sum_{i=1}^{k-1} \sum_{j=i+1}^k (j-i)U_{ij}, \quad (9)$$

where U_{ij} is the Mann-Whitney statistic computed for the samples from the i th and j th populations. Neuhäuser et al. (1998) suggested that this test be used in place of the JT tests because it often has larger powers.

This statistic has a normal distribution under H_0 , and its mean and variance are

$$E(U_{ij}) = \frac{1}{2}n_i n_j, \quad \forall i \neq j$$

$$V(U_{ij}) = \frac{1}{12}n_i n_j (n_i + n_j + 1), \quad \forall i \neq j$$

$$Cov(U_{ij}, U_{il}) = Cov(U_{ji}, U_{li}) = \frac{1}{12}n_i n_j n_l, \quad \text{if all } i, j, l \text{ are different}$$

$$Cov(U_{ij}, U_{li}) = Cov(U_{ji}, U_{il}) = -\frac{1}{12}n_i n_j n_l, \quad \text{if all } i, j, l \text{ are different}$$

$$Cov(U_{ij}, U_{lm}) = 0, \quad \text{if all } i, j, l, m \text{ are different}$$

Terpstra-Magel test

Terpstra and Magel (2003) proposed a test statistic that does not focus on pairwise information. Instead, they use the information present in the $N^* = n_1 \times n_2 \times \dots \times n_k$ k-tuplets, where a k-tuplet

includes one observation from each treatment group. More specifically, the Terpstra–Magel (TM) test is based on the following statistic:

$$TM = \sum_{i_1=1}^{n_1} \dots \sum_{i_k=1}^{n_k} I(X_{1i_1} \leq X_{2i_2} \leq \dots \leq X_{ki_k}) \quad (10)$$

where the indicator function is equal to one when $X_{1i_1} < X_{ki_k}$.

The statistic TM is approximately normally distributed under H_0 . The mean and variance of this statistic are

$$E(TM) = \frac{N^*}{k!}$$

and

$$V(TM) = N^* \left(\frac{1}{k!} \right) \left(1 - \frac{1}{k!} \right) + \sum_{i=1}^{k-1} v_i^2$$

where

$$v_i^2 = \sum_{1 \leq l_1 < \dots < l_i \leq k} N^* \left[\prod_{s=1}^k (n_s - 1)^{I(s \neq l_1) \dots I(s \neq l_i)} \right] \left[\frac{\binom{2(k-l_i)}{k-l_i}}{2k-i} \prod_{s=1}^i \binom{2(l_s - l_{s-1} - 1)}{l_s - l_{s-1} - 1} - \frac{1}{(k!)^2} \right]$$

where $l_0 = 0$.

Ferdhiana-Terpstra-Magel test

Ferdhiana et al. (2008) proposed FTM test statistic can be viewed as a generalization of the TM test.

$$FTM = \sum_{i_1=1}^{n_1} \dots \sum_{i_k=1}^{n_k} \tau(X_{1i_1}, X_{2i_2}, \dots, X_{ki_k}) \quad (11)$$

where $\tau(X_{1i_1}, X_{2i_2}, \dots, X_{ki_k})$ denotes the Kendall correlation coefficient based on $(1, X_{1i_1}), (2, X_{2i_2}), \dots, (k, X_{ki_k})$.

Under H_0 , the statistic FTM is approximately normally distributed with zero mean, and its variance is

$$\begin{aligned} V(FTM) = & \left[\frac{2N^*}{\sqrt{3}k(k-1)} \right]^2 \left[\sum_{r=1}^{k-1} \sum_{s=r+1}^k \frac{n_r + n_s + 1}{n_r n_s} + 2 \sum_{r=1}^{k-2} \frac{1}{n_r} \left(\binom{k}{2} + \frac{r^2 - (2k-1)r}{2} \right) \right. \\ & \left. - 2 \sum_{r=1}^{k-2} \sum_{s=r+1}^{k-1} \frac{k-s}{n_s} + 2 \sum_{r=1}^{k-2} \sum_{s=r+1}^{k-1} \sum_{t=s+1}^k \frac{1}{n_t} \right]. \end{aligned}$$

KTP test

Terpstra et al. (2011) proposed the k-tuplet Terpstra-Page (KTP) test based on the statistic

$$KTP = \sum_{i_1=1}^{n_1} \dots \sum_{i_k=1}^{n_k} r_s(X_{1i_1}, X_{2i_2}, \dots, X_{ki_k}) \quad (12)$$

where $r_s(X_{1i_1}, X_{2i_2}, \dots, X_{ki_k})$ denotes the Spearman rank correlation coefficient based on $(1, X_{1i_1}), (2, X_{2i_2}), \dots, (k, X_{ki_k})$.

Under H_0 , the statistic KTP is approximately normally distributed, and its mean and variance are

$$E(KTP) = 0$$

, and

$$V(KTP) = \frac{144(N^*)^2}{k^2(k^2-1)^2} S,$$

where

$$S = \sum_{i_1=1}^{k-1} \sum_{i_2=i_1+1}^k \left[\frac{(i_2 - i_1)^2(n_{i_1} + n_{i_2} + 1)}{12n_{i_1}n_{i_2}} \right] \\ + \sum_{i_1=1}^{k-2} \sum_{i_2=i_1+1}^{k-1} \sum_{i_3=i_2+1}^k \left[\frac{(i_2 - i_1)(i_3 - i_1)}{6n_{i_1}} + \frac{(i_3 - i_2)(i_1 - i_2)}{6n_{i_2}} + \frac{(i_1 - i_3)(i_2 - i_3)}{6n_{i_3}} \right]$$

In the KTP test, Spearman's rank correlation coefficient r_s is given by the following formula:

$$r_s = 1 - \frac{6 \sum_{i=1}^k d_i^2}{k(k^2 - 1)}$$

where d_i represents the difference between the rank given to the value of the variable for each item of the particular data with y_i . This formula is applied in cases when there are no tied observations. The formula to use when there are tied observations is:

$$r_s = \frac{\sum_{i=1}^k (y_i - \bar{y})(x_i - \bar{x})}{\sqrt{\sum_{i=1}^k (y_i - \bar{y})^2 \sum_{i=1}^k (x_i - \bar{x})^2}}$$

where $(y, x) = (1, X_{1i_1}), (2, X_{2i_2}), \dots, (k, X_{ki_k})$ and x_i is rank of X_i . Note that if all of x_i values is equal, then $\sum(x_i - \bar{x})^2$ is zero. This result is also similar for Kendall correlation coefficient. Therefore, FTM and KTP tests cannot be applied to this type data. See Lehmann's data used in the demonstration of the **npordtests** package.

S test

Shan et al. (2014) proposed the new rank-based nonparametric test by incorporating the actual differences as follows

$$S = \sum_{i=1}^{k-1} \sum_{j=i+1}^k D_{ij} \quad (13)$$

where

$$D_{ij} = \sum_{l=1}^{n_i} \sum_{m=1}^{n_j} Z_{ijlm}, \quad Z_{ijlm} = (R_{jm} - R_{il})I(X_{jm} > X_{il})$$

and $R_{il}(R_{jm})$ is the rank of observation $X_{il}(X_{jm})$ in the combined data.

Under H_0 , the statistic S has a normal distribution with the following mean and variance

$$E(S) = \frac{N+1}{6} \sum_{i=1}^{k-1} \sum_{j=i+1}^k n_i n_j$$

$$V(S) = \left(\frac{N^2 + N}{12} - \frac{(N+1)^2}{36} \right) \sum_{i=1}^{k-1} \sum_{j=i+1}^k n_i n_j \\ + 2 \left[\sum_{i=1}^{k-1} n_i \left(\sum_{j=i+1}^k \frac{n_j}{2} \right) + \sum_{i=2}^k n_i \left(\sum_{j=1}^{i-1} \frac{n_j}{2} \right) \right] CovA + 2 \left(\sum_{i=1}^{k-2} \sum_{j=i+1}^{k-1} \sum_{l=j+1}^k n_i n_j n_l \right) CovB$$

where $CovA = \frac{2N^2+N-1}{90}$, and $CovB = \frac{-7N^2-11N-4}{360}$.

Gaur's G_c test

Let $(w_1, w_2, \dots, w_{k-1})$ be suitably selected real positive constants. Gaur (2017) proposed the G_c statistic to test H_0 against the ordered alternatives,

$$G_c = \sum_{g=1}^{k-1} w_g V_{g,g+1} \quad (14)$$

where

$$V_{g,h} = \left[\binom{n_g}{c} \binom{n_h}{c} \right]^{-1} \sum_0 \phi_{gh}(X_{g\alpha_1}, \dots, X_{g\alpha_c}; X_{h\beta_1}, \dots, X_{h\beta_c})$$

for $g < h$; $h = 1, 2, \dots, k$; \sum_0 is the sum over all combinations $(\alpha_1, \dots, \alpha_c)$ of c integers selected from $(1, \dots, n_g)$ and over all combinations $(\beta_1, \dots, \beta_c)$ of c integers selected from $(1, \dots, n_h)$;

$$\phi_{gh}(X_{g\alpha_1}, \dots, X_{g\alpha_c}; X_{h\beta_1}, \dots, X_{h\beta_c}) = \begin{cases} 1 & \text{if } \max(X_{g\alpha_1}, \dots, X_{g\alpha_c}) \leq \min(X_{h\beta_1}, \dots, X_{h\beta_c}) \\ -1 & \text{if } \max(X_{h\beta_1}, \dots, X_{h\beta_c}) \leq \min(X_{g\alpha_1}, \dots, X_{g\alpha_c}) \\ 0 & \text{otherwise} \end{cases}.$$

The distribution of Gaur's statistic G_c converges to a normal distribution with zero mean under H_0 , and the variance of this statistic are obtained as follows

$$V(G_c) = \mathbf{w}^\top \sum \mathbf{w}$$

where $\mathbf{w}^\top = (w_1, w_2, \dots, w_{k-1})$ and $\sum = [\sigma_{gh}]$ is the variance-covariance matrix, such as:

$$\sigma_{gh} = \begin{cases} \left(\frac{(c-1)!c!}{(2c-1)!} \right)^2 \left(\frac{1}{\lambda_g} + \frac{1}{\lambda_{g+1}} \right) \delta_c & \text{for } g = h = 1, 2, \dots, k-1 \\ -\left(\frac{(c-1)!c!}{(2c-1)!} \right)^2 \frac{\delta_c}{\lambda_{g+1}} & \text{for } h = g+1; g = 1, 2, \dots, k-2 \\ -\left(\frac{(c-1)!c!}{(2c-1)!} \right)^2 \frac{\delta_c}{\lambda_g} & \text{for } h = g-1; g = 2, \dots, k-1 \\ 0 & \text{otherwise} \end{cases}$$

where

$$\delta_c = -1 + \frac{4}{4c-1} \sum_{i=c}^{2c-1} \sum_{j=c}^{2c-1} \binom{2c-1}{i} \binom{2c-1}{j} \binom{4c-2}{i+j}^{-1}.$$

It is recommended to use G_c tests for light-tailed and moderate-tailed distributions with $c = 2$, whereas for heavy-tailed and long-tailed distributions with large values of c . The optimum weights w_g 's in the G_c test are

$$w_g = \frac{g(k-g)}{2k}, \quad g = 1, 2, \dots, k-1.$$

Demonstration of the npordtests package

The **npordtests** package includes thirteen tests and six datasets for ordered alternatives. In this section, firstly, we introduce the datasets included in the package. Then, we demonstrate the usage of the package by using two of these datasets. All the examples in this section should run if you type them in exactly as printed, provided that you have the **npordtests** package not only installed but also loaded into your current search path. This is done by entering

```
R> library(npordtests)
```

at the command prompt.

Datasets

Jonckheere's data: jdata

This hypothetic data given by Jonckheere (1954) are used to test the hypothesis that the four samples have come from the same population against the alternative that the populations are such

that the values from the samples I, II, III, IV are in an expected order of increasing value.

Lehmann's data: lehmann

This dataset was used by [Lehmann \(1975\)](#) to assess if it is possible for a particular diagnostic test to be successfully interpreted without psychological training. This dataset later became one of the classical datasets used to investigate sequential alternatives ([Beier and Buning, 1997](#)). The data included 72 evaluators' (21 staff members, 23 trainees and 28 undergraduate psychology majors) assessment scores for the diagnostic test. If training and experience have any effects, the staff members could be expected to perform the most accurately, the trainees next, and the undergraduates the least.

Chicks' weight data: chicks

These data are given by [Desu and Raghavarao \(2004\)](#) to examine the hypothesis that the chicks' mean weight goes up with the increase in the amount of protein. Eighteen chicks were randomly assigned to three treatments with six chicks in each for balanced data. Treatment 1 had the diet with the lowest level of protein; treatment 2 had the diet with a medium level of protein; and treatment 3 had the highest level of protein. After six weeks of feeding, the values of weight gain were recorded. We wanted to test if the mean weight gain increased with the amount of protein ([Chang and Yen, 2011](#)).

Hepatic vein waveform index data: hvwi

These data were collected by [Pedersen et al. \(2008\)](#) through doppler waveforms corresponding to 66 patients scheduled for a percutaneous liver needle biopsy. The waveforms were characterized using a hepatic vein waveform index (HVWI), whereas the biopsy specimens were grouped according to the degree of fibrosis. The hypothesis of interest was that the HVWI values would tend to decrease as the degree of fibrosis increases ([Terpstra et al., 2011](#)).

Hypertension data: hypertension

These data presented by [Dmitrienko et al. \(2006\)](#) examine the effect of different drug doses on diastolic blood pressure. The patients with hypertension were randomized into four groups with different dose levels, 0, 10, 20, and 40 mg/day, where the group with 0 mg/day was the placebo group. The number of the patients in each group were 17, 17, 18, and 16, respectively. The complete data can be found at the [Dmitrienko et al. \(2006\)](#) or [Shan et al. \(2014\)](#).

Neuhäuser's data: neuhauser

These synthetic data are reported by [Neuhäuser et al. \(1998\)](#). The data consist of 4 groups with 10 observations in each.

In order to compare the distributions of groups for each dataset, the boxplots are given in Figure 1. As can be seen from the figure, there is a ordered alternative pattern in all datasets.

Tests

Using the datasets which are named **jdata** and **lehmann**, demonstration of the tests are given below, respectively.

Jonckheere-Terpstra test: JtTest(...)

The **JtTest** function in the **npordtests** package is used to perform the Jonckheere-Terpstra test.

```
R> data(jdata)
R> JtTest(Y~X,jdata,alpha=0.05,na.rm=TRUE,verbose=TRUE)
```

Test : Jonckheere-Terpstra Test

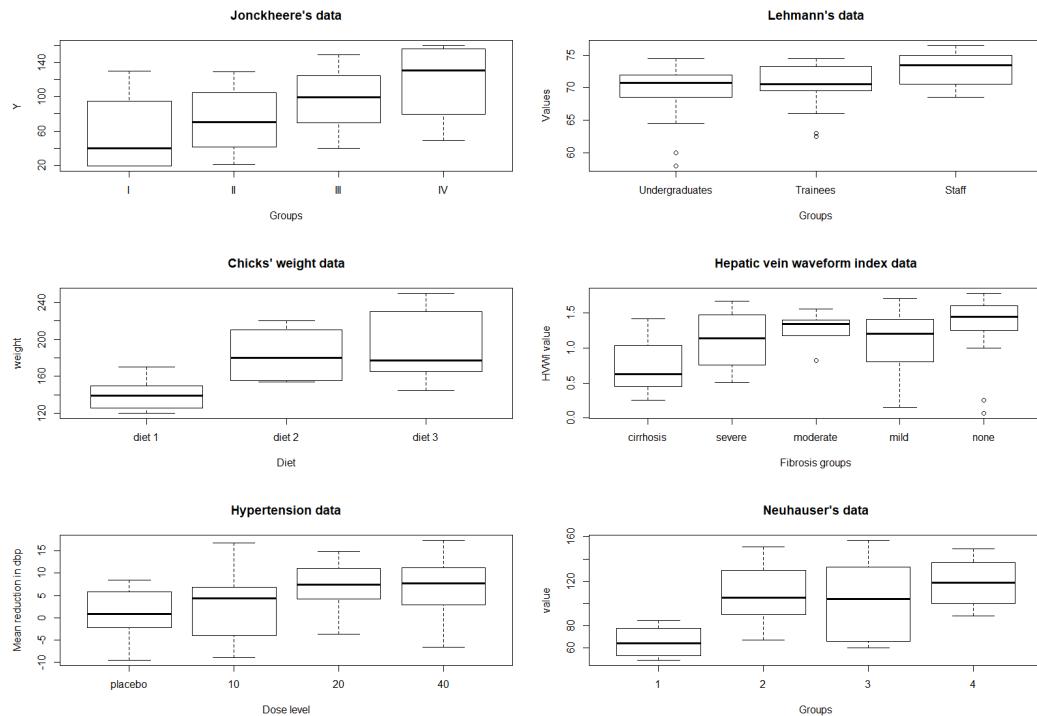


Figure 1: Boxplots for the datasets. Each box plot gives median (the bold line that divides the box into two parts), lower and upper quartiles (start and end points of the box on the vertical axis) and min and max value (the horizontal lines outside the box). The outliers appear as the circles.

```

data : Y and X

Statistic = 71
Mean = 48
Variance = 114.6667
Z = 2.147876
Asymp. p-value = 0.0158618

Result      : Null hypothesis is rejected.
-----
```

Here, the JT statistic is calculated from the Equation (6). Also, the Mean and Variance are expected value and variance of the JT statistic, respectively. Z is calculated from $(JT - E(JT))/\sqrt{V(JT)}$. p-value is the significance value for the JT test. Because this p-value is smaller than $\alpha = 0.05$, the hypothesis of the equality of locations against the ordered alternative is rejected.

alpha is the level of significance to assess the statistical difference. Default is set to alpha = 0.05. na.rm is a logical value indicating whether NA values should be stripped before the computation proceeds. Default is na.rm = TRUE. verbose is a logical for printing output to R console. Default is set to verbose = TRUE. These arguments are available in the functions for ordered alternatives. The users who would like to use the statistics in the output in their programs can use the following codes.

```

R> res<-JtTest(Y~X,jdata,alpha=0.05,na.rm=TRUE,verbose=FALSE)

R> res$statistic
[1] 71

R> res$mean
[1] 48

R> res$variance
```

```
[1] 114.6667

R> res$Z
[1] 2.147876

R> res$p.value
[1] 0.0158618
```

Here, the codes for how to obtain the statistics from the Jonckheere-Terpstra test output are given. Since all ordered alternative tests return similar outputs, similar codes are not repeated in the other tests. For all tests, the level of significance is taken as 0.05.

Beier and Buning's Adaptive test: AtTest(...)

The **AtTest** function in the **npordtests** package is used to perform the Adaptive test. The LS, RS, ST, WS and LT tests are also available as functions in the package.

```
R> LsTest(Y~X,jdata)

-----
Test : LS test
data : Y and X

Statistic = 68
Mean = 48
Variance = 141.3333
Z = 1.682316
Asymp. p-value = 0.04625375

Result      : Null hypothesis is rejected.
```

Here, the **Statistic** is calculated from the Equation (7) using the score $a_{LS}(r)$. Also, the **Mean** and **Variance** are the expected value and variance of the this statistic, respectively. Z is calculated from $(LS - E(LS)) / \sqrt{V(LS)}$. **p-value** is the significance value for the LS test. Since this **p-value** is smaller than $\alpha = 0.05$, the null hypothesis against the ordered alternative is rejected.

```
R> RsTest(Y~X,jdata)

-----
Test : RS test
data : Y and X

Statistic = -27
Mean = -48
Variance = 141.3333
Z = 1.766432
Asymp. p-value = 0.03866168

Result      : Null hypothesis is rejected.
```

In the output, similar to **LsTest**, the **Statistic** is calculated from the Equation (7) using the score $a_{RS}(r)$. Z is calculated from $(RS - E(RS)) / \sqrt{V(RS)}$. **p-value** is the significance value for the RS test. According to these results, because the **p-value** is smaller than $\alpha = 0.05$, the hypothesis of the equality of locations against the ordered alternative is rejected.

```
R> StTest(Y~X,jdata)

-----
Test : ST test
data : Y and X

Statistic = 17.25
```

```

Mean = 0
Variance = 46
Z = 2.543374
Asymp. p-value = 0.005489386

Result      : Null hypothesis is rejected.
-----
```

In the output, the **Statistic** is calculated from the Equation (7) using the score $a_{ST}(r)$. **Z** is calculated from $(ST - E(ST))/\sqrt{V(ST)}$. **p-value** is the significance value for the ST test. Here, the **Statistic** is calculated value of the test statistic. The **p-value** for the TM test is 0.005489386. Thus, we can conclude that the null hypothesis of the equality of locations is rejected under setting $\alpha = 0.05$.

```

R> WsTest(Y~X,jdata)

-----
Test : WS test
data : Y and X

Statistic = 245
Mean = 204
Variance = 453.3333
Z = 1.92564
Asymp. p-value = 0.02707469

Result      : Null hypothesis is rejected.
-----
```

Here, the **WS statistic** is calculated from the Equation (7) using the score $a_{WS}(r)$. **Z** is calculated from $(WS - E(WS))/\sqrt{V(WS)}$. **p-value** is the significance value for the WS test. Because this **p-value** is smaller than $\alpha = 0.05$, the hypothesis of the equality of locations against the ordered alternative is rejected.

```

R> LtTest(Y~X,jdata)

-----
Test : LT test
data : Y and X

Statistic = 27.5
Mean = 0
Variance = 322.6667
Z = 1.530931
Asymp. p-value = 0.06289321

Result      : Null hypothesis is not rejected.
-----
```

The **LT statistic** is calculated from the Equation (7) using the score $a_{LT}(r)$. **Z** is calculated from $(LT - E(LT))/\sqrt{V(LT)}$. **p-value** is the significance value for the LT test. According to these results, because the **p-value** is not smaller than $\alpha = 0.05$, the hypothesis of the equality of locations (null hypothesis) is not rejected.

```

R> AtTest(Y~X,jdata)

-----
Test : Adaptive Test
data : Y and X

Statistic = 17.25
Mean = 0
Variance = 46
Z = 2.543374
Asymp. p-value = 0.005489386
```

```
Result      : Null hypothesis is rejected.
```

Here, the **Statistic** is calculated from the Equation (8). Note that the AT **Statistic** is equal to the ST **Statistic** for this example. Since this **p-value** is smaller than $\alpha = 0.05$, the null hypothesis against the ordered alternative is rejected.

Modified Jonkheere-Terpstra test: MjtTest(...)

The **MjtTest** function in the **npordtests** package is used to perform the MJT test.

```
R> MjtTest(Y~X,jdata)
```

```
-----  
Test : Modified Jonckheere-Terpstra Test  
data : Y and X
```

```
Statistic = 121  
Mean = 80  
Variance = 453.3333  
Z = 1.92564  
Asymp. p-value = 0.02707469
```

```
Result      : Null hypothesis is rejected.
```

Here, the **Statistic** is calculated from the Equation (9). According to these results, because the **p-value** is smaller than $\alpha = 0.05$, the hypothesis of the equality of locations against the ordered alternative is rejected.

Terpstra-Magel test: TmTest(...)

The **TmTest** function in the **npordtests** package is used to perform the TM test.

```
R> TmTest(Y~X,jdata)
```

```
-----  
Test : Terpstra-Magel Test  
data : Values and Group
```

```
Statistic = 78  
Mean = 10.66667  
Variance = 151.327  
Z = 5.473586  
Asymp. p-value = 2.205097e-08
```

```
Result : Null hypothesis is rejected.
```

In the output, the **Statistic** is calculated from the Equation (10). Z is calculated from $(TM - E(TM)) / \sqrt{V(TM)}$. **p-value** is the significance value for the TM test. The **p-value** for the TM test is 0.00000002205097. Thus, we can conclude that the null hypothesis of the equality of locations is rejected under setting $\alpha = 0.05$.

Ferdhiana-Terpstra-Magel test: FtmTest(...)

The **FtmTest** function in the **npordtests** package is used to perform the FTM test.

```
R> FtmTest(Y~X,jdata)
```

```
-----  
Test : Ferdhiana, Terpstra and Magel Test
```

```

data : Y and X

Statistic = 122.6667
Mean = 0
Variance = 3261.63
Z = 2.147876
Asymp. p-value = 0.0158618

Result      : Null hypothesis is rejected.
-----
```

Here, the **Statistic** is calculated from the Equation (11). Z is calculated from $FTM / \sqrt{V(FTM)}$. **p-value** is the significance value for the FTM test. Because this **p-value** is smaller than $\alpha = 0.05$, the hypothesis of the equality of locations against the ordered alternative is rejected.

KTP test: KtpTest(...)

The **KtpTest** function in the **npordtests** package is used to perform the KTP test.

```

R> KtpTest(Y~X,jdata)

-----
Test : KTP Test
data : Y and X

Statistic = 131.2
Mean = 0
Variance = 4642.133
Z = 1.92564
Asymp. p-value = 0.02707469

Result      : Null hypothesis is rejected.
-----
```

Here, the **Statistic** is calculated from the Equation (12). Z is calculated from $KTP / \sqrt{V(KTP)}$. **p-value** is the significance value for the KTP test. Since this **p-value** is smaller than $\alpha = 0.05$, the null hypothesis against the ordered alternative is rejected.

S test: SsTest(...)

The **SsTest** function in the **npordtests** package is used to perform the S test.

```

R> SsTest(Y~X,jdata)

-----
Test : Shan's S test
data : Y and X

Statistic = 436
Mean = 272
Variance = 1973.511
Z = 3.69168
Asymp. p-value = 0.0001113888

Result      : Null hypothesis is rejected.
-----
```

In the output, the **Statistic** is calculated from the Equation (13). Z is calculated from $(S - E(S)) / \sqrt{V(S)}$. **p-value** is the significance value for the S test. According to these results, because the **p-value** is smaller than $\alpha = 0.05$, the hypothesis of the equality of locations against the ordered alternative is rejected.

Gaur's Gc test: GcTest(...)

The GcTest function in the npordtests package is used to perform the Gaur's Gc test.

```
R> GcTest(Y~X,jdata)
```

```
Test : Gaur's Gc Test
data : Values and Group
```

```
Statistic = 0.375
Mean = 0
Variance = 0.06746032
Z = 1.4438
Asymp. p-value = 0.0743976
```

```
Result : Null hypothesis is not rejected.
```

Here, the **Statistic** is calculated from the Equation (14). Z is calculated from $G_c / \sqrt{V(G_c)}$. **p-value** is the significance value for the G_c test. Here, the **Statistic** is calculated value of the test statistic. The **p-value** for the G_c test is 0.0743976. Thus, we can conclude that the null hypothesis of the equality of locations is not rejected under setting $\alpha = 0.05$.

Jonkheere-Terpstra test: JtTest(...)

The JtTest function in the npordtests package is used to perform the JT test.

```
R> data(lehmann)
R> JtTest(Values~Group,lehmann)
```

```
Test : Jonckheere-Terpstra Test
data : Values and Group
```

```
Statistic = 1159
Mean = 857.5
Variance = 9305.917
Z = 3.125415
Asymp. p-value = 0.0008877709
```

```
Result      : Null hypothesis is rejected.
```

Here, the **Statistic** is calculated value of the test statistic. **p-value** is the significance value for this test. The **p-value** for the JT test is 0.0008877709. Thus, we can conclude that the null hypothesis of the equality of locations is rejected under setting $\alpha = 0.05$.

Beier and Buning's Adaptive test: AtTest(...)

The AtTest function in the npordtests package is used to perform the AT test.

```
R> AtTest(Values~Group,lehmann)
```

```
Test : Adaptive Test
data : Values and Group
```

```
Statistic = 851
Mean = 583.1944
Variance = 6570.726
Z = 3.303794
Asymp. p-value = 0.0004769302
```

```
Result      : Null hypothesis is rejected.
```

Here, the **Statistic** is calculated value of the test statistic. **p-value** is the significance value for this test. The **p-value** for the AT test is 0.0004769302. Because this **p-value** is smaller than $\alpha = 0.05$, the hypothesis of the equality of locations against the ordered alternative is rejected.

Modified Jonkheere-Terpstra test: MjtTest(...)

The **MjtTest** function in the **npordtests** package is used to perform the MJT test.

```
R> MjtTest(Values~Group,lehmann)
```

```
-----  
Test : Modified Jonckheere-Terpstra Test  
data : Values and Group
```

```
Statistic = 1610  
Mean = 1151.5  
Variance = 20771.92  
Z = 3.181274  
Asymp. p-value = 0.0007331448
```

```
Result      : Null hypothesis is rejected.
```

Here, the **Statistic** is calculated value of the test statistic. **p-value** is the significance value for the MJT test. The **p-value** for the MJT test is 0.0007331448. Since this **p-value** is smaller than $\alpha = 0.05$, the null hypothesis against the ordered alternative is rejected.

Terpstra-Magel test: TmTest(...)

The **TmTest** function in the **npordtests** package is used to perform the TM test.

```
R> TmTest(Values~Group,lehmann)
```

```
-----  
Test : Terpstra-Magel Test  
data : Values and Group
```

```
Statistic = 5173  
Mean = 2254  
Variance = 405043.8  
Z = 4.586518  
Asymp. p-value = 2.253498e-06
```

```
Result : Null hypothesis is rejected.
```

Here, the **Statistic** is calculated value of the test statistic. **p-value** is the significance value for this test. The **p-value** for the TM test is 0.000002253498. Thus, we can conclude that the null hypothesis of the equality of locations is rejected under setting $\alpha = 0.05$.

Ferdhiana-Terpstra-Magel test: FtmTest(...)

The **FtmTest** function in the **npordtests** package is used to perform the FTM test.

```
R> FtmTest(Values~Group,lehmann)
```

```
-----  
Test : Ferdhiana, Terpstra and Magel Test  
data : Values and Group
```

```
Statistic = NA
```

```

Mean = 0
Variance = 2294071
Z = NA
Asymp. p-value = NA

Error in if (p-value > alpha) { : missing value where TRUE/FALSE needed
In addition: Warning message:
In cor(t(Xmat), Ymat, method = "kendall") : the standard deviation is zero

```

As seen in the output, the error **standard deviation is zero** is encountered. This error occurs because the values of 68.5, 69.0, 70.5, 71.5, 73.0, 74.0, 74.5 are included in all groups.

KTP test: KtpTest(...)

The **KtpTest** function in the **npordtests** package is used to perform the KTP test.

```
R> KtpTest(Values~Group,lehmann)
```

```
-----
Test : KTP Test
data : Values and Group

Statistic = NA
Mean = 0
Variance = 2897517
Z = NA
Asymp. p-value = NA

Error in if (p-value > alpha) { : missing value where TRUE/FALSE needed
In addition: Warning message:
In cor(t(Xmat), Ymat, method = "spearman") : the standard deviation is zero

```

In the output, similar to **FtmTest**, the error **standard deviation is zero** is encountered.

S test: SsTest(...)

The **SsTest** function in the **npordtests** package is used to perform the S test.

```
R> SsTest(Values~Group,lehmann)
```

```
-----
Test : Shan's S test
data : Values and Group

Statistic = 32234
Mean = 20865.83
Variance = 6929623
Z = 4.318527
Asymp. p-value = 7.853701e-06

Result      : Null hypothesis is rejected.
-----
```

Here, the **Statistic** is calculated value of the test statistic. **p-value** is the significance value for the S test. The **p-value** for the S test is 0.000007853701. According to these results, because the p-value is smaller than $\alpha = 0.05$, the hypothesis of the equality of locations against the ordered alternative is rejected.

Gaur's Gc test: GcTest(...)

The **GcTest** function in the **npordtests** package is used to perform the Gaur's Gc test.

```
R> GcTest(Values~Group,lehmann)
```

```

Test : Gaur's Gc Test
data : Values and Group

Statistic = 0.1506891
Mean = 0
Variance = 0.03597884
Z = 0.7944348
Asymp. p-value = 0.2134712

Result : Null hypothesis is not rejected.

```

Here, the **Statistic** is calculated value of the test statistic. **p-value** is the significance value for the G_c test. The **p-value** for the G_c test is 0.2134712. Because this **p-value** is not smaller than $\alpha = 0.05$, the hypothesis of the equality of locations against the ordered alternative is not rejected.

Simulation study

In this section, we compared the JT, AT, Modified JT, TM, FTM, KTP, S and Gaur's Gc tests in terms of power and Type I error under some selected scenarios. Since the AT test includes the LS, RS, ST, LT, WS tests, these tests do not need to be compared. The number of iterations and nominal type I error are 10000 and .05, respectively. The five design factors manipulated in this simulation study are:

- number of samples ($k = 3$ and 4),
- average number of observations per group ($n = 5, 10, 20, 30$, and 50),
- sample size patterns (progressive, equal, and one extreme),
- distribution shapes (symmetric, left skewed, and right skewed),
- ordered alternatives shapes (linear, convex, and concave).

The sample size patterns in this simulation study are shown in Table 1. We used $\log-F(v_1, v_2)$ distributions to generate the random variable $X_{ij} = \theta_i + \varepsilon_{ij}$, where ε_{ij} is the iid $\log-F$ distribution, and θ_i is the location parameter; which is symmetric when $v_1 = v_2$, right skewed when $v_1 > v_2$, and left skewed when $v_1 < v_2$ (Terpstra et al., 2011).

Table 1: Simulation study sample size patterns. k is number of samples and n is average number of observations per group. The values in the table are sample sizes. For example, in case of $k = 3$, $n = 5$ and progressive pattern, the sample sizes of groups are 4, 5 and 6, respectively.

Sample size patterns																
		Progressive					Equal					One extreme				
$k = 3$		1	2	3	4	5	6	7	8	9	10	11	12	13	14	
Average n	5	10	20	30	50	5	10	20	30	50	5	10	20	30	50	
$k = 4$		1	2	3	4	5	6	7	8	9	10	11	12	13	14	
Average n	5	10	20	30	50	5	10	20	30	50	5	10	20	30	50	

In order to evaluate the performances of the tests, we consider the cases of $(v_1, v_2) = (5, 5)$, $(1, 10)$ and $(10, 1)$ for the symmetric, left skewed and right skewed populations, respectively.

While the location parameters of populations are equal, simulated type I error rates are calculated. Otherwise, in case the location parameters of the populations are not equal, the simulated powers of the tests are computed. In order to assess the robustness of the tests in terms of Type I

error rate, we used the robustness criterion recommended by Bradley (1978). This liberal criterion for the robustness is set at $\pm .5\alpha$ around the nominal alpha level. For instance, using the alpha level of .05, a test is considered robust when the simulated Type I error rates fall between .025 and .075.

Results

Figure 2 presents a set of boxplots based on the simulated Type I error rates for all scenarios considered while the nominal alpha level is .05. As shown in Figure 2, although all of the tests ensure the Bradley's liberal criterion, the JT, MJT, and FTM tests are the three best performing approaches that controlled nominal Type I error in all simulation scenarios. On the other hand, the TM test has a wider range than the others for the simulated type I error rates.

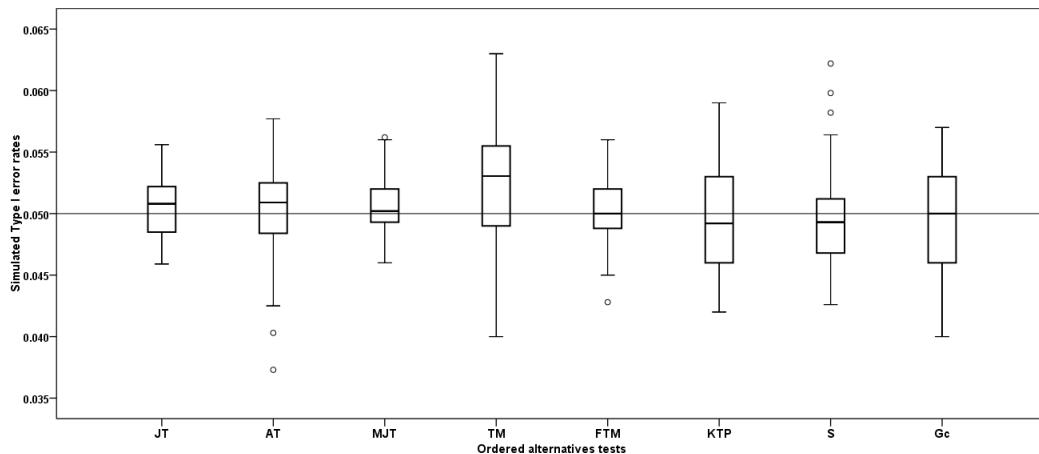


Figure 2: Distributions of simulated Type I error rates across all simulation scenarios when nominal alpha is .05. Each box plot gives median (the bold line that divides the box into two parts), lower and upper quartiles (start and end points of the box on the vertical axis) and min and max value (the horizontal lines outside the box). The outliers appear as the circles.

The simulated power values of the tests for the simulation scenarios above are given in Table 2-4. The results in these tables can be interpreted as follows:

- As seen in Table 2, when the data is generated from the symmetric distribution ($\log-F(5, 5)$), the most powerful test changes according to the shape of ordered alternative. When the shape of ordered alternative is linear, the MJT test are more powerful test than the other tests for all sample size patterns. On the other hand, when the shape of ordered alternative is convex, the S test has the highest power among all tests considered for all sample size patterns. Beside these, the simulated power values of KTP test for ordered alternative with concave shape are higher than those of the other tests when sample size patterns are progressive or one extreme. But, the S test is better than the other tests in terms of power when the sample size pattern is equal. On the other hand, when the average sample size for all distributions was quite large such as 50, the simulated power values for all tests were found to be quite close to 1.
- For the data generated from the $\log-F(1, 10)$ distribution which is a skewed to the left, when the shape of ordered alternative is linear, and average sample size is 5 or 10, the TM test for $k = 3$ gives better results, however, the AT test has the highest powers among the whole tests when average sample size is 20, 30, and 50. On the other hand, when $k = 4$ and average sample size is 10, the AT test has the highest powers among the whole tests. For the data generated from this distribution, the AT test, generally, is the most powerful test for ordered alternative with convex shape as seen in Table 3. For the data generated from the $\log-F(1, 10)$ distribution which is a skewed to the left, the TM test has the highest powers among the whole tests when the shape of ordered alternative is concave and average sample size is 5, but the KTP test for ordered alternative with concave shape is the most powerful among the whole tests when average sample size is 10, 20, 30, and 50. On the other hand, when the average sample size increased in all scenarios considered, the power values of all tests increased as expected.
- When the data is generated from the $\log-F(10, 1)$ distribution which is a skewed to the right, Table 4 shows that the AT test for ordered alternatives with linear shape, generally, gives

Table 2: Simulated power values ($1 - \beta$) of the test for log-F(5,5) distribution.

k	Test	Sample size pattern														
		Progressive						Equal						One Extreme		
		5	10	20	30	50	5	10	20	30	50	5	10	20	30	50
ordered alternatives shape=linear																
3	JT	.422	.706	.932	.979	1.00	.424	.701	.930	.980	1.00	.364	.586	.846	.945	.992
	AT	.372	.666	.922	.977	.998	.373	.661	.917	.967	1.00	.264	.549	.838	.952	.992
	MJT	.442	.709	.934	.993	1.00	.449	.722	.930	.984	1.00	.383	.606	.872	.999	1.00
	TM	.431	.665	.904	.948	.991	.434	.678	.907	.950	.994	.348	.512	.751	.888	.972
	FTM	.431	.696	.933	.978	.999	.415	.700	.929	.974	.997	.316	.542	.816	.979	.988
	KTP	.435	.704	.934	.978	.999	.436	.715	.930	.971	.996	.339	.552	.823	.900	.990
	S	.440	.702	.926	.990	1.00	.448	.699	.926	.967	1.00	.375	.595	.847	.973	.995
	Gc	.359	.674	.915	.924	.990	.388	.656	.914	.958	.990	.233	.439	.757	.841	.902
4	JT	.672	.960	.999	1.00	1.00	.766	.963	.999	1.00	1.00	.769	.962	.999	1.00	1.00
	AT	.616	.939	.999	1.00	1.00	.708	.952	.999	1.00	1.00	.700	.953	.999	1.00	1.00
	MJT	.678	.961	.999	1.00	1.00	.784	.978	.999	1.00	1.00	.782	.967	.999	1.00	1.00
	TM	.597	.894	.993	.999	1.00	.684	.919	.999	1.00	1.00	.586	.815	.999	1.00	1.00
	FTM	.621	.955	.999	1.00	1.00	.778	.972	.999	1.00	1.00	.719	.924	.999	1.00	1.00
	KTP	.592	.958	.999	1.00	1.00	.775	.965	.999	1.00	1.00	.731	.951	.999	1.00	1.00
	S	.667	.951	.999	1.00	1.00	.763	.961	.999	1.00	1.00	.768	.961	.999	1.00	1.00
	Gc	.485	.916	.999	1.00	1.00	.630	.948	.999	1.00	1.00	.455	.869	.992	.999	1.00
ordered alternatives shape=convex																
3	JT	.477	.716	.937	.988	1.00	.405	.682	.920	.977	.999	.486	.756	.955	.985	.999
	AT	.387	.674	.925	.981	1.00	.365	.662	.912	.980	.999	.311	.659	.920	.977	.998
	MJT	.476	.715	.936	.990	1.00	.411	.705	.926	.990	.999	.448	.708	.932	.984	.999
	TM	.388	.577	.827	.972	1.00	.395	.599	.813	.967	.984	.332	.448	.661	.969	.988
	FTM	.391	.672	.922	.980	1.00	.392	.679	.907	.978	.998	.284	.509	.789	.970	.995
	KTP	.421	.686	.921	.979	1.00	.410	.698	.930	.978	.992	.316	.530	.813	.972	.990
	S	.510	.742	.943	.992	1.00	.439	.716	.932	.992	1.00	.519	.765	.958	.988	.999
	Gc	.348	.648	.903	.974	.999	.371	.644	.905	.975	.992	.189	.440	.734	.945	.989
4	JT	.586	.771	.966	.999	1.00	.389	.623	.887	.996	1.00	.655	.894	.994	1.00	1.00
	AT	.468	.702	.938	.997	1.00	.360	.627	.887	.996	1.00	.527	.829	.980	1.00	1.00
	MJT	.525	.732	.950	.999	1.00	.399	.635	.887	.997	1.00	.585	.847	.986	1.00	1.00
	TM	.328	.476	.871	.989	1.00	.329	.473	.812	.990	1.00	.284	.421	.798	.985	1.00
	FTM	.286	.602	.940	.999	1.00	.380	.620	.893	.997	1.00	.333	.569	.914	1.00	1.00
	KTP	.279	.620	.941	.999	1.00	.384	.638	.880	.998	1.00	.320	.594	.838	.990	1.00
	S	.595	.780	.971	1.00	1.00	.401	.642	.899	.999	1.00	.659	.899	.995	1.00	1.00
	Gc	.222	.519	.827	.980	1.00	.309	.597	.833	.977	1.00	.219	.487	.765	.952	1.00
ordered alternatives shape=concave																
3	JT	.341	.633	.914	.984	1.00	.386	.688	.922	.992	1.00	.227	.363	.591	.704	.812
	AT	.305	.612	.897	.978	.998	.350	.642	.905	.987	1.00	.200	.418	.694	.816	.929
	MJT	.384	.661	.923	.993	1.00	.421	.689	.926	.994	1.00	.256	.453	.729	.837	.943
	TM	.381	.605	.839	.919	.952	.375	.595	.850	.919	1.00	.270	.426	.691	.801	.942
	FTM	.399	.678	.930	.998	1.00	.380	.679	.919	.999	1.00	.269	.515	.805	.925	.999
	KTP	.422	.683	.931	.999	1.00	.413	.690	.927	.999	1.00	.302	.538	.819	.931	.999
	S	.371	.663	.920	.990	1.00	.453	.714	.933	1.00	1.00	.245	.392	.639	.738	.846
	Gc	.361	.633	.904	.974	.995	.378	.626	.912	.980	1.00	.234	.433	.755	.863	.983
4	JT	.154	.449	.704	.782	.815	.376	.630	.893	.784	.816	.252	.401	.652	.583	.715
	AT	.158	.468	.755	.835	.862	.342	.604	.887	.842	.862	.250	.459	.757	.752	.872
	MJT	.176	.511	.777	.858	.878	.391	.643	.895	.860	.878	.300	.484	.766	.771	.889
	TM	.290	.484	.720	.810	.830	.332	.468	.821	.811	.842	.275	.405	.744	.774	.905
	FTM	.278	.611	.814	.884	.904	.376	.636	.877	.883	.934	.339	.575	.842	.888	.998
	KTP	.296	.619	.882	.952	.992	.388	.639	.902	.950	1.00	.359	.577	.860	.948	.999
	S	.145	.452	.719	.789	.809	.395	.658	.903	.961	1.00	.247	.417	.666	.614	.724
	Gc	.241	.535	.840	.912	.942	.363	.589	.838	.921	.962	.245	.496	.767	.890	.999

Table 3: Simulated power values ($1 - \beta$) of the test for log-F(1,10) distribution.

k	Test	Sample size pattern														
		Progressive						Equal						One Extreme		
		Average n		Average n		Average n		Average n		Average n		Average n				
ordered alternatives shape=linear																
3	JT	.179	.309	.495	.718	.941	.174	.296	.492	.688	.884	.145	.230	.383	.537	.694
	AT	.181	.318	.544	.815	1.00	.203	.316	.535	.765	.995	.146	.230	.411	.592	.813
	MJT	.198	.307	.496	.723	.950	.187	.304	.494	.684	.882	.145	.243	.398	.553	.708
	TM	.214	.319	.474	.684	.894	.217	.322	.457	.592	.727	.177	.250	.366	.482	.598
	FTM	.187	.313	.484	.689	.894	.185	.295	.475	.655	.835	.137	.209	.369	.529	.689
	KTP	.193	.303	.499	.734	.969	.197	.314	.500	.686	.872	.134	.220	.371	.522	.673
	S	.188	.303	.481	.695	.908	.199	.297	.483	.669	.855	.146	.225	.367	.509	.651
	Gc	.156	.284	.451	.651	.852	.178	.267	.467	.667	.867	.112	.188	.324	.460	.596
	ordered alternatives shape=convex															
4	JT	.293	.538	.801	1.00	1.00	.333	.549	.816	1.00	1.00	.347	.549	.828	1.00	1.00
	AT	.282	.578	.851	1.00	1.00	.353	.599	.863	1.00	1.00	.312	.596	.875	1.00	1.00
	MJT	.285	.545	.802	1.00	1.00	.346	.560	.816	1.00	1.00	.328	.569	.836	1.00	1.00
	TM	.302	.473	.761	1.00	1.00	.328	.491	.742	.993	1.00	.284	.444	.661	.878	1.00
	FTM	.231	.533	.772	1.00	1.00	.335	.562	.720	.878	1.00	.305	.522	.669	.816	.963
	KTP	.247	.530	.770	1.00	1.00	.359	.553	.815	1.00	1.00	.302	.517	.801	1.00	1.00
	S	.267	.510	.769	1.00	1.00	.338	.534	.801	1.00	1.00	.327	.521	.774	1.00	1.00
	Gc	.173	.446	.710	.994	1.00	.298	.493	.749	1.00	1.00	.204	.424	.670	.916	1.00
	ordered alternatives shape=concave															
3	JT	.212	.323	.500	.697	.910	.181	.309	.478	.649	.826	.208	.303	.540	.779	1.00
	AT	.227	.352	.566	.794	1.00	.220	.342	.559	.790	1.00	.215	.316	.543	.782	1.00
	MJT	.209	.317	.499	.687	.893	.190	.306	.489	.674	.877	.180	.292	.493	.704	.927
	TM	.220	.330	.502	.676	.866	.219	.330	.506	.688	.890	.199	.265	.420	.577	.738
	FTM	.191	.295	.485	.685	.899	.176	.298	.495	.702	.927	.144	.220	.392	.570	.766
	KTP	.187	.299	.481	.673	.881	.190	.316	.486	.668	.858	.142	.213	.364	.521	.688
	S	.217	.324	.501	.680	.863	.204	.305	.484	.681	.898	.206	.313	.490	.687	.902
	Gc	.164	.288	.442	.602	.768	.186	.280	.462	.660	.874	.117	.192	.322	.462	.620
	ordered alternatives shape=concave															
4	JT	.264	.349	.555	.769	.989	.186	.283	.450	.631	.824	.275	.440	.701	.978	1.00
	AT	.244	.371	.614	.861	1.00	.209	.324	.521	.738	.957	.244	.442	.705	.980	1.00
	MJT	.223	.331	.521	.713	.911	.184	.283	.447	.629	.827	.237	.389	.641	.897	1.00
	TM	.224	.287	.560	.839	1.00	.203	.282	.468	.656	.860	.191	.258	.567	.894	1.00
	FTM	.140	.257	.546	.841	1.00	.180	.277	.457	.647	.851	.156	.258	.544	.848	1.00
	KTP	.142	.285	.509	.749	.993	.193	.300	.468	.644	.824	.160	.240	.412	.592	.776
	S	.251	.335	.538	.753	.984	.182	.275	.439	.609	.795	.259	.407	.658	.919	1.00
	Gc	.127	.226	.372	.522	.686	.167	.289	.369	.457	.551	.116	.209	.377	.559	.747
	ordered alternatives shape=concave															
3	JT	.150	.274	.467	.670	.891	.165	.285	.481	.679	.887	.118	.163	.245	.331	.429
	AT	.140	.270	.485	.716	.955	.179	.294	.511	.736	.977	.135	.149	.284	.437	.596
	MJT	.172	.278	.470	.678	.904	.178	.294	.485	.696	.909	.119	.182	.296	.422	.552
	TM	.181	.256	.375	.508	.655	.194	.256	.386	.526	.678	.154	.190	.293	.404	.525
	FTM	.166	.285	.476	.679	.896	.176	.288	.492	.712	.948	.119	.203	.360	.527	.710
	KTP	.175	.288	.490	.722	.966	.179	.315	.516	.743	.982	.118	.212	.369	.542	.725
	S	.162	.268	.461	.666	.889	.190	.300	.484	.688	.900	.127	.157	.245	.343	.457
	Gc	.163	.274	.436	.610	.796	.141	.242	.406	.572	.752	.130	.167	.294	.433	.582
	ordered alternatives shape=concave															
4	JT	.093	.195	.305	.433	.567	.167	.269	.445	.641	.839	.123	.181	.286	.405	.528
	AT	.082	.205	.332	.479	.636	.160	.271	.459	.649	.843	.105	.190	.325	.462	.617
	MJT	.101	.208	.350	.504	.668	.172	.277	.448	.633	.830	.126	.201	.337	.485	.647
	TM	.166	.197	.288	.389	.508	.175	.181	.351	.529	.709	.153	.177	.330	.495	.668
	FTM	.119	.247	.356	.473	.598	.162	.248	.458	.686	.930	.139	.219	.391	.583	.791
	KTP	.117	.279	.452	.639	.836	.169	.298	.462	.696	.932	.141	.225	.396	.585	.794
	S	.087	.178	.294	.428	.576	.168	.255	.433	.617	.807	.125	.173	.266	.371	.490
	Gc	.114	.203	.333	.473	.619	.124	.260	.381	.514	.659	.107	.216	.311	.426	.547

Table 4: Simulated power values ($1 - \beta$) of the test for log-F(10,1) distribution.

k	Test	Sample size pattern														
		Progressive						Equal						One Extreme		
		5	10	20	30	50	5	10	20	30	50	5	10	20	30	50
ordered alternatives shape=linear																
3	JT	.190	.310	.491	.682	.893	.179	.298	.491	.690	.909	.180	.256	.404	.566	.742
	AT	.182	.315	.526	.747	.976	.203	.323	.527	.741	.967	.204	.302	.473	.654	.847
	MJT	.208	.304	.490	.690	.896	.191	.318	.491	.676	.863	.176	.266	.415	.574	.753
	TM	.215	.293	.465	.645	.845	.221	.298	.438	.590	.750	.213	.251	.350	.455	.564
	FTM	.187	.280	.482	.700	.924	.186	.290	.463	.642	.827	.158	.242	.390	.540	.698
	KTP	.184	.304	.484	.682	.888	.185	.305	.493	.689	.897	.178	.249	.393	.553	.721
	S	.199	.305	.477	.663	.863	.195	.304	.487	.688	.907	.208	.291	.433	.593	.773
	Gc	.153	.279	.454	.649	.856	.168	.263	.458	.655	.862	.127	.195	.324	.461	.614
4	JT	.305	.537	.796	1.00	1.00	.340	.554	.819	1.00	1.00	.350	.555	.806	1.00	1.00
	AT	.319	.570	.834	1.00	1.00	.344	.581	.851	1.00	1.00	.379	.594	.848	1.00	1.00
	MJT	.304	.544	.801	1.00	1.00	.342	.550	.819	1.00	1.00	.360	.558	.816	1.00	1.00
	TM	.313	.485	.800	1.00	1.00	.319	.478	.812	1.00	1.00	.297	.428	.788	1.00	1.00
	FTM	.262	.535	.812	1.00	1.00	.337	.557	.822	1.00	1.00	.323	.510	.797	1.00	1.00
	KTP	.269	.505	.785	1.00	1.00	.339	.566	.835	1.00	1.00	.334	.511	.782	1.00	1.00
	S	.295	.518	.804	1.00	1.00	.332	.532	.805	1.00	1.00	.364	.576	.823	1.00	1.00
	Gc	.193	.442	.697	.960	1.00	.266	.533	.760	.999	1.00	.207	.430	.670	.916	1.00
ordered alternatives shape=convex																
3	JT	.206	.310	.505	.714	.933	.167	.286	.474	.670	.880	.232	.337	.533	.741	.963
	AT	.176	.307	.510	.757	1.00	.175	.289	.501	.725	.953	.228	.349	.530	.729	.948
	MJT	.210	.310	.497	.696	.913	.177	.302	.487	.690	.895	.214	.320	.482	.646	.816
	TM	.182	.235	.406	.587	.788	.186	.271	.394	.527	.678	.193	.216	.300	.396	.510
	FTM	.174	.253	.503	.723	.956	.167	.304	.499	.708	.935	.162	.228	.354	.492	.646
	KTP	.178	.297	.481	.673	.873	.174	.303	.484	.683	.890	.175	.245	.379	.533	.695
	S	.219	.320	.501	.692	.897	.189	.311	.484	.661	.852	.261	.381	.573	.767	.973
	Gc	.135	.263	.443	.629	.821	.154	.253	.446	.657	.882	.113	.187	.315	.463	.621
4	JT	.254	.344	.566	.806	1.00	.161	.260	.447	.648	.855	.284	.455	.697	.941	1.00
	AT	.227	.326	.546	.782	1.00	.156	.262	.451	.642	.837	.263	.425	.670	.929	1.00
	MJT	.221	.317	.522	.747	.992	.171	.276	.452	.662	.882	.247	.397	.626	.857	1.00
	TM	.198	.208	.540	.892	1.00	.164	.211	.365	.521	.679	.158	.202	.324	.464	.620
	FTM	.149	.280	.556	.834	1.00	.172	.272	.444	.636	.842	.169	.259	.432	.617	.818
	KTP	.155	.250	.515	.800	1.00	.195	.268	.450	.652	.870	.167	.279	.462	.649	.850
	S	.257	.354	.572	.898	1.00	.162	.257	.437	.629	.841	.304	.479	.728	.995	1.00
	Gc	.109	.213	.331	.461	.601	.145	.242	.383	.526	.675	.113	.226	.332	.452	.580
ordered alternatives shape=concave																
3	JT	.165	.286	.467	.654	.857	.179	.299	.485	.675	.877	.139	.178	.265	.354	.445
	AT	.179	.316	.505	.696	.903	.216	.331	.524	.737	.952	.176	.249	.375	.521	.673
	MJT	.192	.286	.475	.670	.883	.184	.306	.480	.670	.862	.153	.218	.329	.458	.605
	TM	.226	.341	.508	.700	.916	.223	.339	.528	.741	.960	.187	.265	.398	.593	.782
	FTM	.196	.306	.474	.662	.868	.180	.300	.498	.710	.930	.160	.211	.385	.573	.771
	KTP	.194	.313	.486	.663	.842	.184	.310	.471	.644	.825	.183	.250	.393	.548	.707
	S	.177	.290	.477	.674	.877	.204	.306	.486	.682	.894	.157	.208	.304	.412	.532
	Gc	.150	.288	.453	.638	.831	.162	.272	.420	.588	.760	.152	.181	.353	.541	.731
4	JT	.104	.213	.324	.447	.576	.180	.280	.456	.638	.822	.138	.191	.295	.405	.535
	AT	.131	.257	.416	.589	.778	.188	.318	.480	.644	.810	.167	.263	.406	.561	.720
	MJT	.117	.232	.365	.518	.679	.179	.281	.455	.637	.839	.156	.232	.356	.500	.662
	TM	.208	.280	.433	.602	.779	.210	.332	.491	.652	.855	.186	.287	.441	.613	.795
	FTM	.160	.276	.425	.576	.737	.181	.271	.445	.635	.841	.171	.269	.410	.561	.724
	KTP	.160	.271	.426	.585	.746	.192	.301	.466	.641	.826	.172	.283	.432	.601	.778
	S	.103	.207	.322	.455	.600	.184	.273	.440	.609	.798	.140	.200	.302	.424	.554
	Gc	.144	.246	.345	.446	.557	.174	.289	.398	.523	.654	.141	.213	.329	.455	.583

better results. On the other hand, when average sample size is 5 and $k = 3$ the TM test for this situation is the most powerful test. As seen in Table 4, when the shape of ordered alternative is a convex, it is observed that the S test generally yields the highest power values. In addition, while the sample size patterns are progressive and equal, and average sample size is 20, 30, and 50, the power values of the AT test for this situation are greater than those of the others. By the examination of the results in Table 6, when ordered alternative has a concave shape, it is seen that the TM test is the most powerful test among the whole tests.

Table 5 gives decision rules indicating which test is more appropriate for which design.

When the ordered alternative has a linear shape and the distribution is symmetric, the MJT test should be preferred. However, when the ordered alternative has a linear shape and the distribution is skewed to left and average sample size is 5 or 10, it can be stated that the TM test has a more significant power advantage than the others. On the other hand, average sample size is 20, 30, or 50, it can be said that the AT test has a more significant power advantage than the others.

On the other hand, when the ordered alternative has a convex shape, the AT test is recommended for the distributions skewed to left. However, if these distributions are symmetric, the S test is proposed. Besides this, if the distributions are skewed to right and the sample size pattern is equal, then the MJT test is recommended. Further, if the distributions are skewed to right and the sample size pattern is progressive or one extreme, then S test is used.

When the ordered alternative has a concave shape and the sample size pattern is equal, then the S test is used for symmetric distribution. In addition, when the ordered alternative has a concave shape and the sample size pattern is progressive or one extreme, then the KTP test is recommended for symmetric distribution. Moreover, if the distributions are skewed to left and the sample size is 5, TM test is recommended, but in the case of 10, 20, 30, 50 for the sample size, the KTP test is recommended. Finally, if the distributions are skewed to right, the TM test is recommended.

Table 5: The rules based on the simulation results for choice the test. For example, when the ordered alternative has a linear shape and the distribution is symmetric, the MJT test should be preferred.

Alternative hypothesis	Distribution shape	Sample size pattern	Average sample size	Test
Linear	symmetric	-	-	MJT
	skewed to left	-	5, 10	TM
	skewed to left	-	20, 30, 50	AT
	skewed to right	-	-	AT
Convex	symmetric	-	-	S
	skewed to left	-	-	AT
	skewed to right	-	-	S
Concave	symmetric	Equal	-	S
	symmetric	Progressive or One Extreme	-	KTP
	skewed to left	-	5	TM
	skewed to left	-	10, 20, 30, 50	KTP
	skewed to right	-	-	TM

Summary

Tests for ordered alternative are the most frequently used nonparametric methods in a wide range of statistical and medical applications. For example, the evaluation of preclinical studies, clinical dose-finding trials, typical toxicity studies, education studies, agricultural studies and etc. We present the **npordtests** package to test the equality hypothesis of the locations against ordered alternative.

In this paper, we compared the tests included in the **npordtests** package in terms of Type I error rate and power. With the results of the simulation study, when the data is generated from a symmetric distribution, we propose that the use of the MJT test for ordered alternatives with linear shape and the S test for ordered alternatives with convex shape. On the other hand, when ordered alternative has a concave shape, the S test for equal sample size patterns is suggested, but the KTP test is recommended when sample size patterns are progressive and one extreme. For the data generated from a left skewed distribution, when $k = 3$ and shape of ordered alternative is linear, we recommend that the use of the TM test for small sample sizes such as $n = 5$ and 10, and the AT test for sample size 20, 30, and 50. However, when $k = 4$ and sample sizes are 10, 20, 30,

and 50, we propose to prefer the AT test. For this kind of data, we propose the use of the AT test when the ordered alternative has a convex shape. On the other hand, if ordered alternative has a concave shape, we propose that the use of the KTP test for sample sizes such as $n = 10, 20, 30$, and 50 and the TM test for small sample size such as $n = 5$. For the data generated from a right skewed distribution, when $k = 4$, we recommend that the use of the AT test for ordered alternative with linear shape. However, when $k = 3$, and the shape of ordered alternative is linear, we propose to choose the AT test for sample sizes $n = 10, 20, 30$, and 50 and the TM test for sample size 5. On the other hand, when ordered alternative has a concave shape, the TM test is the most powerful test in all simulation scenarios. Besides these, for this kind of data, it is understood that it is appropriate to prefer the S test for ordered alternative with convex shape.

To test the equality hypothesis of locations parameters against ordered alternatives, the **npordtests** package covers the prominent nonparametric tests such as Jonckheere-Terpstra test, Beier and Buning's Adaptive test, Modified Jonckheere-Terpstra test, Terpstra-Magel test, Ferdhiana-Terpstra-Magel test, KTP test, S test and Gaur's G_c test. According to the authors knowledge, the tests which are present in the **npordtests** package, except the JT test, are not available in any other R tool. The package will be updated at regular intervals.

Acknowledgments

The authors are genuinely grateful to anonymous reviewer and the Executive Editor (Dianne Cook) for their invaluable contributions to the improvement of our paper.

Bibliography

- B. Altunkaynak and H. Gamgam. *npordtests: An R Package for Nonparametric Tests for Equality of Location Against Ordered Alternatives*, 2019. URL <https://CRAN.R-project.org/package=npordtests>. R package version 1.2. [p]
- F. Beier and H. Buning. An adaptive test against ordered alternatives. *Computational Statistics and Data Analysis*, 25(4):441–452, 1997. URL [https://doi.org/10.1016/S0167-9473\(97\)00014-5](https://doi.org/10.1016/S0167-9473(97)00014-5). [p]
- J. V. Bradley. Robustness? *British Journal of Mathematical and Statistical Psychology*, 31:144–152, 1978. URL <https://doi.org/10.1111/j.2044-8317.1978.tb00581.x>. [p]
- M. A. Bredella, L. S. Steinbach, S. Morgan, M. Ward, and J. C. Davis. MRI of the sacroiliac joints in patients with moderate to severe ankylosing spondylitis. *American Journal of Roentgenology*, 187(6):1420–1426, 2006. URL <https://doi.org/10.2214/AJR.05.1423>. [p]
- H. Buning and W. Kossler. Robustness and efficiency of some tests for ordered alternatives in the c-sample location problem. *Journal of Statistical Computation and Simulation*, 55(4):337–352, 1996. URL <https://doi.org/10.1080/00949659608811774>. [p]
- C. H. Chang and C. H. Yen. A nonparametric test for the ordered alternative based on fast discrete Fourier transform coefficient. *Journal of Testing and Evaluation*, 39(6):1131–1143, 2011. URL <https://doi.org/10.1520/Jte103464>. [p]
- Q. Cheng, W. Yang, S. C. Raimondi, C.-H. Pui, M. V. Relling, and W. E. Evans. Karyotypic abnormalities create discordance of germline genotype and cancer cell phenotypes. *Nature Genetics*, 37(8):878–882, 2005. URL <https://doi.org/10.1038/ng1612>. [p]
- J. Cuzick. A Wilcoxon-Type test for trend. *Statistics in Medicine*, 4(1):87–90, 1985. URL <https://doi.org/10.1002/sim.4780040112>. [p]
- M. Desu and D. Raghavarao. *Nonparametric Statistical Methods for Complete and Censored Data*. Chapman and Hall/CRC, New York, 1st edition, 2004. [p]
- A. Dmitrienko, C. Chuang-Stein, and R. D' Agostino. *Pharmaceutical Statistics using SAS®: A practical guide (SAS Press)*. SAS Institute, NC, 1st edition, 2006. ISBN 159047886X. [p]
- R. Ferdhiana, J. Terpstra, and R. C. Magel. A nonparametric test for the ordered alternative based on Kendall's Correlation Coefficient. *Communications in Statistics-Simulation and Computation*, 37(6):1117–1128, 2008. URL <https://doi.org/10.1080/03610910801894870>. [p]

- J. L. Gastwirth. Asymptotically most powerful rank tests for the two-sample problem with censored data. *Ann. Math. Statist.*, 36(4):1243–1247, 1965. URL <https://doi.org/10.1214/aoms/1177699995>. [p]
- A. Gaur. A new class of distribution-free tests for testing ordered location parameters based on sub-samples. *Statistics and Probability Letters*, 90(1):53–59, 2014. URL <https://doi.org/10.1016/j.spl.2014.03.011>. [p]
- A. Gaur. A class of k-sample distribution-free tests for location against ordered alternatives. *Communications in Statistics-Theory and Methods*, 46(5):2343–2353, 2017. URL <https://doi.org/10.1080/03610926.2015.1041986>. [p]
- Z. Govindarajulu and H. S. Haller. c-sample tests of homogeneity against ordered alternatives. In J. S. Rustagi, editor, *Optimizing Methods in Statistics*, page 479. Academic Press, 1971. ISBN 978-0-12-604550-5. URL <https://doi.org/10.1016/B978-0-12-604550-5.50030-4>. [p]
- S. Hoffmeyer, O. Burk, O. von Richter, H. P. Arnold, J. Brockmöller, A. Johne, I. Cascorbi, T. Gerloff, I. Roots, M. Eichelbaum, and U. Brinkmann. Functional polymorphisms of the human multidrug resistance gene: Multiple sequence variations and correlation of one allele with p glycoprotein expression and activity in vivo. *Proceedings of the National Academy of Sciences of the United States of America*, 97(7):3473–3478, 2000. URL <https://doi.org/10.1073/pnas.050585397>. [p]
- J. W. Hogg, Robert V. McKean and A. T. Craig. *Introduction to Mathematical Statistics*. Pearson, Boston, 7 edition, 2013. ISBN 9780321795434. [p]
- R. V. Hogg, D. M. Fisher, and R. H. Randles. A two-sample adaptive distribution-free test. *Journal of the American Statistical Association*, 70(351):656–661, 1975. URL <https://doi.org/10.2307/2285950>. [p]
- A. R. Jonckheere. A distribution-free k-sample test against ordered alternatives. *Biometrika*, 41(1):133–145, 1954. URL <https://doi.org/10.2307/2333011>. [p]
- T. Kawaguchi, Y. Sumida, A. Umemura, K. Matsuo, M. Takahashi, T. Takamura, K. Yasui, T. Saibara, E. Hashimoto, M. Kawanaka, S. Watanabe, S. Kawata, Y. Imai, M. Kokubo, T. Shima, H. Park, H. Tanaka, K. Tajima, R. Yamada, F. Matsuda, T. Okanoue, and D. Japan Study Group of Nonalcoholic Fatty Liver. Genetic polymorphisms of the human PNPLA3 gene are strongly associated with severity of non-alcoholic fatty liver disease in Japanese. *PloS one*, 7(6):e38322–e38322, 2012. URL <https://doi.org/10.1371/journal.pone.0038322>. [p]
- C. Le. A new rank test against ordered-alternatives in k-sample problems. *Biometrical Journal*, 30(1):87–92, 1988. URL <https://doi.org/10.1002/bimj.4710300116>. [p]
- E. Lehmann. *Nonparametrics: Statistical Methods based on Ranks*. Holden-Day, San Francisco, 1st edition, 1975. ISBN 9780387352121. [p]
- J. Lin, A. Sibley, I. Shterev, and K. Owzar. *jtGWAS: Efficient Jonckheere-Terpstra Test Statistics*, 2017a. URL <https://CRAN.R-project.org/package=jtGWAS>. R package version 1.5.1. [p]
- J. Lin, A. Sibley, I. Shterev, and K. Owzar. *fastJT: Efficient Jonckheere-Terpstra Test Statistics for Robust Machine Learning and Genome-Wide Association Studies*, 2017b. URL <https://CRAN.R-project.org/package=fastJT>. R package version 1.0.4. [p]
- M. Neuhäuser, P.-Y. Liu, and L. A. Hothorn. Nonparametric tests for trend: Jonckheere's Test, a modification and a maximum test. *Biometrical Journal*, 40(8):899–909, 1998. URL [https://doi.org/10.1002/\(SICI\)1521-4036\(199812\)40:8<899::AID-BIMJ899>3.0.CO;2-9](https://doi.org/10.1002/(SICI)1521-4036(199812)40:8<899::AID-BIMJ899>3.0.CO;2-9). [p]
- J. P. Ong, A. Aggarwal, D. Krieger, K. A. Easley, M. T. Karafa, F. Van Lente, A. C. Arroliga, and K. D. Mullen. Correlation between ammonia levels and the severity of hepatic encephalopathy. *The American Journal of Medicine*, 114(3):188–193, 2003. URL [https://doi.org/10.1016/S0002-9343\(02\)01477-8](https://doi.org/10.1016/S0002-9343(02)01477-8). [p]
- J. F. Pedersen, L. G. Madsen, V. A. Larsen, O. Hamberg, T. Horn, B. Federspiel, and P. Bytzer. A doppler waveform index to characterize hepatic vein velocity pattern and evaluate hepatic fibrosis. *Journal of Clinical Ultrasound*, 36(4):208–211, 2008. URL <https://doi.org/10.1002/jcu.20446>. [p]
- T. Pohlert. *PMCMRplus: Calculate Pairwise Multiple Comparisons of Mean Rank Sums Extended*, 2018. URL <https://CRAN.R-project.org/package=PMCMRplus>. R package version 1.4.1. [p]

- M. L. Puri. Some distribution-free k-sample rank tests of homogeneity against ordered alternatives. *Communications on Pure and Applied Mathematics*, 18(1):51–63, 1965. URL <https://doi.org/10.1002/cpa.3160180108>. [p]
- F. Scholz and A. Zhu. *kSamples: K-Sample Rank Tests and Their Combinations*, 2018. URL <https://CRAN.R-project.org/package=kSamples>. R package version 1.2-8. [p]
- G. G. Shan, D. Young, and L. Kang. A new powerful nonparametric rank test for ordered alternative problem. *Plos One*, 9(11):1–10, 2014. URL <https://doi.org/10.1371/journal.pone.0112924>. [p]
- J. Swintek, K. Flynn, and J. Haselman. *StatCharrms: Statistical Analysis of Chemistry, Histopathology, and Reproduction Endpoints Including Repeated Measures and Multi-Generation Studies*, 2018. URL <https://CRAN.R-project.org/package=StatCharrms>. R package version 0.90.91. [p]
- H.-L. Tan, S. M. Zain, R. Mohamed, S. Rampal, K.-F. Chin, R. C. Basu, P.-L. Cheah, S. Mahadeva, and Z. Mohamed. Association of glucokinase regulatory gene polymorphisms with risk and severity of non-alcoholic fatty liver disease: An interaction study with adiponutrin gene. *Journal of Gastroenterology*, 49(6):1056–1064, 2014. URL <https://doi.org/10.1007/s00535-013-0850-x>. [p]
- J. T. Terpstra. The asymptotic normality and consistency of Kendall’s test against trend, when ties are present in one ranking. *Indagationes Mathematicae*, 14(3):327–333, 1952. [p]
- J. T. Terpstra and R. C. Magel. A new nonparametric test for the ordered alternative problem. *Journal of Nonparametric Statistics*, 15(3):289–301, 2003. URL <https://doi.org/10.1080/1048525031000078349>. [p]
- J. T. Terpstra, C. H. Chang, and R. C. Magel. On the use of Spearman’s Correlation Coefficient for testing ordered alternatives. *Journal of Statistical Computation and Simulation*, 81(11):1381–1392, 2011. URL <https://doi.org/10.1080/00949655.2010.485316>. [p]
- P. Tryon and T. Hettmansperger. A class of nonparametric tests for homogeneity against ordered alternatives. *Annals of Statistics*, 1:1061–1070, 1973. [p]
- T. Uchiyama, H. Kanno, K. Ishitani, H. Fujii, H. Ohta, H. Matsui, N. Kamatani, and K. Saito. An snp in CYP39A1 is associated with severe neutropenia induced by docetaxel. *Cancer Chemotherapy and Pharmacology*, 69(6):1617–1624, 2012. URL <https://doi.org/10.1007/s00280-012-1872-4>. [p]
- E. Venkatraman. *clinfun: Clinical Trial Design and Data Analysis Functions*, 2018. URL <https://CRAN.R-project.org/package=clinfun>. R package version 1.0.15. [p]
- K. Yorifuji, Y. Uemura, S. Horibata, G. Tsuji, Y. Suzuki, K. Miyagawa, K. Nakayama, K.-i. Hirata, S. Kumagai, and N. Emoto. CHST3 and CHST13 polymorphisms as predictors of bosentan-induced liver toxicity in Japanese patients with pulmonary arterial hypertension. *Pharmacological Research*, 135:259–264, 2018. URL <https://doi.org/10.1016/j.phrs.2018.08.011>. [p]

Bulent Altunkaynak
Department of Statistics
Gazi University Faculty of Science, Yenimahalle, Ankara
Turkey
bulenta@gazi.edu.tr

Hamza Gamgam
Department of Statistics
Gazi University Faculty of Science, Yenimahalle, Ankara
Turkey
gamgam@gazi.edu.tr

rcosmo: R Package for Analysis of Spherical, HEALPix and Cosmological Data

by Daniel Fryer, Ming Li, Andriy Olenko

Abstract The analysis of spatial observations on a sphere is important in areas such as geosciences, physics and embryo research, just to name a few. The purpose of the package **rcosmo** is to conduct efficient information processing, visualisation, manipulation and spatial statistical analysis of Cosmic Microwave Background (CMB) radiation and other spherical data. The package was developed for spherical data stored in the Hierarchical Equal Area isoLatitude Pixelation (Healpix) representation. **rcosmo** has more than 100 different functions. Most of them initially were developed for CMB, but also can be used for other spherical data as **rcosmo** contains tools for transforming spherical data in cartesian and geographic coordinates into the HEALPix representation. We give a general description of the package and illustrate some important functionalities and benchmarks.

Introduction

Directional statistics deals with data observed at a set of spatial directions, which are usually positioned on the surface of the unit sphere or star-shaped random particles. Spherical methods are important research tools in geospatial, biological, palaeomagnetic and astrostatistical analysis, just to name a few. The books (Fisher et al., 1987; Mardia and Jupp, 2009) provide comprehensive overviews of classical practical spherical statistical methods. Various stochastic and statistical inference modelling issues are covered in (Yadrenko, 1983; Marinucci and Peccati, 2011).

The CRAN Task View *Spatial* shows several packages for Earth-referenced data mapping and analysis. All currently available R packages for spherical data can be classified in three broad groups.

The first group provides various functions for working with geographic and spherical coordinate systems and their visualizations. Probably the most commonly used R package to represent spatial maps and data is **sp** (Bivand et al., 2013). It includes tools for spatial selection, referencing and plotting spatial data as maps. It has comprehensive hierarchical classes and methods for spatial 2d and 3d data. Another example, **sphereplot** (Robotham, 2013), uses **rgl** (Adler et al., 2018) to create 3d visualizations in the spherical coordinate system. The functions **car2sph** and **sph2car** implement transformations between Cartesian and spherical coordinates. The package **geosphere** (Hijmans, 2017) includes functions for computing distances, directions and areas for geographic coordinates.

The second group covers various numerical procedures that can be useful for spherical approximations and computations. For example, **SpherWave** (Fernández-Durán and Gregorio-Domínguez, 2016) is developed to implement the spherical wavelets and conduct the multiresolution analysis on the sphere. Functions for numerical integration over high-dimensional spheres and balls are provided in the package **SphericalCubature** (Nolan and University, 2017).

The third group provides statistical tools for spherical data analysis. In this group, the most commonly used packages are **RandomFields** (Schlather et al., 2015) and **geoR** (Ribeiro Jr and Diggle, 2018). These packages provide a number of tools for model estimation, spatial inference, simulation, kriging, spectral and covariance analyses. Most of their underlying models are for 2d or 3d data, but some additional spherical models are listed for future developments. The package **Directional** (Tsagris et al., 2019) has functions for von Mises-Fisher kernel density estimation, discriminant and regression analysis on the sphere. The package **gensphere** (Nolan, 2017) implements multivariate spherical distributions. **CircNNTSR** (Fernández-Durán and Gregorio-Domínguez, 2016) provides functions for statistical analysis of spherical data by means of non-negative trigonometric sums. The package **VecStatGraphs3D** (Felicísimo et al., 2014) conducts statistical analysis on 3d vectors. It includes estimation of parameters and some spherical test. Another example is the package **sm** (Bowman and Azzalini, 2018) for spherical regression analysis and non-parametric density estimation.

There are also several R packages developed for spherical data in astronomy¹. For example, **cosmoFns** (Harris, 2012) and **CRAC** (Liu, 2014) implement functions to compute spherical geometric quantities useful for cosmological research. The package **FITSio** (Harris, 2016) reads and writes files

¹see the list in <https://asaip.psu.edu/forums/software-forum/459833927>

in one of standard astronomical formats, FITS (Flexible Image Transport System). `spider` (Brown et al., 2012) includes functions for all-sky grid/scatter plots under various astronomical coordinate systems (equatorial, ecliptic, galactic). The package `astro` (Kelvin, 2014) provides functions for basic cosmological statistics and FITS file manipulations.

In recent years the spatial analysis and theory of spherical data have been strongly motivated by the studies on the Cosmic Microwave Background (CMB) radiation data collected by NASA and European Space Agency missions COBE, WMAP and Planck and usually stored in the Hierarchical Equal Area isoLatitude Pixelation (HEALPix) format. Cosmologists have developed comprehensive Python and MATLAB software packages² to work with CMB and HEALPix data.

Although the mentioned before R packages can be used for geographic or spherical coordinate referenced data, comprehensive and easy to use tools for CMB and HEALPix data are not available in R, which motivated the authors to design the package `rcosmo` (Fryer et al., 2020).

The aims of the package `rcosmo` are

- to give convenient access and integrated in one package tools for analysis and visualisation of CMB and HEALPix data to the R statistical community;
- to develop R functions for models and methods in spherical statistics that are not available in the existing R packages;
- to extend familiar R classes to spherical HEALPix data making them cross-compatible and intuitively interactable with many existing R statistical packages.

The HEALPix format has numerous advantages to classical geographic representations of spherical data, see (Gorski et al., 2005). R implementation of computationally expensive statistical and geometrical methods, such as nearest neighbour searches, empirical covariance function estimation, uniform sampling, spectral density estimation, in a way that takes advantage of the HEALPix data structure, could be useful for geostatistics and other applications. It can reduce algorithmic complexity and computational time. Various data processing, manipulation, visualisation and statistical analysis tasks are achieved efficiently in `rcosmo`, using optimised C++ code where necessary.

Basics of CMB data

In the Standard Cosmological Model, the Cosmic Microwave Background is redshifted microwave frequency light that is believed to have originated around 14 billion years ago. CMB is the main source of data about the early universe and seeds of future galaxies. Bell Labs physicists Arno Penzias and Robert Wilson received the Nobel prize in physics in 1978 for their famously happenstance discovery of CMB radiation as an inconvenient background “noise” during their experimentation with the Holmdel Horn Antenna radio telescope (Durrer, 2015).

Over a decade later, NASA’s Cosmic Microwave Background Explorer (COBE) satellite mission produced the first detailed full CMB sky map (Efstathiou et al., 1992). Referred to as the dawn of precision cosmology, COBE results provided fine constraints on many cosmological parameters. Particular attention has been paid to the existence of CMB anisotropies and associated non-Gaussianity, usually investigated through the CMB angular power spectrum (Durrer, 2015). The Wilkinson Microwave Anisotropy Probe, was launched in 2001 by NASA and returned more precise measurements of CMB (Bennett et al., 2003). Then, the third and most detailed space mission to date was conducted by the European Space Agency, via the Planck Surveyor satellite (Adam et al., 2016). The radiation that astronomers detect today forms an expanding spherical surface of radius approximately 46.5 billion light years. The next generation of CMB experiments, CMB-S4, LiteBIRD, and CORE, will consist of highly sensitive telescopes. It is expected that these experiments will provide enormous amount of CMB measurements and maps to nearly the cosmic variance limit.

The term “CMB data” refers to a broad range of location tagged quantities describing properties of the CMB. For example, the Infrared Science Archive by Caltech’s Infrared Processing and Analysis Center (IPAC) hosts curated CMB products from the North American Space Agency (NASA)³.

To produce CMB maps (see Figure 5), the products of the Planck mission data (in the range of frequencies from 30 to 857 GHz) are separated from foreground noise using one of the four detailed methods named COMMANDER, NILC, SEVEM and SMICA⁴. These CMB maps are provided at

²<https://healpix.sourceforge.io/>, <https://healpy.readthedocs.io>, <http://sufoo.c.occo.jp/program/healpix.html>

³hosted at the link <http://irsa.ipac.caltech.edu>

⁴https://wiki.cosmos.esa.int/planckpla2015/index.php/Astrophysical_component_separation

either low resolution ($N_{\text{side}} = 1024$, i.e., 10 arcmin resolution), or high resolution ($N_{\text{side}} = 2048$, i.e., 5 arcmin resolution). The maps include temperature intensity and polarisation data, as well as common masks for identifying regions where the reconstructed CMB is untrusted.

Our focus will mostly be on the CMB temperature intensity data. In Planck CMB products, these data are stored as 4-byte floating point binary numbers in K_{cmb} defined as the unit in which a black body spectrum at 2.725 Kelvin (K) is flat with respect to the frequency (Hurier et al., 2015).

The map of the CMB temperature is usually modelled as a realization of an isotropic Gaussian random field on the unit sphere. The Appendix introduces a statistical model and basic notations of spherical random fields. More details can be found, for example, in the monographs (Marinucci and Peccati, 2011), (Yadrenko, 1983) and the paper (Leonenko et al., 2018).

rcosmo package

The current version of the **rcosmo** package can be installed from CRAN. A development release is available from GitHub (<https://github.com/frycast/rcosmo>).

The package offers various tools for

- Handling and manipulating of CMB radiation and other spherical data,
- Working with Hierarchical Equal Area isoLatitude Pixelation of a sphere (Healpix),
- Spherical geometry,
- Various statistical analysis of CMB and spherical data,
- Visualisation of HEALPix data.

Most of **rcosmo** features were developed for CMB, but it can also be used for other spherical data.

The package has more than 100 different functions. Figure 1 shows the core functions available in **rcosmo** and some typical data analysis flow sequences.

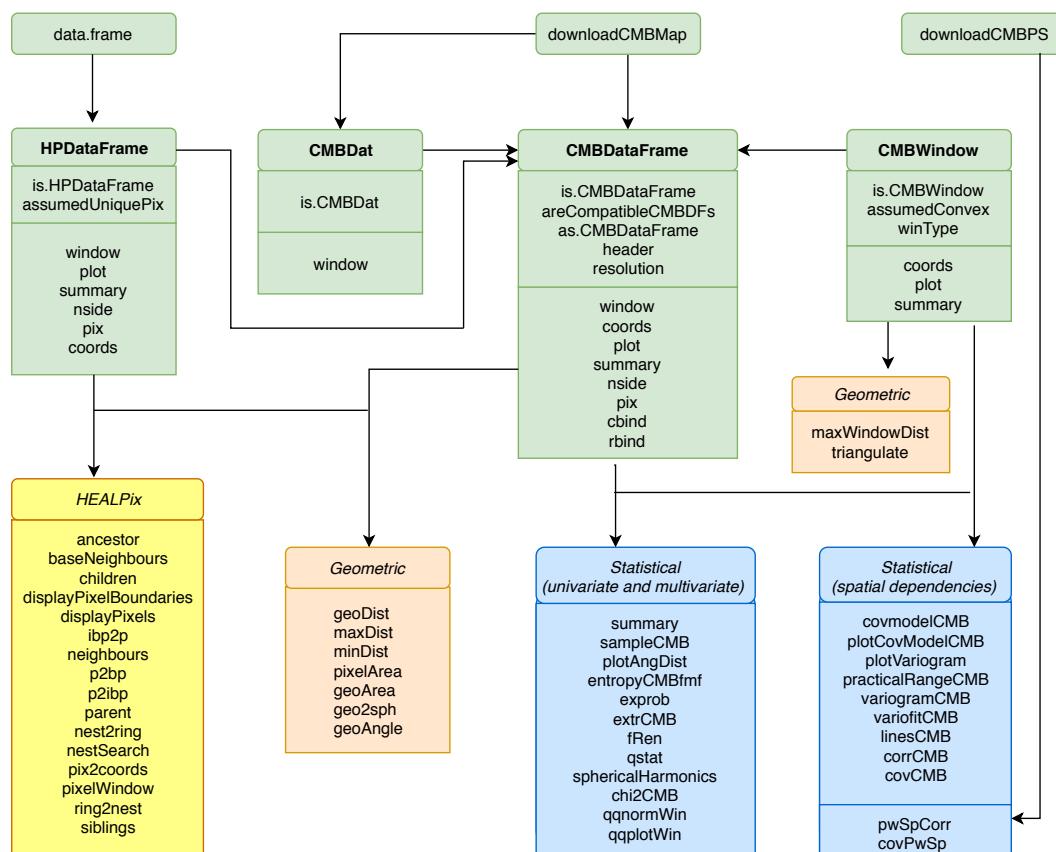


Figure 1: The flowchart of **rcosmo** main structure and core functions for 3 typical inputs: "data.frame" (general input), "downloadCMBMap" (conventional HEALPix format), and "downloadCMBPS" (power spectrum data).

Rather than attempting a systematic description of each functions, the remainder of this paper shows broad classes of methods implemented in **rcosmo** with particular examples of core functions. A reproducible version of the code in this paper for the current version of **rcosmo** is available in the folder "Research materials" from the website <https://sites.google.com/site/olenkoandriy/>.

Visualisation tools

Interactive visualisations of spherical data are a focus point of **rcosmo**. Standard Python and MATLAB tools for CMB and HEALPix visualization use the Mollweide projection of the unit sphere to the 2d plane. This is an equal-area projection, but it distorts spherical angles and distances. In contrast **rcosmo** employs the 'OpenGL' powered 3d visualization device system **rgl** for R to allow 3d interactive plots of data on the unit sphere.

The generic **plot** function produces interactive 3d vector graphics that may be easily exported to a HTML document. Some examples of using different plot functions are provided later in the paper. The results of **plot.CMBDataFrame** are given in Figures 5 and 6. For an example of using **plot.HPDataFrame** see Figure 15 and examples of using **plot.CMBWindow** are shown in Figures 5 and 7. For better visualization some figures produced by the R code in this paper were rotated and zoomed in before including in the article.

By default, the Planck colour scale is applied to CMB intensity data for plotting⁵. Additional features such as automatic plot legends, alternative colour scales, and greater configurability are planned for future releases. In addition, **rcosmo** provides a variety of 2d plot functionality to support visualisation of statistical analysis results and some additional 3d plot functionality for demonstrating HEALPix pixel properties.

rcosmo classes

Four R classes have been developed to support HEALPix data representation and analysis in the package **rcosmo**: "CMBDat", "CMBDataFrame", "HPDataFrame" and "CMBWindow". First three are main parent classes of objects to store spherical data. The class "CMBWindow" is used to choose observation windows.

- The function **CMBDat** creates objects of class "CMBDat". **CMBDat** objects are lists containing header information, other metadata and a data slot. Data slots may include standard information about CMB intensity (I), polarisation (Q, U), PMASK and TMASK. It also may contain a **mmap** object that points to the CMB map data table in a FITS file. As for standard data frames new data slots can be created to store other types of spherical data.
- The function **CMBDataFrame** creates objects of class "CMBDataFrame". These class is a special modification of "data.frame" that also carries metadata about, e.g., the HEALPix ordering scheme, coordinate system, and nside parameter (i.e., the resolution of the HEALPix grid that is being used). Each row of a **CMBDataFrame** is associated with a unique HEALPix pixel index.
- The class "HPDataFrame" is a type of "data.frame" designed to carry data located on the unit sphere. Unlike "CMBDataFrame", "HPDataFrame" may have repeated pixel indices. It allows to store multiple data points falling within a given pixel in different rows of **HPDataFrame** objects.
- The function **CMBWindow** creates objects of class "CMBWindow". These objects are polygons, spherical discs, or their compliments, unions and intersections.

As the main **rcosmo** data classes are special modifications of "data.frame" it means that spatial objects produced by **rcosmo** can be subsequently processed by other R packages/functions that work with standard data frames. The **rbind** and **cbind** generics that work with the "data.frame" class have been customised in **rcosmo** to preserve the validity of **CMBDataFrame** objects.

Getting data into rcosmo

In this section, we demonstrate how to import CMB data in the typical case of a full sky map stored as a FITS file. Such maps can be sourced from the NASA/IPAC Infrared Science Archive⁶.

⁵Colour scale is available here: <https://github.com/zonca/paperplots/tree/master/data>

⁶hosted by Caltech at the link http://irsa.ipac.caltech.edu/data/Planck/release_2/

The function `downloadCMBMap` can be used to download Planck CMB maps. One can specify the type of map ('COMMANDER', 'NILC', 'SEVEM' or 'SMICA'), the resolution ($N_{\text{side}} = 1024$ or 2048), and save it in a working directory with a specified file name.

The map 'COM_CMB_IQU-smica_1024_R2.02_full.fits' used in most of the examples in this paper is a FITS file of approximate size 200 megabytes. This map has the resolution $N_{\text{side}} = 1024$, so it contains $N_{\text{pix}} = 12 \times 1024^2 = 12582912$ pixels, each having its own intensity I , polarisation Q, U , temperature mask value $T_{\text{mask}} \in \{0, 1\}$ and polarisation mask value $P_{\text{mask}} \in \{0, 1\}$.

After downloading the map with `downloadCMBMap` and applying the function `CMBDataFrame`, we obtain an object of class "CMBDataFrame".

```
> library(rcosmo)
> filename <- "CMB_map_smica1024.fits"
> downloadCMBMap(foreground = "smica", nside = 1024, filename)
> sky<-CMBDataFrame(filename)
> str(sky)
Classes 'CMBDataFrame' and 'data.frame':      12582912 obs. of  1 variable:
 $ I: num -9.20e-05 -8.04e-05 -8.99e-05 -7.71e-05 -7.01e-05 ...
- attr(*, "nside")= int 1024
- attr(*, "ordering")= chr "nested"
- attr(*, "resolution")= num 10
...

```

An alternative to the above act of reading the entire map into memory is to take a random sample of points on the sphere. This is achieved without reading the entire map into R memory. Simple random sampling in `rcosmo` will be discussed further under the section on statistical functions.

```
> set.seed(0); s <- 2e6;
> cmb_sample <- CMBDataFrame(filename, sample.size = s, include.m = T, include.p = T)
> cmb_sample
A CMBDataFrame
# A tibble: 2,000,000 x 5
      I        Q        U    TMASK PMASK
      <dbl>     <dbl>     <dbl> <int> <int>
1 -0.0000771 5.45e-9 -0.000000718     0     0
2 -0.0000701 -7.10e-8 -0.000000730     0     0
...
10 -0.0000710 8.25e-8 -0.000000618     0     0
# ... with 1,999,990 more rows
```

Use of memory mapping

The standard library for reading data from FITS files is a collection of C and FORTRAN subroutines called 'CFITSIO'. In R, the package `FITSio` (Harris, 2016) is the only general FITS file reader that the authors are aware of. However, importing a full sky CMB map with approximately 12 million intensity samples from a FITS file using `FITSio` took too long when development of `rcosmo` began. We were able to reduce the necessary run time to under 4 seconds with the `rcosmo` function `CMBData`. `rcosmo` still substantially outperforms the last version of `FITSio`. In the following example we used the CMB map with 'SMICA' foreground separation and $N_{\text{side}} = 2048$ (having approximately 50 million intensity samples) to test it on a modern laptop⁷.

```
> filename1 <- "CMB_map_smica2048.fits"
> downloadCMBMap(foreground = "smica", nside = 2048, filename1)
> system.time(sky <- CMBDataFrame(filename1))
  user  system elapsed
  1.36    0.29   1.73

> system.time(fits <- FITSio:::readFITS(filename1))
  user  system elapsed
822.28   90.05  942.14
```

The approach used in `rcosmo` is based on a novel application of the `mmap` package by Jeffrey Ryan (Ryan, 2018). The package `mmap` is a highly optimised interface to 'POSIX mmap' and Windows

⁷Laptop specifications: Microsoft Surface Laptop with 7th Gen Intel Core m3 (i5) processor; 8GB LPDDR3 SDRAM (1866MHz)

'MapViewOfFile'. Using **mmap** in **rcosmo** required an update of **mmap** to support big-Endian byte order. The current version of **mmap** allows us to import data from a FITS binary table very efficiently, one row at a time, using a C struct data type. Ideally, for a typical **rcosmo** user, the details of using **mmap** are abstracted away while the user constructs and interacts with "CMBDataFrame" objects.

Another use of **mmap** in **rcosmo** concerns the elimination of the need to read a large full sky CMB map into R memory. Often it is unmanageable to read the entire contents of a FITS file. For example, it may not be possible to obtain sufficiently large blocks of contiguous memory from the operating system when importing more than a few hundred megabytes of data as a numeric vector. In **rcosmo**, integration of **mmap** allows one to maintain a C-style pointer at a particular byte-offset to the target binary file (e.g., the FITS file), so that data can be read into R memory on command from this offset. In the following example, only rows 1, 2, 4, 7 and 11 are read into memory from the file.

```
> v <- c(1,2,4,7,11)
> sky <- CMBDataFrame(filename, spix = v, include.p = T, include.m = T,
  coords = "spherical")
> sky
A CMBDataFrame
# A tibble: 5 x 7
  theta    phi      I      Q          U    TMASK PMASK
  <dbl> <dbl> <dbl> <dbl> <dbl> <int> <int>
1 1.57 0.785 -0.0000920 6.47e-8 -0.000000657     0     0
2 1.57 0.786 -0.0000804 -9.19e-9 -0.000000694     0     0
3 1.57 0.785 -0.0000771 5.45e-9 -0.000000718     0     0
4 1.57 0.786 -0.0000663 -5.81e-8 -0.000000751     0     0
5 1.57 0.783 -0.0000836 7.60e-8 -0.000000697     0     0

> pix(sky)
[1] 1 2 4 7 11
```

The next section discusses the HEALPix data structure. HEALPix ordering schemes can be used to map coordinates on the sphere to row indices in a FITS binary table. Combining this feature with the technique in the above example allows **rcosmo** to efficiently import random samples of data from a variety of geometric regions of the sphere without ever having to import the entire CMB map. This is particularly useful on larger maps and will become increasingly important in future as advances in cosmology allow for higher resolution CMB maps to be produced.

Introduction to HEALPix

Present generation Cosmic Microwave Background experiments produce data with up to 5 arcminutes resolution on the sphere. For a full-sky map, this amounts to approximately 50 million pixels, each describing distinct location, intensity, polarisation and other attributes. The statistical analysis of such massive datasets, and associated discretisation of functions on the sphere, can involve prohibitive computational complexity and non-adequate sampling in the absence of an appropriate data structure. The Hierarchical Equal Area isoLatitude Pixelation is a geometric structure designed to meet this demand using a self-similar refinable mesh. It is currently the most widely used pixelation for storing and analysing CMB data (Gorski et al., 2005).

The package **rcosmo** provides various tools to visualize and work with the HEALPix structure.

HEALPix initially divides the sphere into 12 equiareal *base pixels*. To visualise these with **rcosmo**, we can first generate a **CMBDataFrame** at some low resolution (e.g, $N_{\text{side}} = 64$) and then take three separate window subsets in the pixels that we intend to colour, as shown in Figure ??). Note that, while all HEALPix pixels are 4-sided, their edges are not geodesics, i.e., they are not spherical quadrilaterals (Calabretta and Roukema, 2007).

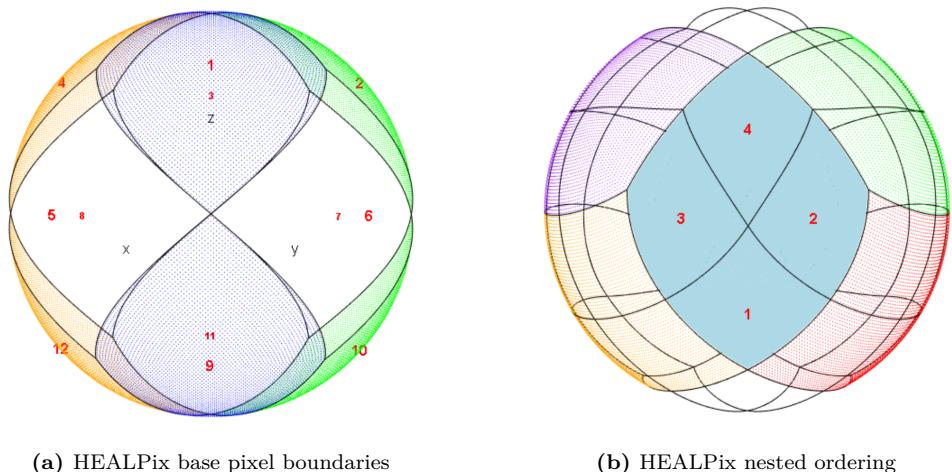
```
> ns <- 64; rand <- rnorm(12 * ns ^ 2)
> cmbdf <- CMBDataFrame(nside = ns, I = rand, ordering = "nested")
> w1 <- window(cmbdf, in.pixels = c(1,9))
> w2 <- window(cmbdf, in.pixels = c(2,10))
> w3 <- window(cmbdf, in.pixels = c(4,12))
> plot(w1, col = "blue", back.col = "white", xlab = '', ylab = '', zlab = '')
> plot(w2, col = "green", add = TRUE)
```

```
> plot(w3, col = "orange", add = TRUE)
> displayPixelBoundaries(nside = 1, ordering = "nested", incl.labels = 1:12, col = "red")
```

Each of the 12 base pixels can be further subdivided into 4 equiareal 4-sided pixels. For a demonstration, we can create another window subset based on a higher resolution `CMBDataFrame` and display the outputs in Figure ??.

```
> ns <- 256; rand <- rnorm(12 * ns ^ 2)
> w21 <- window(CMBDataFrame(nside = ns, I = rand, ordering = "nested"), in.pixels = 1)
> plot(w21, col = "light blue", back.col = "white", add = TRUE, size = 1.2)
> displayPixelBoundaries(nside=2, ordering="nested", incl.labels=c(1,2,3,4), col = "black")
> plot(window(cmbdf, in.pixels = 2), col = "green", add = TRUE)
> plot(window(cmbdf, in.pixels = 4), col = "purple", add = TRUE)
> plot(window(cmbdf, in.pixels = 5), col = "orange", add = TRUE)
> plot(window(cmbdf, in.pixels = 6), col = "red", add = TRUE)
```

This process of subdivision can be repeated until a desired resolution is achieved. At the required resolution, the number of edge segments per base pixel edge is referred to as the N_{side} parameter, and satisfies $N_{\text{pix}} = 12N_{\text{side}}^2$, where we use N_{pix} to denote the total number of pixels.



(a) HEALPix base pixel boundaries

(b) HEALPix nested ordering

Figure 2: (a): HEALPix base pixel boundary visualisation. The 12 base pixels are labelled 1 to 12; (b): HEALPix nested ordering visualisation at $N_{\text{side}} = 2$. Pixels 1, 2, 3 and 4 (labelled) all fall within base pixel 1 (coloured solid).

At a given N_{side} , the HEALPix representation provides a bijection from the first $12N_{\text{side}}^2$ natural numbers P to a set of locations L on the unit sphere. We refer to P as the set of *pixel indices* and L as the set of *pixel centers*. For assigning the pixel indices to the pixel centers there are two approaches, known respectively as the "ring" and "nested" ordering schemes. The nested scheme is demonstrated with the numbering in Figure ?? . The ring scheme is demonstrated in Figure 3.

```
> cmbdf <- CMBDataFrame(nside = 8, ordering = "ring")
> plot(cmbdf, type = 'l', col = "black", back.col = "white")
> tolabel <- c(1,100:107,768)
> plot(cmbdf[tolabel,], labels = tolabel, col = "red", add = TRUE)
```

Regardless of the choice of ordering scheme, all HEALPix pixel centers lie on $4N_{\text{side}} - 1$ rings of constant latitude. This feature facilitates fast discrete spherical harmonic transforms, since the associated Legendre functions need only be evaluated once per isolatitude ring of pixel centers (Gorski et al., 2005).

Many tasks gain or lose efficiency with the choice of ordering scheme. For example, nearest neighbour searches are best conducted in the nested scheme (Gorski et al., 2005). As such, every object of class "`CMBDataFrame`" or "`HPDataFrame`" has an attribute named `ordering` to indicate which of the two schemes is being used. This allows `rcosmo` functions to choose the most efficient scheme for each task, performing any necessary conversions with the internal functions `nest2ring` and `ring2nest`.

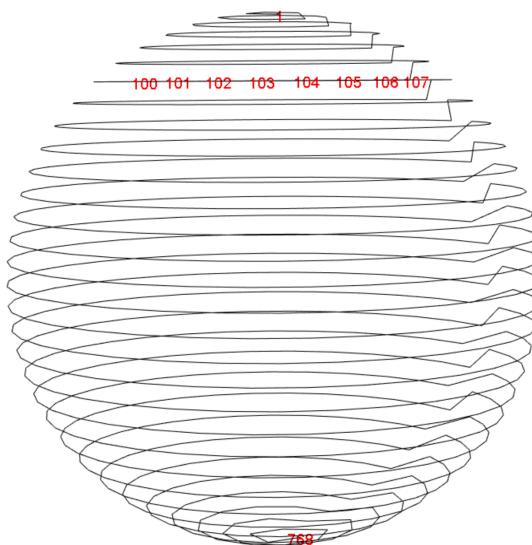


Figure 3: HEALPix ring ordering scheme visualisation. The black line traces in order through pixel centers from 1 to $N_{\text{pix}} = 768$. The locations of pixels 100 to 107 are labelled.

HEALPix functions

For working directly with HEALPix properties there are a number of **rcosmo** functions. Some core functions are shown in Figure 1 in the yellow colour, other are internal and some were exported. Broadly, there are the following categories:

- Working with ordering schemes,
- Navigating the nested ordering hierarchy,
- Geometric functions involving pixel indices,
- Visualising the HEALPix structure.

The main ordering functions include converting between two ordering schemes and getting information about a type of ordering (**ordering** generic and internals **nest2ring** and **ring2nest**). For example, the **ordering** generic function is useful for getting and setting the ordering attribute of a **CMBDataFrame** or **HPDataFrame**.

```
> sky <- CMBDataFrame(nside = 2, ordering = "ring"); ordering(sky)
[1] "ring"
> ordering(sky) <- "nested"; ordering(sky)
[1] "nested"
```

rcosmo functions for navigating the HEALPix structure provide various tools to investigate local neighbourhoods of specific pixels and relative positions of pixels at different levels of the nested ordering hierarchy, see, for example, **ancestor**, **pixelWindow**, **neighbours**, etc. Since the nested ordering is self-similar, many of these functions are resolution independent.

For example, the k^{th} ancestor of a pixel index p at resolution $j := \log_2(N_{\text{side}})$ is the pixel index p_a to which p belongs, k steps up the hierarchy (i.e., at resolution $j - k$). It turns out that p_a is a function $p_a = f(p, k)$ that is independent of j . The following example produces the ancestors of pixel $p = 10^3$, for $k = 1, 2, \dots, 5$, using the internal function **ancestor**.

```
> ancestor(1e3, 1:5)
[1] 250 63 16 4 1
```

A function that is not resolution independent is **pixelWindow**. In the following code, **pixelWindow** retrieves all pixels at resolution $j_2 = 5$ that lie within pixel $p = 1$, specified at resolution $j_1 = 1$. The result is shown in Figure 4.

```
p <- 1; j1 <- 1; j2 <- 5
P <- pixelWindow(j1 = j1, j2 = j2, pix.j1 = p)
displayPixels(spix = P, j = j2, plot.j = j2)
```

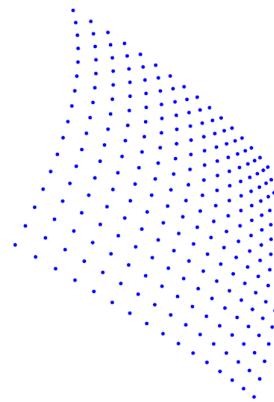


Figure 4: All pixel centers (at resolution 5), within pixel 1 (at resolution 1)

The group of **rccosmo** functions that includes `pix2coords`, `pixelArea`, `nestSearch`, etc., computes spherical geometric properties in relation to by pixel indices. For example, the `nestSearch` function searches a pixel closest to a point in 3d space. It uses an algorithm that achieves a high level of efficiency using the nested hierarchy. A comparison, via [microbenchmark](#) (Mersmann, 2018), with a basic linear search algorithm which we call `geoDistSearch` reveals the following.

```
> library(microbenchmark)
> geoDistSearch <- function(target, nside, spix) {
  xyz <- pix2coords(nside = nside, spix = spix)
  dists <- geoDist(xyz, target)
  return(xyz[which.min(dists),])}

> t <- data.frame(x = 0.6, y = 0.8, z = 0)
> nside <- 16; p <- 1:(12*(nside)^2)
> mb <- microbenchmark::microbenchmark(
  geoDistSearch(target = t, nside = nside, spix = p),
  nestSearch(target = t, nside = nside))
> summary(mb)$median[1]/summary(mb)$median[2]
[1] 283.9834
```

From the above, it was observed that `nestSearch` was over 200 times faster than `geoDistSearch` at finding the closest pixel center at $N_{\text{side}} = 16$ to the point $(x, y, z) = (0.6, 0.8, 0)$. With `nestSearch`, the closest pixel to a target point is found by checking only $12 + 4 \log_2(N_{\text{side}})$ pixels rather than $12N_{\text{side}}^2$. When $N_{\text{side}} = 2048$, only $12 + 4 \log_2(2048) = 56$ pixels must be checked from over 50 million.

Subsetting and combining spherical regions

rccosmo functions for selecting and visualizing spherical regions can be broadly divided in the following groups:

- Creating basic `CMBWindow` objects (polygons or spherical discs),
- Combining different sub-regions by using compliments, unions and intersections to create a new `CMBWindow` object,
- Plotting a region with boundary and inside points,
- Extracting data from a given `CMBDataFrame` restricted to a `CMBWindow` region.

Class `CMBWindow` is designed to carry geometrical information describing the interior or exterior of spherical figures (polygons, spherical discs (caps), and their complements, unions and intersections). The polygons can be non-convex, though `CMBWindow` carries a boolean attribute `assumedConvex` that should be set to `TRUE`, if the polygon is known in advance to be convex. In this case special methods that decrease computation times are applied.

A `CMBWindow` object can be created using the `CMBWindow` function. The code below illustrates the creation of two `CMBWindows` that correspond to the Dragon and Scorpion constellations. Files

of constellation boundaries⁸ include coordinates of spherical polygons vertices that correspond to each constellation. The function `hms2deg` converts celestial coordinates (hours, minutes, seconds) to the degrees format. Then "phi" and "theta" columns of the `data.frame` `CB` are used to create `CMBWindow` objects. To inspect these `CMBWindow` objects using interactive 3D graphics, we can pass them to the generic plot function. Below, we also plot the CMB map as a background. The resulting plot is displayed in Figure 5.

```
> download.file("https://www.iau.org/static/public/constellations/txt/dra.txt",
  "bound1.txt")
> x1 <- readLines("bound1.txt")
> x1 <- gsub("\\|", " ", x1)
> Constellation_Boundary1 <- read.table(text = x1,col.names=c("H","M","S","D","Con_N"))
> download.file("https://www.iau.org/static/public/constellations/txt/sgr.txt",
  "bound2.txt")
> x2 <- readLines("bound2.txt")
> x2 <- gsub("\\|", " ", x2)
> Constellation_Boundary2 <- read.table(text = x2,col.names=c("H","M","S","D","Con_N"))

> CB0 <- Constellation_Boundary1
> deg <- celestial::hms2deg(CB0[,1],CB0[,2],CB0[,3])
> CB1 <- data.frame(pi*deg/180, pi*CB0[,4]/90)
> colnames(CB1) <- c("phi", "theta")
> polygon1 <- CMBWindow(phi = CB1$phi, theta = CB1$theta)
> plot(cmb_sample, back.col = "white")
> plot(polygon1, lwd=2)
```

After repeating the steps above for the Scorpion constellation we obtain and the second `CMBWindow` object `polygon2` in Figure 5.

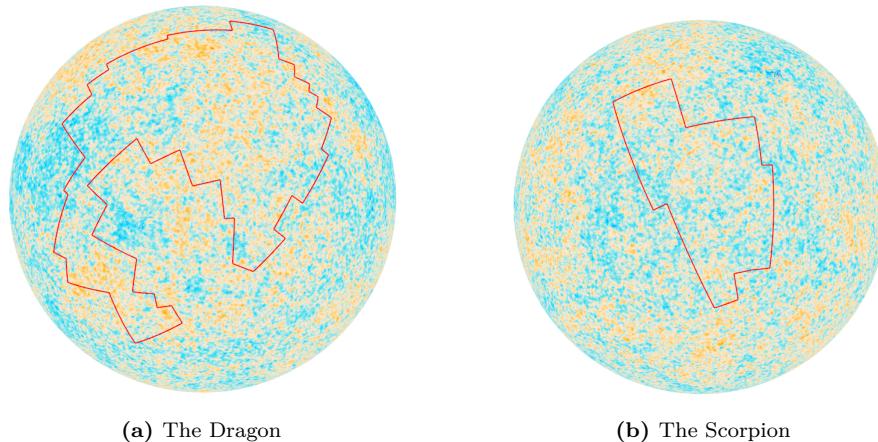


Figure 5: Boundary visualisation of polygon `CMBWindow` objects, plotted against 10^5 CMB intensities.

Note that for the `CMBWindow` polygons defined above, entire polygons lie within any one hemisphere of S_2 . To obtain `CMBWindow` objects that occupy more than one hemisphere, we can specify a polygon or disc exterior (complement in S_2) using the `set.minus = TRUE` parameter. For example, the following command gives the exterior of a spherical cap with a base radius 0.5.

```
> d.exterior <- CMBWindow(theta = pi/2, phi = 0, r = 0.5, set.minus = TRUE)
```

To specify more complicated regions, we can combine multiple `CMBWindow` objects into a `list`. For example, the following command results in the list containing `d.exterior` and the interior of a spherical disc (cap) of base radius 1 (disc's radius is computed on the sphere surface), which is a spherical segment shown in Figure 6.

```
> wins <- list(d.exterior, CMBWindow(theta = pi/2, phi = 0, r = 1))
```

By passing `CMBWindow` objects to the `window` function, one can extract data from a `CMBDataFrame` or `mmap` object. Below, using the above spherical window `wins` the `CMBDataFrame` named `sky.annulus` is created and plotted in Figure 6.

⁸available at <https://www.iau.org/public/themes/constellations/>

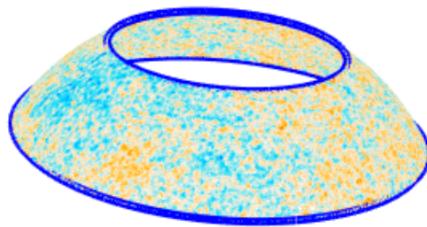


Figure 6: CMB intensity data extracted from an `CMBDataFrame` object by the `window` function.

```
> df <- CMBDataFrame(filename)
> sky.annulus <- window(df, new.window = wins)
> plot(sky.annulus, back.col = "white")
> plot(wins[[1]], lwd = 5, col="blue"); plot(wins[[2]], lwd = 5, col="blue")
```

Spherical geometry functions

Several basic tools for spherical geometry are implemented in `rcosmo`:

- Converting between different coordinate systems on the sphere,
- Computing geodesic distances between points and windows,
- Calculating spherical angles,
- Computing areas of spherical figures,
- Triangulating spherical polygons.

The currently implemented core geometric functions are shown in Figure 1 in the orange colour. Some other spherical geometric tools are specified for the HEALPix representation or `CMBWindows` and are shown in the green colour.

For example, the functions `geoArea` computes the area of a figure on the unit sphere that is encompassed by its pixels.

```
> geoArea(sky.annulus)
[1] 2.11917
```

Another example is the function `maxDist` that computes the maximum geodesic distance either between all points in a `data.frame` pairwise, or between all points in a `data.frame` and a target point.

```
> p <- c(0,0,1)
> maxDist(sky.annulus, p)
[1] 2.570114
```

Various geometric problems require triangulations of spherical polygons. For a polygonal `CMBWindow` the function `triangulate` produces a set of spherical triangles with pairwise disjoint interiors and the union equals to the original polygon. For example, Figure 7 shows a triangulation of the Dragon constellation spherical polygon.

```
> win1 <- triangulate(polygon2)
> for (i in 1:11) {plot(win1[[i]], col=i)}
```

Statistical functions

In this section we overview core statistical functions implemented in `rcosmo`. The package provides various tools for statistical analysis of spherical data that can be broadly divided in the following types:

- Spherical random sampling,
- Univariate spherical statistics and plots,
- Multivariate statistics for data from different `CMBWindows`,

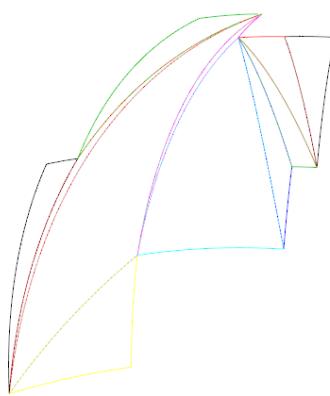


Figure 7: Triangulation of a spherical polygon.

- Measures of spatial dependencies.

The main currently implemented statistical functions are shown in Figure 1 in the blue colour. Below we provide few examples of functions for each type.

Random sampling

An immediate advantage of equiareal HEALPix pixel sizes is that simple random sampling is not regionally dependent (Gorski et al., 2005). That is, a simple random sample of pixel indices produces an approximately uniform sample of locations on the sphere.

To get a simple random sample from a `CMBDataFrame` one can use the function `sampleCMB`. This function returns a `CMBDataFrame` which size equals to the function's parameter `sample.size`. This new object has rows that comprise a simple random sample of the rows from the input CMB-`DataFrame`.

```
> set.seed(0)
> sampleCMB(df, sample.size = 3)
A CMBDataFrame
# A tibble: 3 x 1
      I
      <dbl>
1 -0.0000198
2 -0.000307
3 -0.0000915
```

Univariate spherical statistics and plots

There are several methods in `rcosmo` for statistical analysis and visualisation of CMB temperature intensity data. For example, function `summary` produces a `CMBDataFrame` summary that includes information about window's type and area, total area covered by observations, and the main statistics for the intensity data in the spherical window.

```
> summary(sky.annulus)
===== CMBDataFrame Object =====
Number of CMBWindows: 2
+-----+
|           |
| Window type: minus.disc   |
| Window area: 11.7972     |
|           |
+-----+
+-----+
|           |
| Window type: disc        |
| Window area: 2.8884       |
|           |
```

```

|           |
+-----+
METHOD  = 'smica'          / Separation method
Total area covered by all pixels: 2.11917
=====
Intensity quartiles
Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
-7.609e-04 -6.908e-05 -6.208e-07 -1.487e-06  6.737e-05  7.697e-04
=====
```

The function `entropyCMB` returns an estimated entropy for specified column intensities and `CMBWindow`.

```
> entropyCMB(cmbdf = df, win = d.exterior, intensities = "I")
[1] 2.13523
```

The first Minkowski functional `fmf` returns an area of the spherical region, where the intensities are above of the specified threshold level α .

```
> fmf(cmbdf = sky.annulus, alpha = 0, intensities = "I")
[1] 1.054269
```

`fRen` computes values of the sample fractal scaling exponent on the grid of N uniformly spaced points in the interval $[q.\min, q.\max]$. The scaling exponent describes fractal properties of random fields and can be used for testing departures from Gaussianity, see (Leonenko and Shieh, 2013). For example, Figure 8 shows that the sample scaling exponent is an approximate strait line for the data in `CMBWindow sky.annulus`. Thus, there is not enough evidence for substantial multifractality of the random field based on the data in this `CMBWindow`. More details can be found in the paper (Leonenko et al., 2020).

```
> Tq <- fRen(cmbdf = sky.annulus, q.min = 1.01, q.max = 10, N = 20, intensities = "I")
> plot(Tq[,1], Tq[,2], ylab = expression(T[q]), xlab = "q", main = "Sample fractal
scaling exponent", pch = 20, col = "blue")
```

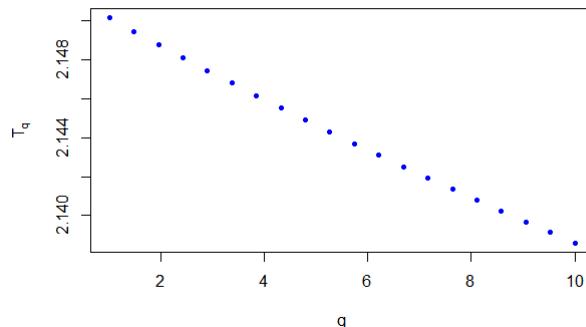


Figure 8: Sample fractal scaling exponent of `sky.annulus` on [1.01,10].

The function `plotAngDis` helps to visualise the marginal distributions of temperature intensities versus θ and ϕ angles. It produces scatterplots and barplots of the corresponding means computed over bins, see Figure 9.

```
> df1 <- sampleCMB(df, sample.size = 100000)
> cmbdf.win <- window(df1, new.window = polygon2)
> plotAngDis(cmbdf.win, intensities = "I")
```

Multivariate statistics for data from different CMBWindows.

There are several `rcommo` functions for comparison of data from two or more `CMBWindows`. For example, the function `qqplotWin` is a modification of the standard `qqplot` to produces a QQ plot of quantiles of observations in two `CMBWindows` against each other for a specified `CMBDataFrame` column.

The example below shows that the distributions of temperatures in the Dragon and Scorpion constellations are similar.

```
> qqplotWin(df1, polygon1, polygon2)
```

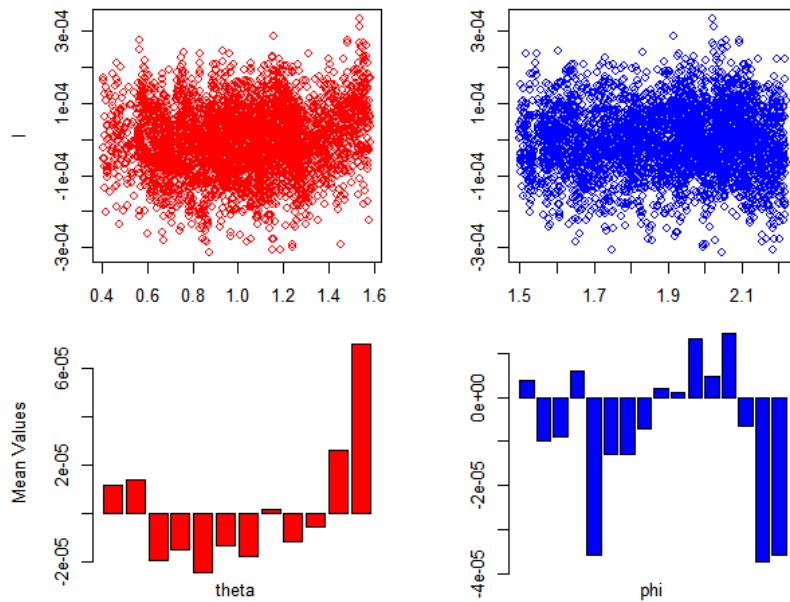


Figure 9: Distributions of temperature versus θ and ϕ angles for the Scorpion constellation region.

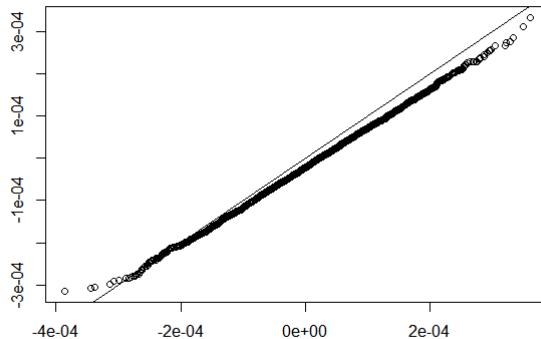


Figure 10: QQ plot of observations in the Dragon vs Scorpion constellation regions.

The function `qstatq` can be used to measure spatial stratified heterogeneity in a list of `CMBWindows`. It takes values in $[0, 1]$, where 0 corresponds to no spatial stratified heterogeneity, 1 means a perfect heterogeneity case. For example, the results below shows that there is not enough evidence for spatial stratified heterogeneity, i.e. the value of the temperature intensities are not different in these two `CMBWindows`.

```
> lw <- list(polygon1, polygon2)
> qstat(df1, lw)
[1] 0.01089514
```

Investigating spatial dependencies

This section presents some of `rkosmo` tools for the analysis of spatial dependencies in spherical data.

As the geodesic and Euclidean distances are different, covariance functions on \mathbb{R}^3 can not be used directly for S^2 . The package implemented several parametric models of covariance functions (2) on the sphere, see theoretical foundations in (Gneiting, 2013). `rkosmo` uses the package `geoR` and extends its list of general spatial models and some functions to the spherical case. Currently available choices of covariance models are `matern`, `exponential`, `spherical`, `powered.exponential`, `cauchy`, `gencauchy`, `pure.nugget`, `askey`, `c2wendland`, `c4wendland`, `sinepower`, and `multiquadric`. The default option is `matern`.

The function `covmodelCMB` computes values of theoretical covariance functions given the sepa-

ration distance of locations. The function returns the value of the covariance $\Gamma(h)$ at the geodesic distance h . The covariance model uses the general form $\Gamma(h) = \sigma^2 \cdot \rho(h/\psi)$, where the variance σ^2 and the range ψ are vertical and horizontal scaling parameters respectively.

For example, the following command computes the value of the Askey covariance function with the parameters $\sigma^2 = 1$, $\psi = \pi$, and $\kappa = 4$ at the geodesic distance $h = \pi/4$.

```
> covmodelCMB(pi/4, cov.pars = c(1, pi), kappa = 4, cov.model = "askey" )
[1] 0.3164062
```

The command `plotcovmodelCMBplot` is designed to produce quick plots of theoretical covariance functions. The result for the Askey covariance function is shown in Figure 11.

```
> plotcovmodelCMB("askey", phi = pi/4, to = pi/2, kappa = 4)
```

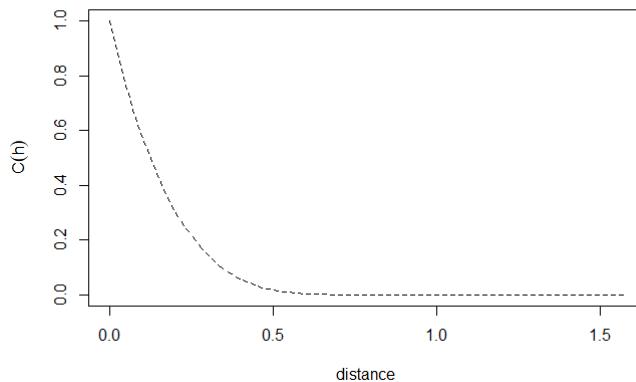


Figure 11: Plot of the Askey covariance function for the parameters $\sigma^2 = 1$, $\psi = \pi/4$, and $\kappa = 4$.

If a random fields is isotropic its covariance function depends only on a geodesic distance between locations. In this case the function `covCMB` can be used to compute an empirical covariance function for intensity data in a `CMBDataFrame` or `data.frame`. Output is given up to a maximal geodesic distance `max.dist`, which can be chosen between 0 and π (by default equals π).

```
> df1 <- sampleCMB(df, sample.size = 100000)
> Cov <- covCMB(df1, max.dist = 0.03, num.bins = 10)
> Cov$v
[1] 1.041607e-08 6.955709e-09 4.305906e-09 3.180806e-09
[5] 2.675364e-09 2.436718e-09 2.407982e-09 2.316794e-09
[9] 2.319271e-09 2.308250e-09 2.263268e-09
```

Obtained estimated covariance values can be visualised using the command

```
> plot(Cov)
```

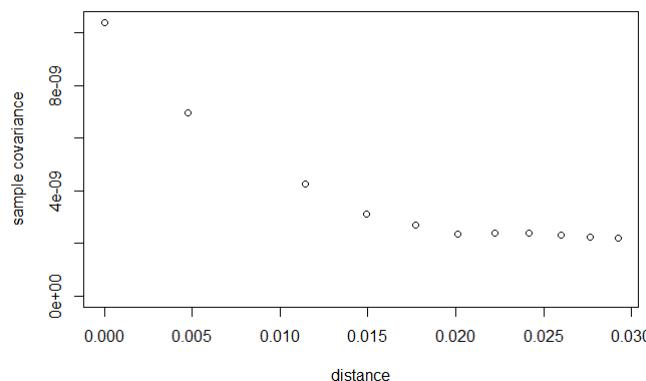


Figure 12: Plot of the empirical covariance function for `max.dist` = 0.03.

The function `variofitCMB` estimates parameters of variogram models (see equation (3) for the link between covariance and variogram functions) by fitting a parametric model from the list

`covmodelCMB` to a sample variogram estimated by the function `variogramCMB`. This function is built on and extends `variofit` from the package `geoR` to specific `rcosmo` covariance models on the sphere.

In the example below the Matern variogram is fitted to the empirical variogram on the interval $[0, 0.1]$ using the ordinary least squares method, see Figure 13.

```
> varcmb <- variogramCMB(df1, max.dist = 0.1, num.bins = 30)
> ols <- variofitCMB(varcmb, fix.nug=FALSE, wei="equal", cov.model= "matern")
> plot(varcmb)
> lines(ols, lty=2)
```

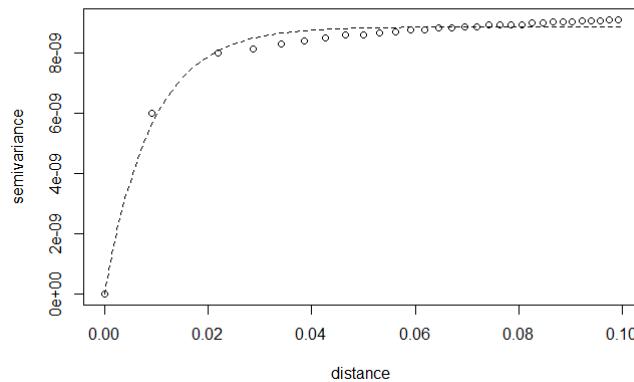


Figure 13: Plots of the empirical and fitted variograms.

The package also has tools to work with angular power spectra of spherical random fields. The CMB power spectrum data are freely available from the section "Cosmology products" of the Planck Legacy Archive⁹. They can be easily downloaded in a ready-to-use `rcosmo` format by the function `downloadCMBPS`.

The function `covPwSp` uses values of an angular power spectra to provide a covariance estimate by equation (2). As the argument of the covariance function in equation (2) is $\cos \Theta$ the following code uses the inverse transformation `acos` to plot the covariance estimate as a function of angular distances in Figure 14.

```
> COM_PowerSpectra <- downloadCMBPS(link=1)
> Cov_est <- covPwSp(COM_PowerSpectra[,1:2], 2000)
> plot(acos(Cov_est[,1]), Cov_est[,2], type ="l", xlab ="angular distance",
      ylab ="Estimated Covariance")
```

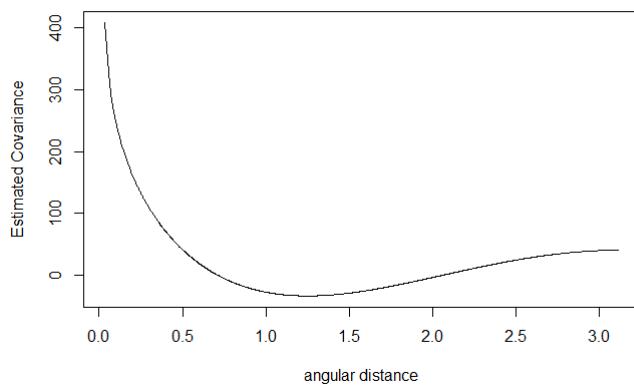


Figure 14: Plot the covariance estimate using CMB power spectrum.

Converting other spherical data to HEALPix format

While the HEALPix is the main representation in cosmological applications there are numerous spherical data, for example, in geosciences, that use different coordinate systems and spherical

⁹hosted at the link <https://pla.esac.esa.int/pla/>

formats. This example shows how non-HEALPix spherical data can be converted to the HEALPix format for **rcosmo** analysis.

A **HPDataFrame** is suitable for storing data that is located on a sphere, but has not been pre-processed to suit HEALPix structured storage. There are various ways to assign HEALPix pixel indices to the rows of a **HPDataFrame**. This example presents the way when a desired resolution is specified in advance. Then the **HPDataFrame** constructor automatically assigns pixel indices based on coordinates provided. A row is assigned the pixel index of its closest pixel center. **rcosmo** also has the option of automatic computing of a required resolution for given data to assign data locations to unique pixels.

As an example we use the database with the latitude and longitude of over 13 thousand world's large cities and towns available from the World Cities Database¹⁰. First, cities latitudes and longitudes in degrees are converted to spherical coordinates (θ, ϕ) in radians. Then, we create and visualize **HPDataFrame** at the resolution `nside = 1024`, see Figure 15.

```
> worldcities <- read.csv("worldcities.csv")
> sph <- geo2sph(data.frame(lon = pi/180*worldcities$lng, lat = pi/180*worldcities$lat))
> df1 <- data.frame(phi = sph$phi, theta = sph$theta, I = rep(1,nrow(sph)))
> hp <- HPDataFrame(df1, auto.spix = TRUE, nside = 1024)
> plot(hp, size = 3, col = "darkgreen", back.col = "white")
```



Figure 15: Plot of world's large cities and towns at the resolution `nside = 1024`.

Some other examples of converting directional and star-shaped data into **rcosmo** formats are given in (Fryer and Olenko, 2019).

Summary and future directions

This article introduces the package **rcosmo** for analysis of CMB, HEALPix and other spherical data. The package integrates the HEALPix representation and various spherical geometric and statistical methods in a convenient unified framework. It opens efficient handling and analysis of HEALPix and CMB data to the R statistical community. **rcosmo** also introduces several new spherical statistical models and methods that were not available in R before. The package can also be very useful for researchers working in geosciences.

There are several possible extensions that would be useful to the package. Some of them include:

- integrating with available Python and C++ HEALPix software,
- including new spherical statistical models and methods,
- further development of spherical spectral and multifractal methods,
- adding new visualisation tools,
- improved use of memory mapping to use on extremely high resolution images.

¹⁰hosted at the link <https://simplemaps.com/data/world-cities>

Acknowledgements

This research was partially supported under the Australian Research Council's Discovery Projects funding scheme (project number DP160101366). We are thankful to the anonymous referee for suggestions that helped to improve the paper. We also would like to thank V.V. Anh, P.Broadbridge, N.Leonenko, I.Sloan, and Y.Wang for their comments on early drafts of **rcosmo** and discussions of CMB and spherical statistical methods, and J.Ryan for developing and extending the **mmap** package.

Bibliography

- R. Adam, P. Ade, N. Aghanim, Y. Akrami, M. Alves, F. Argüeso, M. Arnaud, F. Arroja, M. Ashdown, J. Aumont, et al. Planck 2015 results-I. Overview of products and scientific results. *Astronomy and Astrophysics*, 594(A1):1–38, 2016. URL <https://doi.org/10.1051/0004-6361/201527101>. [p]
- D. Adler, D. Murdoch, and others. *rgl: 3D Visualization Using OpenGL*, 2018. URL <https://CRAN.R-project.org/package=rgl>. R package version 0.99.16. [p]
- C. L. Bennett, M. Halpern, G. Hinshaw, N. Jarosik, A. Kogut, M. Limon, S. Meyer, L. Page, D. Spergel, G. Tucker, et al. First-year Wilkinson Microwave Anisotropy Probe (WMAP) observations: Preliminary maps and basic results. *The Astrophysical Journal Supplement Series*, 148(1):1–27, 2003. URL <https://doi.org/10.1086/377253>. [p]
- R. S. Bivand, E. Pebesma, and V. Gomez-Rubio. *Applied spatial data analysis with R, Second edition*. Springer, NY, 2013. ISBN 9781461476184. [p]
- A. W. Bowman and A. Azzalini. *R package sm: nonparametric smoothing methods (version 2.2-5.6)*. University of Glasgow, UK and Università di Padova, Italia, 2018. URL <http://www.stats.gla.ac.uk/~adrian/sm/>. [p]
- S. D. J. Brown, R. A. Collins, S. Boyer, M.-C. Lefort, J. Malumbres-Olarre, C. J. Vink, and R. H. Cruickshank. SPIDER: an R package for the analysis of species identity and evolution, with particular reference to DNA barcoding. *Molecular Ecology Resources*, 12:562–565, 2012. URL <https://doi.org/10.1111/j.1755-0998.2011.03108.x>. [p]
- M. R. Calabretta and B. F. Roukema. Mapping on the healpix grid. *Monthly Notices of the Royal Astronomical Society*, 381(2):865–872, 2007. URL <https://doi.org/10.1111/j.1365-2966.2007.12297.x>. [p]
- R. Durrer. The cosmic microwave background: the history of its experimental investigation and its significance for cosmology. *Classical and Quantum Gravity*, 32(12), 2015. URL <https://doi.org/10.1088/0264-9381/32/12/124007>. [p]
- G. Efstathiou, J. Bond, and S. White. COBE background radiation anisotropies and large-scale structure in the universe. *Monthly Notices of the Royal Astronomical Society*, 258(1):1P–6P, 1992. URL <https://doi.org/10.1093/mnras/258.1.1P>. [p]
- A. Felicísimo, J. C. R. Cuetos, M. E. P. García, A. Cuartero, and P. G. Rodriguez. *VecStatGraphs3D: Vector analysis using graphical and analytical methods in 3D*, 2014. URL <https://CRAN.R-project.org/package=VecStatGraphs3D>. R package version 1.6. [p]
- J. J. Fernández-Durán and M. M. Gregorio-Domínguez. CircNNTSR: An R package for the statistical analysis of circular, multivariate circular, and spherical data using nonnegative trigonometric sums. *Journal of Statistical Software*, 70(6):1–19, 2016. URL <https://doi.org/10.18637/jss.v070.i06>. [p]
- N. I. Fisher, T. Lewis, and B. J. J. Embleton. *Statistical Analysis of Spherical Data*. Cambridge University Press, Cambridge, 1987. ISBN 978-0521456999. [p]
- D. Fryer and A. Olenko. Spherical data handling and analysis with R package rcosmo. In H. Nguyen, editor, *Statistics and Data Science. RSSDS 2019*, pages 211–225. Springer, Singapore, 2019. URL https://doi.org/10.1007/978-981-15-1960-4_15. [p]
- D. Fryer, A. Olenko, M. Li, and Y. Wang. *rcosmo: Cosmic Microwave Background Data Analysis*, 2020. URL <https://CRAN.R-project.org/package=rcosmo>. R package version 1.1.2. [p]

- T. Gneiting. Strictly and non-strictly positive definite functions on spheres. *Bernoulli*, 19(4):1327–1349, 09 2013. URL <https://doi.org/10.3150/12-BEJSP06>. [p]
- K. M. Gorski, E. Hivon, A. Banday, B. D. Wandelt, F. K. Hansen, M. Reinecke, and M. Bartelmann. Healpix: a framework for high-resolution discretization and fast analysis of data distributed on the sphere. *The Astrophysical Journal*, 622(2):759–771, 2005. URL <https://doi.org/10.1086/427976>. [p]
- A. Harris. *cosmoFns: Functions for cosmological distances, times, luminosities, etc.*, 2012. URL <https://CRAN.R-project.org/package=cosmoFns>. R package version 1.0-1. [p]
- A. Harris. *FITSio: FITS (Flexible Image Transport System) Utilities*, 2016. URL <https://CRAN.R-project.org/package=FITSio>. R package version 2.1-0. [p]
- R. J. Hijmans. *geosphere: Spherical Trigonometry*, 2017. URL <https://CRAN.R-project.org/package=geosphere>. R package version 1.5-7. [p]
- G. Hurier, M. Douspis, N. Aghanim, E. Pointecouteau, J. Diego, and J. Macias-Perez. Cosmological constraints from the observed angular cross-power spectrum between Sunyaev-Zel'dovich and x-ray surveys. *Astronomy and Astrophysics*, 576(A90):1–11, 2015. URL <https://doi.org/10.1051/0004-6361/201425555>. [p]
- L. Kelvin. *astro: Astronomy Functions, Tools and Routines*, 2014. URL <https://CRAN.R-project.org/package=astro>. R package version 1.2. [p]
- N. Leonenko and N.-R. Shieh. Rényi function for multifractal random fields. *Fractals*, 21(02):1350009(1–13), 2013. URL <https://doi.org/10.1142/S0218348X13500096>. [p]
- N. Leonenko, M. Taqqu, and G. Terdik. Estimation of the covariance function of Gaussian isotropic random fields on spheres, related Rosenblatt-type distributions and the cosmic variance problem. *Electron. J. Statist.*, 12(2):3114–3146, 2018. URL <https://doi.org/10.1214/18-EJS1473>. [p]
- N. Leonenko, R. Nanayakkara, and A. Olenko. Analysis of spherical monofractal and multifractal random fields with cosmological applications. *arXiv*, (2004.14522):1–30, 2020. URL <https://arxiv.org/abs/2004.14522>. [p]
- J. Liu. *CRAC: Cosmology R Analysis Code*, 2014. URL <https://CRAN.R-project.org/package=CRAC>. R package version 1.0. [p]
- K. Mardia and P. Jupp. *Directional Statistics*. Wiley, 2009. ISBN 9780470317815. [p]
- D. Marinucci and G. Peccati. *Random Fields on the Sphere: Representation, Limit Theorems and Cosmological Applications*. Cambridge University Press, Cambridge, 2011. ISBN 9780521175616. [p]
- O. Mersmann. *microbenchmark: Accurate Timing Functions*, 2018. URL <https://CRAN.R-project.org/package=microbenchmark>. R package version 1.4-6. [p]
- J. P. Nolan. *gensphere: Generalized Spherical Distributions*, 2017. URL <https://CRAN.R-project.org/package=gensphere>. R package version 1.1. [p]
- J. P. Nolan and A. University. *SphericalCubature: Numerical Integration over Spheres and Balls in n-Dimensions; Multivariate Polar Coordinates*, 2017. URL <https://CRAN.R-project.org/package=SphericalCubature>. R package version 1.4. [p]
- P. J. Ribeiro Jr and P. J. Diggle. *geoR: Analysis of Geostatistical Data*, 2018. URL <https://CRAN.R-project.org/package=geoR>. R package version 1.7-5.2.1. [p]
- A. Robotham. *sphereplot: Spherical plotting*, 2013. URL <https://CRAN.R-project.org/package=sphereplot>. R package version 1.5. [p]
- J. A. Ryan. *mmap: R interface to POSIX mmap and Window's MapViewOfFile*, 2018. R package Version 0.6-15. [p]
- M. Schlather, A. Malinowski, P. J. Menck, M. Oesting, and K. Strokorb. Analysis, simulation and prediction of multivariate random fields with package RandomFields. *Journal of Statistical Software*, 63(8):1–25, 2015. URL <http://www.jstatsoft.org/v63/i08/>. [p]
- M. Tsagris, G. Athineou, A. Sajib, E. Amson, and M. J. Waldstein. *Directional: Directional Statistics*, 2019. URL <https://CRAN.R-project.org/package=Directional>. R package version 3.7. [p]

M. I. Yadrenko. *Spectral Theory of Random Fields*. Optimization Software, New York, 1983. ISBN 0911575006. [p]

Daniel Fryer
School of Mathematics and Physics
The University of Queensland, St Lucia, 4067
Australia
ORCID: 0000-0001-6032-0522
daniel.fryer@uq.edu.au

Ming Li
Department of Educational Technology
Zhejiang Normal University, Jinhua, 321004
China
ORCID: 0000-0002-1218-2804
mingli@zjnu.edu.cn

Andriy Olenko
Department of Mathematics and Statistics
La Trobe University, VIC, 3086
Australia
ORCID: 0000-0002-0917-7000
a.olenko@latrobe.edu.au

Appendix: Statistical model

Consider a sphere in the three-dimensional Euclidean space $\mathbb{S}^2 = \{\mathbf{x} \in \mathbb{R}^3 : \|\mathbf{x}\| = 1\} \subset \mathbb{R}^3$.

A spherical random field on a probability space $(\Omega, \mathcal{F}, \mathbf{P})$, denoted by

$$T = \{T(\theta, \varphi) = T_\omega(\theta, \varphi) : 0 \leq \theta \leq \pi, 0 \leq \varphi \leq 2\pi, \omega \in \Omega\},$$

or $\tilde{T} = \{\tilde{T}(\mathbf{x}) : \mathbf{x} \in \mathbb{S}^2\}$, is a stochastic function defined on the sphere \mathbb{S}^2 .

The field $\tilde{T}(\mathbf{x})$ is called isotropic on the sphere \mathbb{S}^2 if its mean $\mathbf{E}T(\theta, \varphi) = c = \text{constant}$ and the covariance function $\mathbf{E}T(\theta, \varphi)T(\theta', \varphi')$ depends only on the angular distance $\Theta = \Theta_{PQ}$ between the points $P = (\theta, \varphi)$ and $Q = (\theta', \varphi')$ on \mathbb{S}^2 .

A real-valued second-order mean-square continuous spherical random field T can be expanded in the series

$$T(\theta, \varphi) = \sum_{l=0}^{\infty} \sum_{m=-l}^l a_{lm} Y_{lm}(\theta, \varphi), \quad (1)$$

where $\{Y_{lm}(\theta, \varphi)\}$ represents the complex spherical harmonics and a_{lm} are random variables.

If a random field is isotropic then

$$\mathbf{E}a_{lm}a_{l'm'}^* = \delta_l^{l'} \delta_m^{m'} C_l, \quad -l \leq m \leq l, \quad -l' \leq m' \leq l', \quad l \geq 0.$$

Thus, $\mathbf{E}|a_{lm}|^2 = C_l$, $m = 0, \pm 1, \dots, \pm l$. The series $\{C_1, C_2, \dots, C_l, \dots\}$ is called the angular power spectrum of the isotropic random field $T(\theta, \varphi)$.

The covariance function of the isotropic random fields T has the following representation

$$\Gamma(\cos \Theta) = \mathbf{E}T(\theta, \varphi)T(\theta', \varphi') = \frac{1}{4\pi} \sum_{l=0}^{\infty} (2l+1) C_l P_l(\cos \Theta), \quad (2)$$

where $P_l(\cdot)$ is the l -th Legendre polynomial and $\sum_{l=0}^{\infty} (2l+1) C_l < \infty$.

The variogram (semivariogram) function is defined by

$$\gamma(h) = \Gamma(0) - \Gamma(h). \quad (3)$$

The package uses observations of the field T at HEALPix points. There are two main approaches in the statistical analysis of T that are realised in **rcosmo**. First approach directly uses the observations for parameter estimation and hypothesis tests. For example, the classical estimator of the

isotropic variogram $\gamma(h)$ takes the form of

$$\hat{\gamma}(h) = \frac{1}{2N_h} \sum_{(\mathbf{x}_1, \mathbf{x}_2) \in N_h} (T(\mathbf{x}_1) - T(\mathbf{x}_2))^2,$$

where N_h is the set of the spherical location pairs at the geodesic distance h . The second approach is spectrum-based. Initially, the estimates \hat{a}_{lm} are computed by inverting (1). Then, empirical covariance functions and variograms can be obtained by substituting the estimated values of the angular power spectrum

$$\hat{C}_l = \frac{1}{2l+1} \sum_{m=-l}^l |\hat{a}_{lm}|^2$$

in equations 2 and 3.

Tools for Analyzing R Code the Tidy Way

by Lucy D'Agostino McGowan, Sean Kross, Jeffrey Leek

Abstract With the current emphasis on reproducibility and replicability, there is an increasing need to examine how data analyses are conducted. In order to analyze the between researcher variability in data analysis choices as well as the aspects within the data analysis pipeline that contribute to the variability in results, we have created two R packages: `matahari` and `tidycode`. These packages build on methods created for natural language processing; rather than allowing for the processing of natural language, we focus on R code as the substrate of interest. The `matahari` package facilitates the logging of everything that is typed in the R console or in an R script in a tidy data frame. The `tidycode` package contains tools to allow for analyzing R calls in a tidy manner. We demonstrate the utility of these packages as well as walk through two examples.

Introduction

With the current emphasis on reproducibility and replicability, there is an increasing need to examine how data analyses are conducted (Goecks et al. 2010; Peng 2011; McNutt 2014; Miguel et al. 2014; Ioannidis et al. 2014; Richard 2014; Leek and Peng 2015; Nosek et al. 2015; Sidi and Harel 2018). In order to accurately replicate a result, the exact methods used for data analysis need to be recorded, including the specific analytic steps taken as well as the software utilized (Waltemath and Wolkenhauer 2016). Studies across multiple disciplines have examined the global set of possible data analyses that can be conducted on a specific data set (Silberzhan et al. 2018). While we are able to define this global set, very little is known about the actual variation that exists between researchers. For example, it is possible that the true range of data analysis choices is realistically a much more narrow set than the global sets that are presented. There is a breadth of excellent research and experiments examining how people read visual information (Majumder, Hofmann, and Cook 2013; Loy, Hofmann, and Cook 2017; Wickham, Cook, and Hofmann 2015; Buja et al. 2009; Loy, Follett, and Hofmann 2016), for example the Experiments on Visual Inference detailed here: (<http://mamajumder.github.io/html/experiments.html>), but not how they actually make analysis choices, specifically analysis *coding* choices. In addition to not knowing about the “data analysis choice” variability between researchers, we also do not know which portions of the data analysis pipeline result in the most variability in the ultimate research result. We seek to build tools to analyze these two aspects of data analysis:

1. The between researcher variability in data analysis choices
2. The aspects within the data analysis pipeline that contribute to the variability in results

Specifically, we have designed a framework to conduct such analyses and created two R packages that allow for the study of data analysis code conducted in R. In addition to answering these crucial questions for broad research fields, we see these tools having additional concrete use cases. These tools will facilitate data science and statistics pedagogy, allowing researchers and instructors to investigate how students are conducting data analyses in the classroom. Alternatively, a researcher could use these tools to examine how collaborators have conducted a data analysis. Finally, these tools could be used in a meta-manner to explore how current software and tools in R are being utilized.

Tidy principles

We specifically employ *tidy* principles in our proposed packages. *Tidy* refers to an implementation strategy propagated by Hadley Wickham and implemented by the Tidyverse team at RStudio (Wickham and Grolemund 2016). Here, by *tidy* we mean our packages adhere to the following principles:

1. Our functions follow the principles outlined in *R packages* (Wickham 2015) as well as the *tidyverse style guide* (Wickham 2019).
2. Our output data sets are tidy, as in:
 - Each variable has its own column.
 - Each observation has its own row.
 - Each value has its own cell.

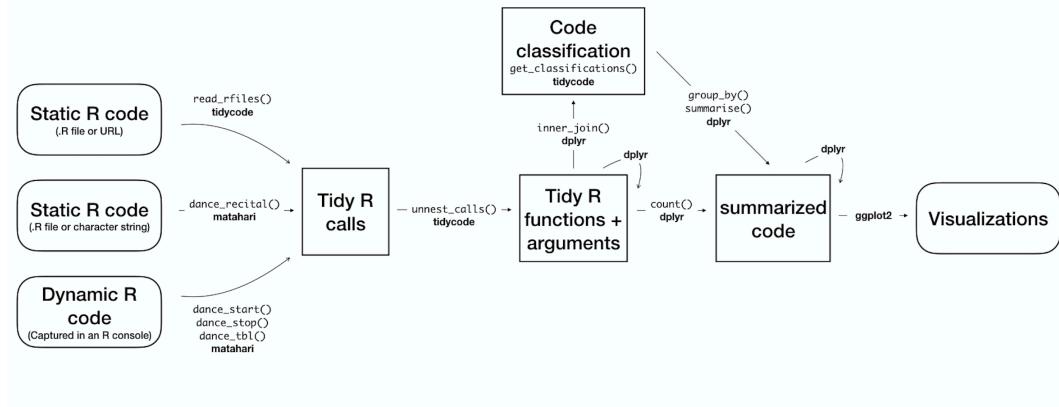


Figure 1: A flowchart of a typical analysis that uses `matahari` and `tidycode` to analyze and classify R code.

By implementing these tidy principles, and thus outputting tidy data frames, we allow for data manipulation and analysis to be conducted using a specific set of tools, such as those included in the `tidyverse` meta package (Wickham et al. 2019).

Ultimately, we create a mechanism to utilize methods created for natural language processing; here the substrate is *code* rather than natural language. We model our tools to emulate the `tidytext` package (Silge and Robinson 2016, 2017); instead of analyzing tokens of text, we are analyzing tokens of code.

We present two packages, `matahari`, a package for logging everything that is typed in the R console or in an R script, and `tidycode`, a package with tools to allow for analyzing R calls in a tidy manner. In this paper, we first explain how these packages work. We then demonstrate two examples, one that analyzes data collected from an online experiment, and one that analyzes “old” data via previously created R scripts.

Methods

We have created two R packages, `matahari` and `tidycode`. The former is a way to log R code, the latter allows the user to analyze R calls on the function-level in a tidy manner. Figure 1 is a flowchart of the process described in more detail below. This flowchart is adapted from Figure 2.1 in *Text Mining with R: A Tidy Approach* (Silge and Robinson 2017).

We demonstrate how to create these tidy data frames of R code and then emulate the data analysis workflow similar to that put forth in the tidy text literature.

Terminology

In this paper, we refer to R “expressions” or “calls” as well as R “functions” and “arguments”. An R call is a combination of an R function with arguments. For example, the following is an R call (Example 1).

```
library(tidycode)
```

Example 1. R call, library

Another example of an R call is the following piped chain of functions from the `dplyr` package (Example 2).

```
starwars %>%
  select(height, mass)
```

Example 2. Piped R call

Specifically, we know something is a call in R if `is.call()` is TRUE.

```
quote(starwars %>%
  select(height, mass)) %>%
  is.call()
```

```
#> [1] TRUE

Calls in R are made up of a function or name of a function, and arguments. For example, the call library(tidycode) from Example 1 is comprised of the function library() and the argument tidycode. Example 2 is a bit more complicated. The piped code can be rewritten, as seen in Example 3.

`%>%`(starwars, select(height, mass))
```

Example 3. Rewritten piped R call

From this example, it is easier to see that the function for this R call is `%>%` with two arguments, `starwars` and `select(height, mass)`. Notice that one of these arguments is an R call itself, `select(height, mass)`.

matahari

matahari is a simple package for logging R code in a tidy manner. It can be installed from CRAN using the following code.

```
install.packages("matahari")
```

There are three ways to use the matahari package:

1. Record R code as it is typed and output a tidy data frame of the contents
2. Input a character string of R code and output a tidy data frame of the contents
3. Input an R file containing R code and output a tidy data frame of the contents

In the following sections, we will split these into two categories, tidy logging from the R console (1) and tidy logging from an R script (2 and 3).

Tidy logging from the R console In order to begin logging from the R console, the `dance_start()` function is used. Logging is paused using `dance_stop()` and the log can be viewed using `dance_tbl()`. For example, the following code will result in the subsequent tidy data frame.

```
library(matahari)
dance_start()
1 + 2
"here is some text"
sum(1:10)
dance_stop()
dance_tbl()

#> # A tibble: 6 x 6
#>   expr      value      path      contents      selection      dt
#>   <list>    <list>    <list>    <list>    <list>    <dttm>
#> 1 <languag... <S3: sessionIn... <lgl [1... <lgl [1... <lgl [1]> 2018-09-11 22:22:12
#> 2 <languag... <lgl [1]>      <lgl [1... <lgl [1... <lgl [1]> 2018-09-11 22:22:12
#> 3 <languag... <lgl [1]>      <lgl [1... <lgl [1... <lgl [1]> 2018-09-11 22:22:12
#> 4 <chr [1]> <lgl [1]>      <lgl [1... <lgl [1... <lgl [1]> 2018-09-11 22:22:12
#> 5 <languag... <lgl [1]>      <lgl [1... <lgl [1... <lgl [1]> 2018-09-11 22:22:12
#> 6 <languag... <S3: sessionIn... <lgl [1... <lgl [1... <lgl [1]> 2018-09-11 22:22:12
```

Example 4. Logging R code from the R console using `matahari`

The resulting tidy data frame consists of 6 columns: `expr`, the R call that was run, `value`, the value that was output, `path`, if the code was run within RStudio, this will be the path to the file in focus, `contents`, the file contents of the RStudio editor tab in focus, `selection`, the text that is highlighted in the RStudio editor tab in focus, and `dt`, the date and time the expression was run. By default, `value`, `path`, `contents` and `selection` will not be logged unless the argument is set to `TRUE` in the `dance_start()` function. For example, if the analyst wanted the output data frame to include the values computed, they would input `dance_start(value = TRUE)`.

In this particular data frame, there are 6 rows. The first and final rows report the R session information at the time when `dance_start()` was initiated (row 1) and when `dance_stop()` was run (row 6). The second row holds the R call `dance_start()`, the first command run in the R console, was run; the third row holds `1 + 2`, the fourth holds `here is some text`, and the fifth holds `sum(1:10)`.

```
dance_tbl() [["expr"]]

#> [[1]]
#> sessionInfo()
#>
#> [[2]]
#> dance_start()
#>
#> [[3]]
#> 1 + 2
#>
#> [[4]]
#> [1] "here is some text"
#>
#> [[5]]
#> sum(1:10)
#>
#> [[6]]
#> sessionInfo()
```

These functions work by saving an invisible data frame called `.dance` that is referenced by `dance_tbl()`. Each time `dance_start()` is subsequently run after `dance_stop()`, new rows of data are added to this data frame. This invisible data frame exists in a new environment created by the [matahari](#) package. We can remove this data frame by running `dance_remove()`.

This data frame can be manipulated using common R techniques. Below, we rerun the same code as above, this time saving the values that are computed in the R console by using the `value = TRUE` parameter.

```
dance_start(value = TRUE)
1 + 2
"here is some text"
sum(1:10)
dance_stop()
tbl <- dance_tbl()
```

As an example of the type of data wrangling that this tidy format allows for, using [dplyr](#) and [purrr](#), we can manipulate this to only examine expressions that result in numeric values.

```
library(dplyr)
library(purrr)

t_numeric <- tbl %>%
  mutate(
    numeric_output = map_lgl(value, is.numeric)
  ) %>%
  filter(numeric_output)

t_numeric

#> # A tibble: 3 x 7
#>   expr      value     path     contents  selection dt          numeric_output
#>   <list>    <list>    <list>    <list>    <list>    <dttm>        <lgl>
#> 1 <language> <int [1]> <lgl [1]> <lgl [1]> <lgl [1]> 2019-04-29 22:39:05 TRUE
#> 2 <language> <dbl [1]> <lgl [1]> <lgl [1]> <lgl [1]> 2019-04-29 22:39:05 TRUE
#> 3 <language> <int [1]> <lgl [1]> <lgl [1]> <lgl [1]> 2019-04-29 22:39:05 TRUE
```

Here, three rows are output, since we have filtered to only calls with numeric output:

1. The `dance_start()` call (this defaults to have a numeric value of 1)
2. The `1 + 2` call, resulting in a `value` of 3
3. The `sum(1:10)`, resulting in a `value` of 55

Tidy logging from an R script In addition to allowing for the logging of everything typed in the R console, the [matahari](#) package also allows for the logging of pre-created R scripts. This

can be done using the `dance_recital()` function, which allows for either a .R file or a character string of R calls as the input. For example, if we have a code file called `sample_code.R`, we can run `dance_recital("sample_code.R")` to create a tidy data frame. Alternatively, we can enter code directly as a string of text, such as `dance_recital("1 + 2")` to create the tidy data frame. Below illustrates this functionality.

```
code_file <- system.file("test", "sample_code.R", package = "matahari")
dance_recital(code_file)

#> # A tibble: 7 x 6
#>   expr      value    error      output    warnings  messages
#>   <list>    <list>    <list>    <list>    <list>    <list>
#> 1 <language> <dbl [1]> <NULL>    <chr [1]> <chr [0]> <chr [0]>
#> 2 <chr [1]> <chr [1]> <NULL>    <chr [1]> <chr [0]> <chr [0]>
#> 3 <language> <dbl [1]> <NULL>    <chr [1]> <chr [0]> <chr [0]>
#> 4 <language> <NULL>    <S3: simpleError> <NULL>    <NULL>    <NULL>
#> 5 <language> <chr [1]> <NULL>    <chr [1]> <chr [1]> <chr [0]>
#> 6 <language> <NULL>    <NULL>    <chr [1]> <chr [0]> <chr [1]>
#> 7 <language> <NULL>    <NULL>    <chr [1]> <chr [0]> <chr [0]>
```

Example 5. R call, Logging code from a .R file using `matahari`

```
code_string <- '
4 + 4
-wow!
mean(1:10)
stop("Error!")
warning("Warning!")
message("Hello?")
cat("Welcome!")
'

dance_recital(code_string)

#> # A tibble: 7 x 6
#>   expr      value    error      output    warnings  messages
#>   <list>    <list>    <list>    <list>    <list>    <list>
#> 1 <language> <dbl [1]> <NULL>    <chr [1]> <chr [0]> <chr [0]>
#> 2 <chr [1]> <chr [1]> <NULL>    <chr [1]> <chr [0]> <chr [0]>
#> 3 <language> <dbl [1]> <NULL>    <chr [1]> <chr [0]> <chr [0]>
#> 4 <language> <NULL>    <S3: simpleError> <NULL>    <NULL>    <NULL>
#> 5 <language> <chr [1]> <NULL>    <chr [1]> <chr [1]> <chr [0]>
#> 6 <language> <NULL>    <NULL>    <chr [1]> <chr [0]> <chr [1]>
#> 7 <language> <NULL>    <NULL>    <chr [1]> <chr [0]> <chr [0]>
```

Example 6. Logging code from a character string using `matahari`

The resulting tidy data frame from `dance_recital()`, as seen in Examples 5 and 6, is different from that of `dance_tbl()`. This data frame has 6 columns. The first is the same as the `dance_tbl()`, `expr`, the R calls in the .R script or string of code. The subsequent columns are, `value`, the computed result of the R call, `error`, which contains the resulting error object from a poorly formed call, `output`, the printed output from a call, `warnings`, the contents of any warnings that would be displayed in the console, and `messages`, the contents of any generated diagnostic messages. Now that we have a tidy data frame with R calls obtained either from the R console or from a .R script, we can analyze them using the `tidycode` package.

The development version of the `matahari` package can be found on GitHub at <https://github.com/jhudsl/matahari>. Users can submit feature requests, issues, and bug reports here.

tidycode

The goal of `tidycode` is to allow users to analyze R scripts, calls, and functions in a tidy way. There are two main tasks that can be achieved with this package:

1. We can “tokenize” R calls

2. We can classify the functions run into one of nine potential data analysis categories: “Setup”, “Exploratory”, “Data Cleaning”, “Modeling”, “Evaluation”, “Visualization”, “Communication”, “Import”, or “Export”.

The `tidycode` package can be installed from CRAN in the following manner.

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

We can first create a tidy data frame using the `matahari` package. Alternatively, we can use a function in the `tidycode` package that wraps the `dance_recital()` function called `read_rfiles()`. This function allows you to read in multiple .R files or links to .R files. There are a few example files included in the `tidycode` package. The paths to these files can be accessed via the `tidycode_example()` function. For example, running the following code will give the file path for the `example_analysis.R` file.

```
tidycode_example("example_analysis.R")
```

```
#> [1] "/Library/Frameworks/R.framework/Versions/3.5/Resources/library/tidycode/extdata/example_analysis.R"
```

Running the function without any file specified will supply a vector of all available file names.

```
tidycode_example()
#> [1] "example_analysis.R" "example_plot.R"
```

We can use these example files in the `read_rfiles()` function.

```
df <- read_rfiles(tidycode_example(c("example_analysis.R", "example_plot.R")))
df

#> # A tibble: 9 x 3
#>   file                               expr      line
#>   <chr>                             <list>    <int>
#> 1 /Library/Frameworks/R.framework/Versions/3.5/Resources/li~ <langua~     1
#> 2 /Library/Frameworks/R.framework/Versions/3.5/Resources/li~ <langua~     2
#> 3 /Library/Frameworks/R.framework/Versions/3.5/Resources/li~ <langua~     3
#> 4 /Library/Frameworks/R.framework/Versions/3.5/Resources/li~ <langua~     4
#> 5 /Library/Frameworks/R.framework/Versions/3.5/Resources/li~ <langua~     5
#> 6 /Library/Frameworks/R.framework/Versions/3.5/Resources/li~ <langua~     6
#> 7 /Library/Frameworks/R.framework/Versions/3.5/Resources/li~ <langua~     7
#> 8 /Library/Frameworks/R.framework/Versions/3.5/Resources/li~ <langua~     1
#> 9 /Library/Frameworks/R.framework/Versions/3.5/Resources/li~ <langua~     2
```

This will give a tidy data frame with three columns: `file`, the path to the file, `expr` the R call, and `line` the line the call was made in the original .R file.

We can then use the `unnest_calls()` function to create a data frame of the calls, splitting each into the individual functions and arguments. We liken this to the `tidytext` `unnest_tokens()` function. This function has two parameters, `.data`, the data frame that contains the R calls, and `input` the name of the column that contains the R calls. In this case, the data frame is `df` and the input column is `expr`.

```
u <- unnest_calls(df, expr)
u

#> # A tibble: 35 x 4
#>   file                               line func    args
#>   <chr>                            <int> <chr>   <list>
#> 1 /Library/Frameworks/R.framework/Versions/3.5/R~      1 libra~ <list [1]>
#> 2 /Library/Frameworks/R.framework/Versions/3.5/R~      2 libra~ <list [1]>
#> 3 /Library/Frameworks/R.framework/Versions/3.5/R~      3 <-     <list [2]>
#> 4 /Library/Frameworks/R.framework/Versions/3.5/R~      3 %>%  <list [2]>
#> 5 /Library/Frameworks/R.framework/Versions/3.5/R~      3 %>%  <list [2]>
#> 6 /Library/Frameworks/R.framework/Versions/3.5/R~      3 mutate <named lis~
#> 7 /Library/Frameworks/R.framework/Versions/3.5/R~      3 /      <list [2]>
#> 8 /Library/Frameworks/R.framework/Versions/3.5/R~      3 (      <list [1]>
#> 9 /Library/Frameworks/R.framework/Versions/3.5/R~      3 ^      <list [2]>
#> 10 /Library/Frameworks/R.framework/Versions/3.5/R~     3 (      <list [1]>
#> # ... with 25 more rows
```

This results is a tidy data frame with two additional columns: `func` the name of the function called and `args` the arguments of the function called. Because this function takes a data frame as the first argument, it works nicely with the tidyverse data manipulation packages. For example, we could get the same data frame as above by using the following code.

```
df %>%
  unnest_calls(expr)

#> # A tibble: 35 x 4
#>   file                      line func   args
#>   <chr>                     <int> <chr>  <list>
#> 1 /Library/Frameworks/R.framework/Versions/3.5/R~    1 libra~ <list [1]>
#> 2 /Library/Frameworks/R.framework/Versions/3.5/R~    2 libra~ <list [1]>
#> 3 /Library/Frameworks/R.framework/Versions/3.5/R~    3 <-    <list [2]>
#> 4 /Library/Frameworks/R.framework/Versions/3.5/R~    3 %>% <list [2]>
#> 5 /Library/Frameworks/R.framework/Versions/3.5/R~    3 %>% <list [2]>
#> 6 /Library/Frameworks/R.framework/Versions/3.5/R~    3 mutate <named lis~
#> 7 /Library/Frameworks/R.framework/Versions/3.5/R~    3 /     <list [2]>
#> 8 /Library/Frameworks/R.framework/Versions/3.5/R~    3 (     <list [1]>
#> 9 /Library/Frameworks/R.framework/Versions/3.5/R~    3 ^     <list [2]>
#> 10 /Library/Frameworks/R.framework/Versions/3.5/R~   3 (     <list [1]>
#> # ... with 25 more rows
```

We can further manipulate this, for example we could select just the `func` and `args` columns using `dplyr`'s `select()` function.

```
df %>%
  unnest_calls(expr) %>%
  select(func, args)

#> # A tibble: 35 x 2
#>   func      args
#>   <chr>    <list>
#> 1 library <list [1]>
#> 2 library <list [1]>
#> 3 <-      <list [2]>
#> 4 %>%    <list [2]>
#> 5 %>%    <list [2]>
#> 6 mutate  <named list [1]>
#> 7 /       <list [2]>
#> 8 (       <list [1]>
#> 9 ^       <list [2]>
#> 10 (      <list [1]>
#> # ... with 25 more rows
```

The `get_classifications()` function calls a classification data frame that we curated that classifies the individual functions into one of nine categories: setup, exploratory, data cleaning, modeling, evaluation, visualization, communication, import, or export. This can also be merged into the data frame. For this classification analysis, we are using an `inner_join()`, keeping only the functions that are classified, similar to the workflow for a *sentiment analysis* in `tidytext`(Silge and Robinson 2017). If you did not want to remove unclassified functions from your dataframe, the `left_join()` function would be appropriate.

```
u %>%
  inner_join(get_classifications()) %>%
  select(func, classification, lexicon, score)

#> # A tibble: 322 x 4
#>   func      classification lexicon      score
#>   <chr>    <chr>        <chr>        <dbl>
#> 1 library setup          crowdsource  0.687
#> 2 library import         crowdsource  0.213
#> 3 library visualization crowdsource  0.0339
#> 4 library data cleaning crowdsource  0.0278
#> 5 library modeling      crowdsource  0.0134
#> 6 library exploratory    crowdsource  0.0128
```

```
#> 7 library communication  crowdsource 0.00835
#> 8 library evaluation    crowdsource 0.00278
#> 9 library export       crowdsource 0.00111
#> 10 library setup       leeklab     0.994
#> # ... with 312 more rows
```

There are two lexicons for classification, `crowdsource` and `leeklab`. The former was created by volunteers who classified R code using the [classify shiny application](#). The latter was curated by [Jeff Leek's Lab](#). To select a particular lexicon, you can specify the `lexicon` parameter. For example, the following code will merge in the `crowdsource` lexicon only.

```
u %>%
  inner_join(get_classifications("crowdsource")) %>%
  select(func, classification, score)

#> # A tibble: 271 x 3
#>   func    classification  score
#>   <chr>   <chr>        <dbl>
#> 1 library setup          0.687
#> 2 library import         0.213
#> 3 library visualization 0.0339
#> 4 library data cleaning 0.0278
#> 5 library modeling       0.0134
#> 6 library exploratory    0.0128
#> 7 library communication  0.00835
#> 8 library evaluation      0.00278
#> 9 library export          0.00111
#> 10 library setup          0.687
#> # ... with 261 more rows
```

It is possible for a function to belong to multiple classes. This will result in multiple lines (and multiple classifications) for a given function. By default, these multiple classifications are included along with the prevalence of each, indicated by the `score` column. To merge in only the most prevalent classification, set the `include_duplicates` option to `FALSE`.

```
u %>%
  inner_join(get_classifications("crowdsource", include_duplicates = FALSE)) %>%
  select(func, classification)

#> # A tibble: 33 x 2
#>   func    classification
#>   <chr>   <chr>
#> 1 library setup
#> 2 library setup
#> 3 <-    data cleaning
#> 4 %>%    data cleaning
#> 5 %>%    data cleaning
#> 6 mutate  data cleaning
#> 7 /      data cleaning
#> 8 (      data cleaning
#> 9 ^      modeling
#> 10 (     data cleaning
#> # ... with 23 more rows
```

In text analysis, there is the concept of “stopwords”. These are often small common filler words you want to remove before completing an analysis, such as “a” or “the”. In a tidy `code` analysis, we can use a similar concept to remove some functions. For example we may want to remove the assignment operator, `<-`, before completing an analysis. We have compiled a list of common stop functions in the `get_stopfuncs()` function to anti join from the data frame.

```
u %>%
  inner_join(get_classifications("crowdsource", include_duplicates = FALSE)) %>%
  anti_join(get_stopfuncs()) %>%
  select(func, classification)

#> # A tibble: 15 x 2
#>   func    classification
```

```
#>      <chr>    <chr>
#> 1 library setup
#> 2 library setup
#> 3 mutate  data cleaning
#> 4 select   data cleaning
#> 5 options  setup
#> 6 summary  exploratory
#> 7 plot     visualization
#> 8 library  setup
#> 9 select   data cleaning
#> 10 filter  data cleaning
#> 11 is.na   data cleaning
#> 12 is.na   data cleaning
#> 13 ggplot  visualization
#> 14 aes     visualization
#> 15 geom_point visualization
```

The development version of the `tidycode` package can be found on GitHub at <https://github.com/jhudsl/tidycode>. Users can submit feature requests, issues, and bug reports here.

Examples

Online experiment: P-hack-a-thon

This first example demonstrates how to use the `matahari` and `tidycode` packages to analyze data from a prospective study, using the “recording” capabilities of the `matahari` package to capture the code as participants run it. Recently, we launched a “p-hack-a-thon” where we encouraged users to analyze a dataset with the goal of producing the smallest p-value (IRB # IRB00008885, Not Human Subjects Research Classification, Johns Hopkins Bloomberg School of Public Health IRB). We captured the code the participants ran using the `dance_start()` and `dance_stop()` functions from the `matahari` package. This resulted in a tidy data frame of R calls for each participant. We use the `tidycode` package to analyze these matahari data frames.

Setup

```
library(tidyverse)
library(tidycode)

## load the dataset, called df
load("data/df_phackathon.Rda")
```

The data from the “p-hack-a-thon” is saved as a data frame called `df`. This includes data from 29 participants. We have bound the `expr` column from the `matahari` data frame for each participant. Using the `unnest_calls()` function, we unnest each of these R calls into a function and its arguments.

```
tbl <- df %>%
  unnest_calls(expr)
```

We can then remove the “stop functions” by doing an anti join with the `get_stopfuncs()` function and merge in the crowd-sourced classifications with the `get_classifications()` function.

```
classification_tbl <- tbl %>%
  anti_join(get_stopfuncs()) %>%
  inner_join(get_classifications("crowdsource", include_duplicates = FALSE))
```

Classifications We can use common data manipulation functions from `dplyr`. For example, on average, “data cleaning” functions made up 36.4% of the functions run by participants (Table 1).

```
classification_tbl %>%
  group_by(id, classification) %>%
  summarise(n = n()) %>%
  mutate(pct = n / sum(n)) %>%
  group_by(classification) %>%
```

Table 1: Average percent of functions spent on each task.

classification	Average percent
data cleaning	36.40
visualization	23.17
exploratory	21.32
setup	18.87
modeling	17.69
import	8.58
communication	5.14
evaluation	3.62
export	0.82

```
summarise(`Average percent` = mean(pct) * 100) %>%
arrange(-`Average percent`)
```

We can also examine the most common functions in each classification.

```
func_counts <- classification_tbl %>%
  count(func, classification, sort = TRUE) %>%
  ungroup()

func_counts

#> # A tibble: 152 x 3
#>   func      classification     n
#>   <chr>    <chr>        <int>
#> 1 summary  exploratory      361
#> 2 lm       modeling        277
#> 3 factor   data cleaning   141
#> 4 select   data cleaning   138
#> 5 library  setup          128
#> 6 as.factor data cleaning  116
#> 7 filter   data cleaning   107
#> 8 aes      visualization   89
#> 9 ggplot   visualization   82
#> 10 lmer    modeling        80
#> # ... with 142 more rows

func_counts %>%
  filter(classification %in% c("data cleaning", "exploratory", "modeling", "visualization")) %>%
  group_by(classification) %>%
  top_n(5) %>%
  ungroup() %>%
  mutate(func = reorder(func, n)) %>%
  ggplot(aes(func, n, fill = classification)) +
  theme_bw() +
  geom_col(show.legend = FALSE) +
  facet_wrap(~classification, scales = "free_y") +
  scale_x_discrete(element_blank()) +
  scale_y_continuous("Number of function calls in each classification") +
  coord_flip()
```

We could then examine a word cloud of the functions used, colored by the classification. We can do this using the `wordcloud` library.

```
library(wordcloud)

classification_tbl %>%
  count(func, classification) %>%
  with(
    wordcloud(func, n,
```

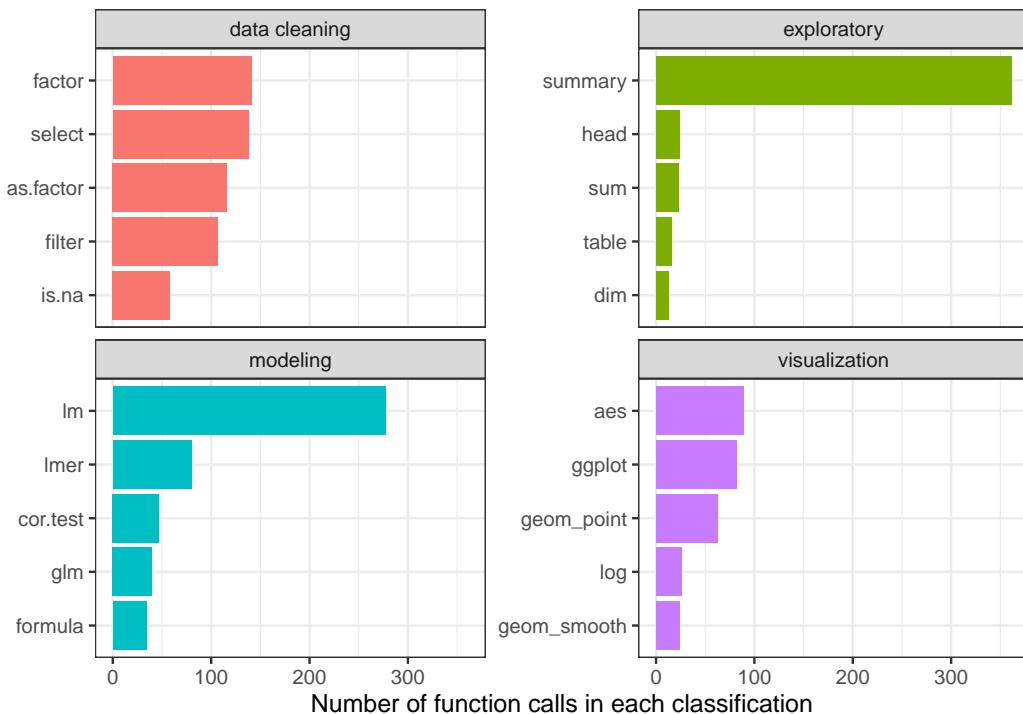


Figure 2: Functions that contribute to data cleaning, exploratory analysis, modeling and visualization classifications in p-hack-a-thon trial. This plot ranks the most common functions in each classification and displays the top 5, with the x-axis representing the number of times the given function was called. For example, by far the most common function classified as “exploratory” is the ‘summary()’ function. The most common function classified as “modeling” is the ‘lm()’ function.

```

colors = brewer.pal(9, "Set1")[factor(.classification)],
random.order = FALSE,
ordered.colors = TRUE
)
)

```

Additionally, we could examine the variability in the types of functions used between groups. For example, we asked participants whether they perform analyses as part of their job. 82.76% of participants ($n = 24$) answered “Yes”.

```

classification_tbl %>%
  group_by(id, analysis_job, classification) %>%
  summarise(n = n()) %>%
  mutate(pct = n / sum(n)) %>%
  group_by(analysis_job, classification) %>%
  summarise(n = n()) %>%
  mutate(avg_pct = n / sum(n)) %>%
  ggplot(aes(x = analysis_job, y = avg_pct, fill = classification)) +
  geom_bar(stat = "identity") +
  scale_y_continuous("Average percent", labels = scales::percent) +
  scale_x_discrete("Participant conducts analyses as part of their job")

```

Figure 4 demonstrates the variability in the types of functions users ran, split by whether they conduct analyses as part of their jobs. It appears that users who conduct analyses as part of their jobs ran a larger percentage of functions classified as “modeling”, “exploratory”, and “communication”, whereas those who do not ran a larger percentage of “setup” functions. Of note, among those who do not conduct analyses as part of their job, there were 0 functions used that classify as “communication”. Had this experiment been run on a larger scale, we could potentially draw inference on the differences between these two groups and how they choose to code.

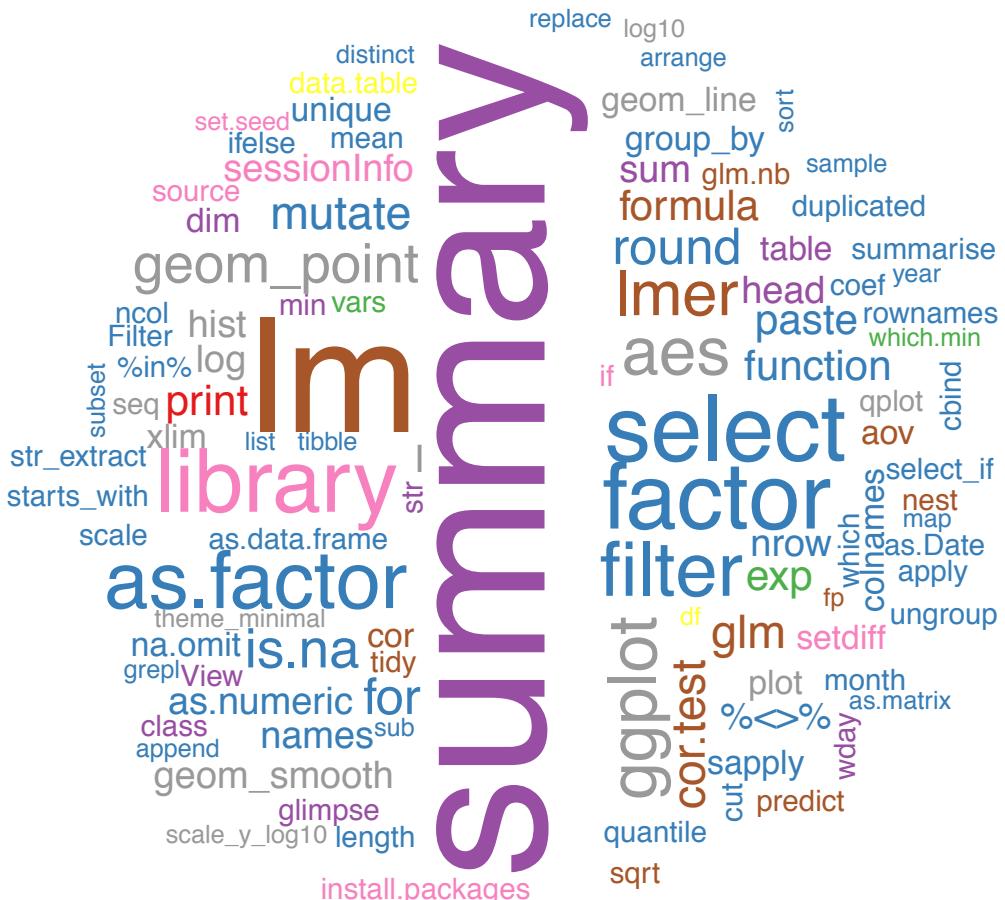


Figure 3: Word cloud of functions used in the p-hack-athon trial, colored by classification. This provides another view of the most common functions run in the p-hack-athon. The size of the function corresponds to the number of times the function is called. For example, the ‘summary()’ function is the largest, as this was the most frequently called function. The color of the function corresponds to the classification of the function: red: communication, blue: data cleaning, green: evaluation, purple: exploratory, orange: export, yellow: import, brown: modeling, pink: setup, grey: visualization.

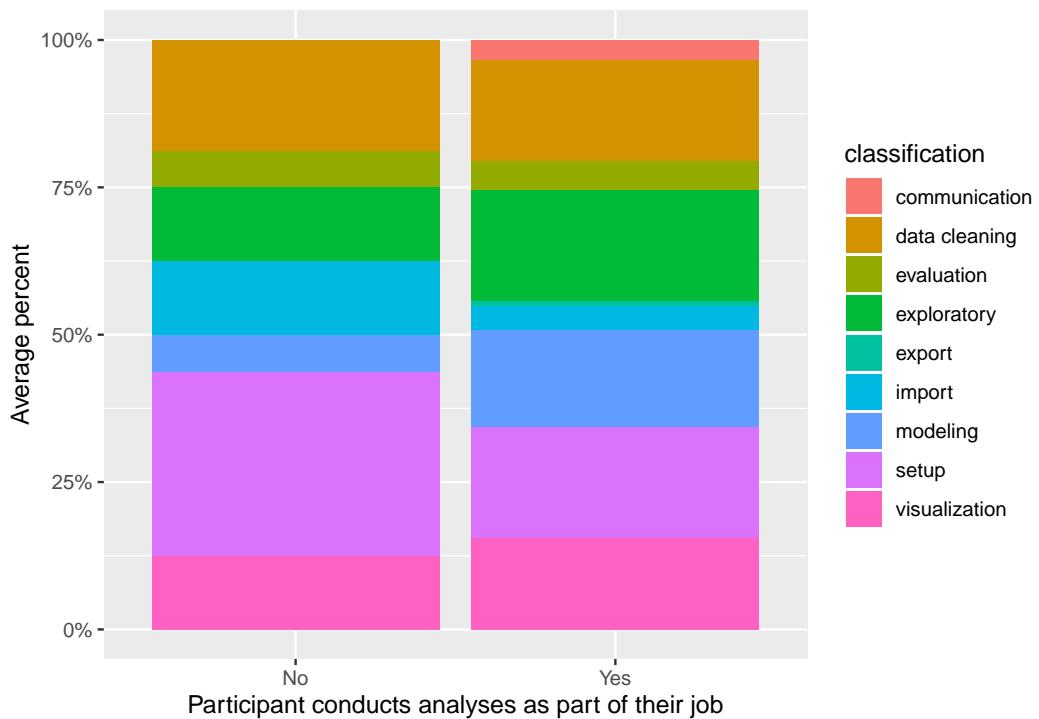


Figure 4: Variability in the types of functions used by whether the participant performs analyses as part of their job. The bar on the left represents the distribution of average classifications among those who do not perform analyses as part of their job, the bar on the right represents the distribution of the average classifications among those who do perform analyses as part of their job. It appears that users who conduct analyses as part of their jobs ran a larger percentage of functions classified as "modeling", "exploratory", and "communication", whereas those who do not ran a larger percentage of "setup" functions. Of note, among those who do not conduct analyses as part of their job, there were 0 functions used that classify as "communication".

Static Analysis

This second example demonstrates how to use the `matahari` and `tidycode` packages to analyze data from a retrospective study, or static R scripts. Here, we use the `read_rfiles()` function from the `tidycode` package. This wraps the `dance_recital()` `matahari` function and allows for multiple file paths or URLs to be read, resulting in a tidy data frame. As an example, we are going to scrape all of the .R files from two of the most widely used data manipulation packages, the `data.table` package (Dowle and Srinivasan 2019) and the `dplyr` package. We are going to use the `gh` package (Bryan and Wickham 2017) to scrape these files from GitHub.

Setup We access the files via GitHub using the `gh()` function from the `gh` package. This gives a list of download URLs that can be passed to the `read_rfiles()` function from the `tidycode` package.

```
library(tidyverse)
library(gh)
library(tidycode)

dplyr_code <- gh("/repos/tidyverse/dplyr/contents/R") %>%
  purrr::map("download_url") %>%
  read_rfiles()

datatable_code <- gh("/repos/Rdatatable/data.table/contents/R") %>%
  purrr::map("download_url") %>%
  read_rfiles()
```

Data Cleaning We can combine these two tidy data frames. We will do some small data manipulation, removing R calls that were either `NULL` or `character`. For example, in the `dplyr` package some .R files just reference data frames as a character string.

```
pkg_data <- bind_rows(
  list(
    dplyr = dplyr_code,
    datatable = datatable_code
  ),
  .id = "pkg"
) %>%
  filter(
    !map_lgl(expr, is.null),
    !map_lgl(expr, is.character)
)
```

Analyze R functions Now we can use the `tidycode unnest_calls()` function to create a tidy data frame of the individual functions along with the arguments used to create both packages. Notice here we are not performing an anti join on “stop functions”. For this analysis, we are interested in examining some key differences in the commonly used functions contained the two packages. Common operators may actually be of interest, so we do not want to drop them from the data frame. We can count the functions by package.

```
func_counts <- pkg_data %>%
  unnest_calls(expr) %>%
  count(pkg, func, sort = TRUE)

func_counts

#> # A tibble: 1,163 x 3
#>   pkg      func     n
#>   <chr>    <chr>   <int>
#> 1 datatable =       1640
#> 2 dplyr     <-      1634
#> 3 datatable if      1590
#> 4 datatable {      1172
#> 5 dplyr     {      1047
#> 6 dplyr     function 724
```

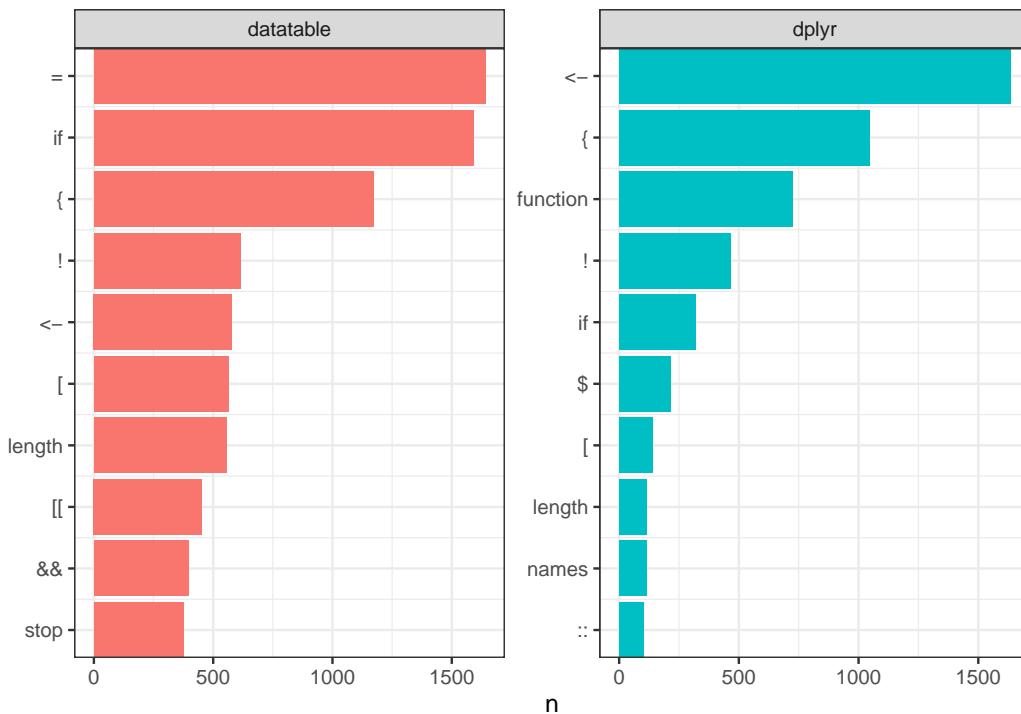


Figure 5: Most frequent functions used in `data.table` and `dplyr` package development. The y-axis displays the 10 most frequently used functions for each package, the x-axis represents the number of times that function is implemented. This allows us to examine coding style, for example the most frequent function in the `data.table` packages is ‘=’, compared to ‘<-’ in the `dplyr` package. The authors of `data.table` use the ‘=’ as an assignment operator at times, explaining this difference.

```
#> 7 datatable !      616
#> 8 datatable <-    579
#> 9 datatable [     564
#> 10 datatable length   557
#> # ... with 1,153 more rows
```

Using this data frame, we can visualize which functions are most commonly called in each package.

```
top_funcs <- func_counts %>%
  group_by(pkg) %>%
  top_n(10) %>%
  ungroup() %>%
  arrange(pkg, n) %>%
  mutate(i = row_number())

ggplot(top_funcs, aes(i, n, fill = pkg)) +
  theme_bw() +
  geom_col(show.legend = FALSE) +
  facet_wrap(~pkg, scales = "free") +
  scale_x_continuous(
    element_blank(),
    breaks = top_funcs$i,
    labels = top_funcs$func,
    expand = c(0, 0)
  ) +
  coord_flip()
```

We can glean a few interesting details from Figure 5. First, the `data.table` authors sometimes use the `=` as an assignment operator, resulting in this being the most frequent function used. The `dplyr` authors always use `<-` for assignment, therefore this is the most frequent function seen in

this package (Wickham 2019). Additionally, the `dplyr` authors often create modular code as a combination of small functions to complete specific tasks. This may explain why `function` is the third most frequent R call in this package, and less prevalent in the `data.table` package. This just serves as a glimpse of what can be accomplished with these tools.

Discussion

We have designed a framework to analyze the data analysis pipeline and created two R packages that allow for the study of data analysis code conducted in R. We present two packages, `matahari`, a package for logging everything that is typed in the R console or in an R script, and `tidycode`, a package with tools to allow for analyzing R calls in a tidy manner. These tools can be applied both to prospective studies, where a researcher can intentionally record code typed by participants, and retrospectively, where the researcher can retrospectively analyze code. We believe that these tools will help shape the next phase of reproducibility and replicability, allowing the analysis of code to inform data science pedagogy, examine how collaborates conduct data analyses, and explore how current software tools are being utilized.

Acknowledgements

We would like to extend a special thank you to the members of the Leek Lab at Johns Hopkins Bloomberg School of Public Health as well as volunteers who used the “classify” shiny application for helping classify R functions.

References

- Bryan, Jennifer, and Hadley Wickham. 2017. *Gh: ‘GitHub’ ‘Api’*. <https://CRAN.R-project.org/package=gh>.
- Buja, Andreas, Dianne Cook, Heike Hofmann, Michael Lawrence, Eun-Kyung Lee, Deborah F Swayne, and Hadley Wickham. 2009. “Statistical Inference for Exploratory Data Analysis and Model Diagnostics.” *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 367 (1906). The Royal Society Publishing: 4361–83. <https://doi.org/10.1098/rsta.2009.0120>.
- Dowle, Matt, and Arun Srinivasan. 2019. *Data.table: Extension of ‘Data.frame’*. <https://CRAN.R-project.org/package=data.table>.
- Goecks, Jeremy, Anton Nekrutenko, James Taylor, and Galaxy Team. 2010. “Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences.” *Genome Biology* 11 (8). <https://doi.org/10.1186/gb-2010-11-8-r86>.
- Ioannidis, John P A, Marcus R Munafo, Paolo Fusar-Poli, Brian A Nosek, and Sean P David. 2014. “Publication and other reporting biases in cognitive sciences: detection, prevalence, and prevention.” *Trends in Cognitive Sciences* 18 (5): 235–41. <https://doi.org/10.1016/j.tics.2014.02.010>.
- Leek, Jeffrey T, and Roger D Peng. 2015. “Opinion: Reproducible research can still be wrong: Adopting a prevention approach.” *Proceedings of the National Academy of Sciences* 112 (6): 1645–6. <https://doi.org/10.1073/pnas.1421412111>.
- Loy, Adam, Lendie Follett, and Heike Hofmann. 2016. “Variations of Q–Q Plots: The Power of Our Eyes!” *The American Statistician* 70 (2). Taylor & Francis: 202–14. <https://doi.org/10.1080/00031305.2015.1077728>.
- Loy, Adam, Heike Hofmann, and Dianne Cook. 2017. “Model Choice and Diagnostics for Linear Mixed-Effects Models Using Statistics on Street Corners.” *Journal of Computational and Graphical Statistics* 26 (3). Taylor & Francis: 478–92. <https://doi.org/10.1080/10618600.2017.1330207>.
- Majumder, Mahbubul, Heike Hofmann, and Dianne Cook. 2013. “Validation of Visual Statistical Inference, Applied to Linear Models.” *Journal of the American Statistical Association* 108 (503). Taylor & Francis Group: 942–56. <https://doi.org/10.1080/01621459.2013.808157>.
- McNutt, M. 2014. “Reproducibility.” *Science* 343 (6168): 229–29. <https://doi.org/10.1126/science.1250475>.
- Miguel, E, C Camerer, K Casey, J Cohen, K M Esterling, A Gerber, R Glennerster, et al. 2014. “Promoting Transparency in Social Science Research.” *Science* 343 (6166): 30–31. <https://doi.org/10.1126/science.1245317>.

- Nosek, B A, G Alter, G C Banks, D Borsboom, S D Bowman, S J Breckler, S Buck, et al. 2015. “Promoting an open research culture.” *Science* 348 (6242): 1422–5. <https://doi.org/10.1126/science.aab2374>.
- Peng, Roger D. 2011. “Reproducible Research in Computational Science.” *Science* 334 (6060): 1226–7. <https://doi.org/10.1126/science.1213847>.
- Richard, Blaustein. 2014. “Reproducibility Undergoes Scrutiny.” *BioScience* 64 (4): 368–68. <https://doi.org/https://doi.org/10.1093/biosci/biu017>.
- Sidi, Yulia, and Ofer Harel. 2018. “The treatment of incomplete data: Reporting, analysis, reproducibility, and replicability.” *Social Science & Medicine* 209 (July): 169–73. <https://doi.org/10.1016/j.socscimed.2018.05.037>.
- Silberzhan, Raphael, Eric L Uhlmann, Daniel P Martin, Pasquale Anselmi, Frederick Aust, Eli Awtrey, Štěpán Bahník, et al. 2018. “Many Analysts, One Data Set: Making Transparent How Variations in Analytic Choices Affect Results.” *Advances in Methods and Practices in Psychological Science* 1 (3). Sage Publications Sage CA: Los Angeles, CA: 337–56. <https://doi.org/10.1177/2515245917747646>.
- Silge, Julia, and David Robinson. 2016. “Tidytext: Text Mining and Analysis Using Tidy Data Principles in R.” *The Journal of Open Source Software* 1 (3): 37. <https://doi.org/10.21105/joss.00037>.
- . 2017. *Text Mining with R: A Tidy Approach.* ” O’Reilly Media, Inc.”
- Waltemath, Dagmar, and Olaf Wolkenhauer. 2016. “How Modeling Standards, Software, and Initiatives Support Reproducibility in Systems Biology and Systems Medicine.” *Ieee Transactions on Biomedical Engineering* 63 (10): 1999–2006. <https://doi.org/10.1109/TBME.2016.2555481>.
- Wickham, Hadley. 2015. *R Packages: Organize, Test, Document, and Share Your Code.* ” O’Reilly Media, Inc.”
- . 2019. *The Tidyverse Style Guide.* <https://style.tidyverse.org>.
- Wickham, Hadley, Mara Averick, Jennifer Bryan, Winston Chang, Lucy D’Agostino McGowan, Romain François, Garrett Grolemund, et al. 2019. “Welcome to the tidyverse.” *Journal of Open Source Software* 4 (43): 1686. <https://doi.org/10.21105/joss.01686>.
- Wickham, Hadley, Dianne Cook, and Heike Hofmann. 2015. “Visualizing Statistical Models: Removing the Blindfold.” *Statistical Analysis and Data Mining: The ASA Data Science Journal* 8 (4). Wiley Online Library: 203–25. <https://doi.org/10.1002/sam.11271>.
- Wickham, Hadley, and Garrett Grolemund. 2016. *R for Data Science: Import, Tidy, Transform, Visualize, and Model Data.* ” O’Reilly Media, Inc.”

Bibliography

Lucy D’Agostino McGowan
Wake Forest University
Winston-Salem, North Carolina, USA
lucydagostino@gmail.com

Sean Kross
UC San Diego
La Jolla, California, USA
seankross@ucsd.edu

Jeffrey Leek
Johns Hopkins Bloomberg School of Public Health
Baltimore, Maryland, USA
jtleek@gmail.com

spinifex: An R Package for Creating a Manual Tour of Low-dimensional Projections of Multivariate Data

by Nicholas Spyropoulos and Dianne Cook

Abstract Dynamic low-dimensional linear projections of multivariate data collectively known as *tours* provide an important tool for exploring multivariate data and models. The R package **tourr** provides functions for several types of tours: grand, guided, little, local and frozen. Each of these can be viewed dynamically, or saved into a data object for animation. This paper describes a new package, **spinifex**, which provides a manual tour of multivariate data where the projection coefficient of a single variable is controlled. The variable is rotated fully into the projection, or completely out of the projection. The resulting sequence of projections can be displayed as an animation, with functions from either the **plotly** or **ggridanimate** packages. By varying the coefficient of a single variable, it is possible to explore the sensitivity of structure in the projection to that variable. This is particularly useful when used with a projection pursuit guided tour to simplify and understand the solution. The use of the manual tour is applied particle physics data to illustrate the sensitivity of structure in a projection to specific variable contributions.

Introduction

Exploring multivariate spaces is a challenging task, increasingly so as dimensionality increases. Traditionally, static low-dimensional projections are used to display multivariate data in two dimensions including principal component analysis, linear discriminant spaces or projection pursuit. These are useful for finding relationships between multiple variables, but they are limited because they show only a glimpse of the high-dimensional space. An alternative approach is to use a tour (Asimov, 1985) of dynamic linear projections to look at many different low-dimensional projections. Tours can be considered to extend the dimensionality of visualization, which is important as data and models exist in more than 3D. The package **tourr** (Wickham et al., 2011) provides a platform for generating tours. It can produce a variety of tours, each paired with a variety of possible displays. A user can make a grand, guided, little, local or frozen tour, and display the resulting projected data as a scatterplot, density plot, histogram, or even as Chernoff faces if the projection dimension is more than 3.

This work adds a manual tour to the collection. The manual tour was described in Cook and Buja (1997) and allows a user to control the projection coefficients of a selected variable in a 2D projection. The manipulation of these coefficients allows the analyst to explore their sensitivity to the structure within the projection. As manual tours operate on only one variable at a time, they are particularly useful once a feature of interest has been identified.

One way to identify “interesting” features is with the use of a guided tour (Cook et al., 1995-09). Guided tours select a very specific path, which approaches a projection that optimizes an objective function. The optimization used to guide the tour is simulated annealing (Kirkpatrick et al., 1983). The direct optimization of a function allows guided tours to rapidly identify interesting projection features given the relatively large parameter-space. After a projection of interest is identified, an analyst can then use the “finer brush” of the manual tour to control the contributions of individual variables to explore the sensitivity they have on the structure visible in the projection.

The paper is organized as follows. Section 2 describes the algorithm used to perform a radial manual tour as implemented in the package **spinifex**. Section 2.2 explains how to generate an animation of the manual tour using the animation frameworks offered by **plotly** (Sievert, 2020) and **ggridanimate** (Pedersen and Robinson, 2020). Package functionality and code usage following the order applied in the algorithm follows in section 3.3. Section 4 illustrates how this can be used for sensitivity analysis applied to multivariate data collected on high energy physics experiments (Wang et al., 2018). Section 5 summarizes this paper and discusses potential future directions.

Algorithm

The algorithm to conduct a manual tour interactively, by recording mouse/cursor motion, is described in detail in Cook and Buja (1997). Movement can be in any direction and magnitude, but it can also be constrained in several ways:

- *radial*: fix the direction of contribution, and allow the magnitude to change.
- *angular*: fix the magnitude, and allow the angle or direction of the contribution to vary.
- *horizontal, vertical*: allow rotation only around the horizontal or vertical axis of the current 2D projection.

The algorithm described here produces a **radial** tour as an *animation sequence*. It takes the current contribution of the chosen variable, and using rotation brings this variable fully into the projection, completely removes it, and then returns to the original position.

Notation

The notation used to describe the algorithm for a 2D radial manual tour is as follows:

- \mathbf{X} , the data, an $n \times p$ numeric matrix to be projected to 2D.
- $\mathbf{B} = (B_1, B_2)$, any 2D orthonormal projection basis, $p \times 2$ matrix, describing the projection from $\mathbb{R}^p \Rightarrow \mathbb{R}^2$. This is called this the “original projection” because it is the starting point for the manual tour.
- k , is the index of the variable to manipulate, called the “manip var”.
- \mathbf{e} , a 1D basis vector of length p , with 1 in the k -th position and 0 elsewhere.
- \mathbf{M} is a $p \times 3$ matrix, defining the 3D subspace where data rotation occurs and is called the manip(ulation) space.
- \mathbf{R} , the 3D rotation matrix, for performing unconstrained 3D rotations within the manip space, \mathbf{M} .
- θ , the angle of in-projection rotation, for example, on the reference axes; c_θ, s_θ are its cosine and sine.
- ϕ , the angle of out-of-projection rotation, into the manip space; c_ϕ, s_ϕ are its cosine and sine. The initial value for animation purposes is ϕ_1 .
- \mathbf{U} , the axis of rotation for out-of-projection rotation orthogonal to \mathbf{e} .
- \mathbf{Y} , the resulting projection of the data through the manip space, \mathbf{M} , and rotation matrix, \mathbf{R} .

The algorithm operates entirely on projection bases and incorporates the data only when making the projected data plots, in light of efficiency.

Steps

Step 0) Set up

The flea data ([Lubischew \(1962\)](#)), available in the **tourr** package, is used to illustrate the algorithm. The data contains 74 observations on 6 variables, which are physical measurements made on flea beetles. Each observation belongs to one of three species.

An initial 2D projection basis must be provided. A suggested way to start is to identify an interesting projection using a projection pursuit guided tour. Here the holes index is used to find a 2D projection of the flea data, which shows three separated species groups. Figure 1 shows the initial projection of the data. The left panel displays the projection basis (\mathbf{B}) and can be used as a visual guide of the magnitude and direction that each variable contributes to the projection. The right panel shows the projected data, $\mathbf{Y}_{[n, 2]} = \mathbf{X}_{[n, p]} \mathbf{B}_{[p, 2]}$. The color and shape of points are mapped to the flea species. This plot is made using the `view_basis()` function in **spinifex**, which generates a `ggplot2` ([Wickham, 2016](#)) object.

Step 1) Choose manip variable

In figure 1 the contribution of the variables `tars1` and `aede2` mostly contrast the contribution of the other four variables. These two variables combined contribute in the direction of the projection where the purple cluster is separated from the other two clusters. The variable `aede2` is selected as the manip var, the variable to be controlled in the tour. The question that will be explored is: how important is this variable to the separation of the clusters in this projection?

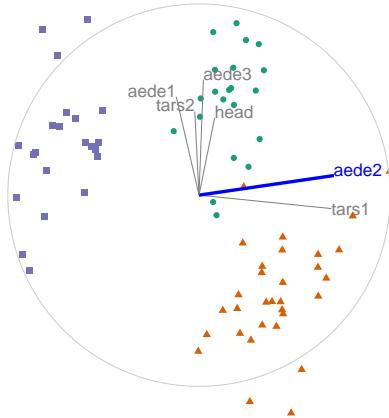


Figure 1: Initial 2D projection: representation of the basis (left) and resulting data projection (right) of standardized flea data. The color and shape of data points are mapped to beetle species. The basis was identified using a projection pursuit guided tour, with the holes index. The contribution of the variables *aede2* and *tars1* approximately contrasts the other variables. The visible structure in the projection are the three clusters corresponding to the three species. Produced with the function `view_basis()`.

Step 2) Create the 3D manip space

Initialize the coordinate basis vector as a zero vector, \mathbf{e} , of length p , and set the k -th element to 1. In the example data, *aede2* is the fifth variable in the data, so $k = 5$, set $e_5 = 1$. Use a Gram-Schmidt process to orthonormalize the coordinate basis vector on the original 2D projection to describe a 3D manip space, \mathbf{M} .

$$\begin{aligned} e_k &\leftarrow 1 \\ \mathbf{e}_{[p, 1]}^* &= \mathbf{e} - \langle \mathbf{e}, \mathbf{B}_1 \rangle \mathbf{B}_1 - \langle \mathbf{e}, \mathbf{B}_2 \rangle \mathbf{B}_2 \\ \mathbf{M}_{[p, 3]} &= (\mathbf{B}_1, \mathbf{B}_2, \mathbf{e}^*) \end{aligned}$$

The manip space provides a 3D projection from p -dimensional space, where the coefficient of the manip var can range completely between [0, 1]. This 3D space serves as the medium to rotate the projection basis relative to the selected manipulation variable. Figure 2 illustrates this 3D manip space with the manip var highlighted. This representation is produced by calling the `view_manip_space()` function. This diagram is purely used to help explain the algorithm.

Step 3) Defining a 3D rotation

The basis vector corresponding to the manip var (red line in Figure 2), can be operated like a lever anchored to the origin. This is the process of the manual control, that rotates the manip variable into and out of the 2D projection (Figure 3). As the variable contribution is controlled, the manip space rotates, and the projection onto the horizontal projection plane correspondingly changes. This is a manual tour. Generating a sequence of values for the rotation angles produces a path for the rotation of the manip space.

For a radial tour, fix θ , the angle describing rotation within the projection plane, and compute a sequence for ϕ , defining movement out of the plane. This will change ϕ from the initial value, ϕ_1 , the angle between \mathbf{e} and its shadow in \mathbf{B} , to a maximum of 0 (manip var fully in projection), then to a minimum of $\pi/2$ (manip var out of projection), before returning to ϕ_1 .

Rotations in 3D can be defined by the axes they pivot on. Rotation within the projection, θ , is rotation around the Z axis. Out-of-projection rotation, ϕ , is the rotation around an axis on the XY plane, \mathbf{U} , orthogonal to \mathbf{e} . Given these axes, the rotation matrix, \mathbf{R} can be written as follows, using Rodrigues' rotation formula (originally published in [Rodrigues \(1840\)](#)):

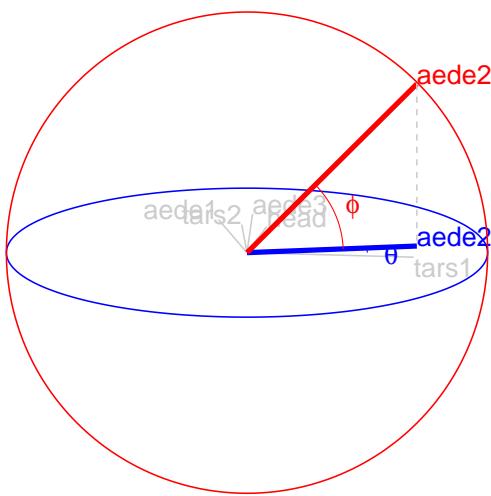


Figure 2: Illustration of a 3D manip space, this space is rotated effectively changing the contribution of the manip variable, `aede2` in the example data. The blue circle and variable map lies on the projection plane. The red circle, orthogonal to the projection plane, illustrates the manipulation space and how the manip var can be controlled and how this affects the variable contribution back onto the projection plane. The other variables are omitted from the manipulation dimension for simplicity. Picturing the other variables in that dimension reveals the intuition that rotating one variable performs a constrained rotation on the others. This is illustrated with the `view_manip_space()` function.

$$\begin{aligned}
 \mathbf{R}_{[3, 3]} &= \mathbf{I}_3 + s_\phi \mathbf{U} + (1 - c_\phi) \mathbf{U}^2 \\
 &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 0 & c_\phi s_\phi \\ 0 & 0 & s_\phi s_\phi \\ -c_\phi s_\phi & -s_\phi s_\phi & 0 \end{bmatrix} + \begin{bmatrix} -c_\phi(1 - c_\phi) & s_\phi^2(1 - c_\phi) & 0 \\ -c_\phi s_\theta(1 - c_\phi) & -s_\phi^2(1 - c_\phi) & 0 \\ 0 & 0 & c_\phi - 1 \end{bmatrix} \\
 &= \begin{bmatrix} c_\phi^2 + s_\phi^2 & -c_\phi s_\theta(1 - c_\phi) & -c_\phi s_\phi \\ -c_\phi s_\theta(1 - c_\phi) & s_\phi^2 c_\phi + c_\phi^2 & -s_\phi s_\phi \\ c_\phi s_\phi & s_\phi s_\phi & c_\phi \end{bmatrix}
 \end{aligned}$$

where

$$\begin{aligned}
 \mathbf{U} &= (u_x, u_y, u_z) = (s_\theta, -c_\theta, 0) \\
 &= \begin{bmatrix} 0 & -u_z & u_y \\ u_z & 0 & -u_x \\ -u_y & u_x & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & -c_\theta \\ 0 & 0 & -s_\theta \\ c_\theta & s_\theta & 0 \end{bmatrix}
 \end{aligned}$$

Step 4) Creating an animation of the radial rotation

The steps outlined above can be used to create any arbitrary rotation in the manip space. To use these for sensitivity analysis, the radial rotation is built into an animation where the manip var is rotated fully into the projection, completely out, and then back to the initial value. This involves allowing ϕ to vary between 0 and $\pi/2$, call the steps ϕ_i .

```

#> function (`_class` = NULL, `_inherit` = NULL, ...)
#> {
#>   e <- new.env(parent = emptyenv())
#>   members <- list(...)
#>   if (length(members) != sum(nzchar(names(members)))) {

```

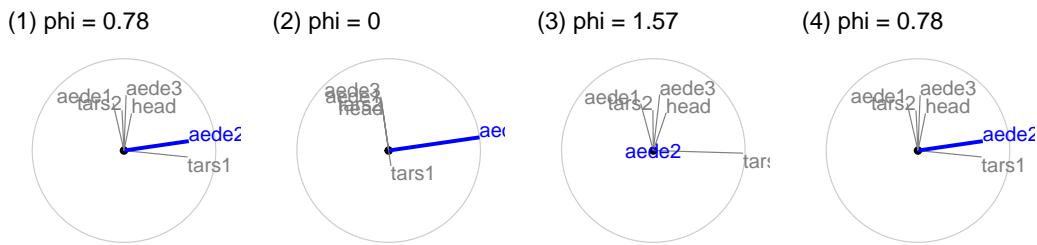


Figure 3: Snapshots of a radial manual tour manipulating aede2: (1) original projection, (2) full contribution, (3) zero contribution, (4) back to original.

```
#>      abort("All members of a ggproto object must be named.")
#> }
#> if (length(members) > 0) {
#>   list2env(members, envir = e)
#> }
#> `_inherit` <- substitute(`_inherit`)
#> env <- parent.frame()
#> find_super <- function() {
#>   eval(`_inherit`, env, NULL)
#> }
#> super <- find_super()
#> if (!is.null(super)) {
#>   if (!is.ggproto(super)) {
#>     abort("`_inherit` must be a ggproto object.")
#>   }
#>   e$super <- find_super
#>   class(e) <- c(`_class`, class(super))
#> }
#> else {
#>   class(e) <- c(`_class`, "ggproto", "gg")
#> }
#> e
#> }
#> <bytecode: 0x7f9c6f227828>
#> <environment: namespace:ggplot2>
```

1. Set initial value of ϕ_1 and θ : $\phi_1 = \cos^{-1} \sqrt{B_{k1}^2 + B_{k2}^2}$, $\theta = \tan^{-1} \frac{B_{k2}}{B_{k1}}$. Where ϕ_1 is the angle between e and its shadow in \mathbf{B} .
2. Set an angle increment (Δ_ϕ) that sets the step size for the animation, to rotate the manip var into and out of the projection. (Note: Using angle increment, rather than a number of steps, to control the movement, is consistent with the tour algorithm as implemented in the tour).
3. Step towards 0, where the manip var is completely in the projection plane.
4. Step towards $\pi/2$, where the manip variable has no contribution to the projection.
5. Step back to ϕ_1 .

In each of the steps 3-5, a small step may be added to ensure that the endpoints of ϕ ($0, \pi/2, \phi_1$) are reached.

Step 5) Projecting the data

The operation of a manual tour is defined on the projection bases. Only when the data plot needs to be made is the data projected into the relevant basis.

$$\mathbf{Y}_{[n, 3]}^{(i)} = \mathbf{X}_{[n, p]} \mathbf{M}_{[p, 3]} \mathbf{R}_{[3,3]}^{(i)}$$

where $\mathbf{R}_{[3,3]}^{(i)}$ is the incremental rotation matrix, using ϕ_i . To make the data plot, use the first two columns of \mathbf{Y} . Show the projected data for each frame in sequence to form an animation.

Table 1: Summary of available functions.

Type	Function	Description
construction	create_manip_space	forms the 3D space of rotation
construction	rotate_manip_space	performs 3D rotation
construction	manual_tour	generates sequence of 2D frames
render	array2df	turn the tour path array into long form, for plotting
render	render_	render long form as a ggplot2 object for animation
render	render_plotly	render the animation as a plotly object (default)
render	render_gganimate	render the animation as a gganimate object
animation	play_tour_path	composite function animating the specified tour path
animation	play_manual_tour	composite function animating the specified manual tour
specialty	print_manip_space	table of the rotated basis and manip space
specialty	oblique_frame	display the reference axes of a given basis
specialty	view_manip_space	illustrative display of any manip space

Figure 4 illustrates a manual tour sequence having 15 steps. The projection axes are displayed on the top half, which corresponds to the projected data in the bottom half. When aede2 is removed from the projection, the purple cluster overlaps with the green cluster. This suggests that aede2 is important for distinguishing between these species.

Tours are typically viewed as an animation. The animation of this tour can be viewed online at https://github.com/nspyrison/spinifex_paper/blob/master/paper/gifs/flea_radialtour_mvar5.gif. The page may take a moment to load. Animations can be produced using the function `play_manual_tour()`.

Package structure and functionality

This section describes the functions available in the package, and how to use them.

Installation

The `spinifex` is available from CRAN, and can be installed by:

```
## Install from CRAN
install.package("spinifex")
## Load into session
library("spinifex")
```

Also see the shiny app for understandign and the vignette for basic usage:

```
## Shiny app for visualizing basic application
run_app("intro")
## View the code vignette
vignette("spinifex_vignette")
```

The development version can be installed from github:

```
## Optionally install latest developmentent version from GitHub
remotes::install_github("nspyrison/spinifex")
```

Functions

Table 1 lists the primary functions and their purpose. These are grouped into four types: construction for building a tour path, render to make the plot objects, animation for running the animation, and specialty for providing illustrations used in the algorithm description.

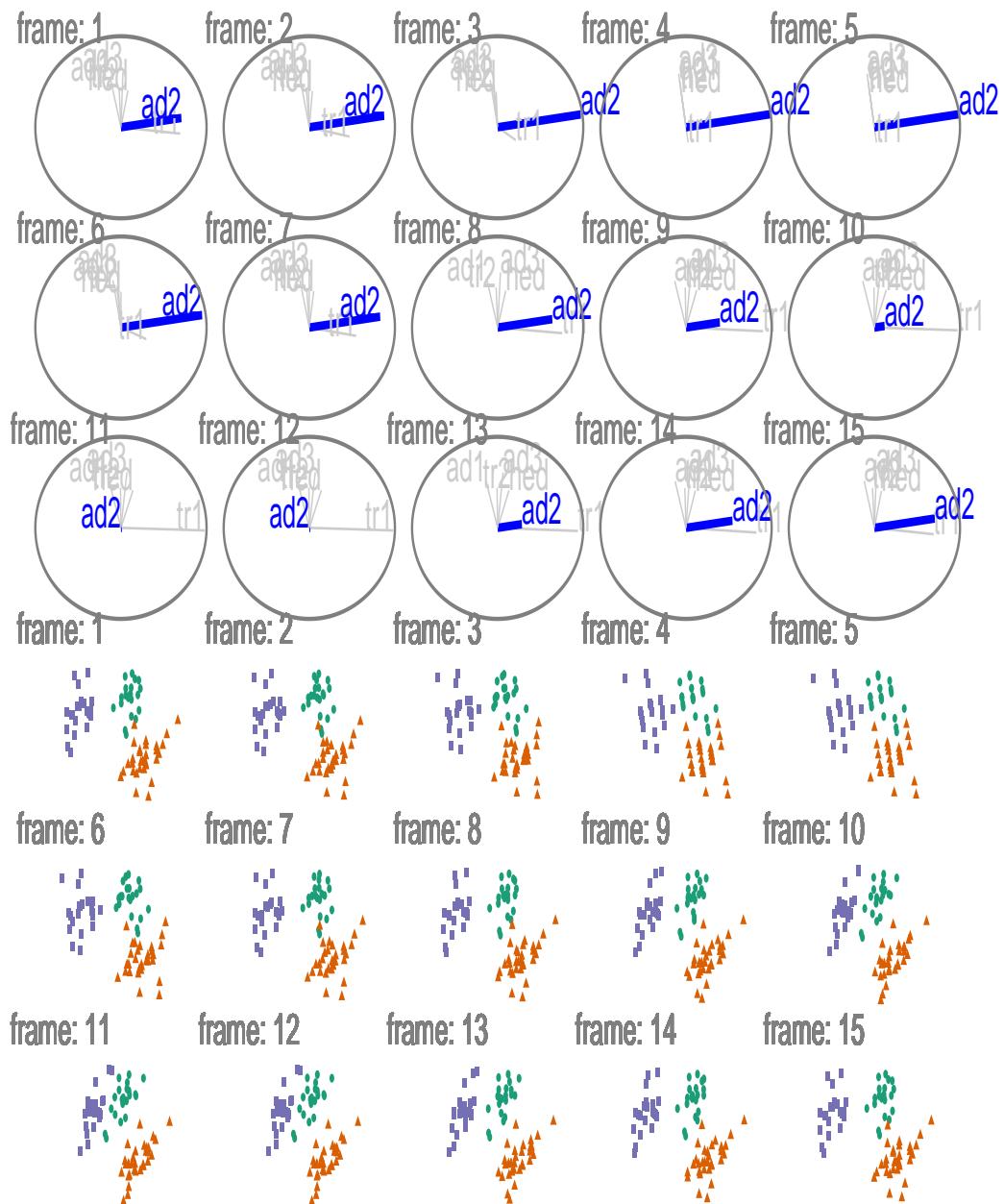


Figure 4: Radial manual tour manipulating *aede2* of standardized flea data. The axis for *aede2* increases in contribution to the projection, from its initial value to 1, decreasing to 0 and then returning to the initial value. This effects the separation between the purple and green clusters. This shows that *aede2* is important for distinguishing the purple species, because the separation disappears when *aede2* is not contributing to the projection.

Usage

Using the `flea` data from the `tourr` package, we will illustrate generating a manual tour to explore the sensitivity of the cluster structure is to the variable `aede2`.

```
library(spinifex)
## Standardized flea data
f_data <- tourr::rescale(flea[, 1:6])
## Guided tour path, holes index
f_path <- save_history(f_data, guided_tour(holes()))
## Local extrema found
f_basis <- matrix(f_path[,, max(dim(f_path)[3])], ncol=2)
## Categorical class variable
f_clas <- factor(flea$species)
## Manip var, number of the variable to alter
f_mvar <- 5
## Angular dist between frames (radians)
step_size <- .26
## Render and play animate, as plotly object by default
play_manual_tour(data = f_data,
                   basis = f_basis,
                   manip_var = f_mvar,
                   angle = step_size,
                   col = f_clas,
                   pch = f_clas)
```

The `play_manual_tour()` function is a composite function handling interaction between `manual_tour()`, `array2df()`, and `render_plotly()`. This will generate an html animation using `plotly`. Switching the renderer to `render_gganimate()` alternatively creates an animated gif. Each of these formats allows for the animation to be made available on a web site, or directly visible in an html formatted document.

Making illustrations

The function `oblique_frame` can be used to show a projection of the basis, or with the data overlaid. For example, the plots in Figures 1 and 3 were made with code similar to this:

```
## View a basis and projected data
oblique_frame(basis = f_basis,
               data = f_data,
               color = f_clas,
               shape = f_clas)
```

An illustration of the manip space (as shown in Figure 2) is made with the `view_manip_space` function, as follows:

```
## Displays the projection plane and manipulation space for the
view_manip_space(basis = f_basis,
                  manip_var = f_mvar,
                  lab = colnames(f_data))
```

Application

Wang et al. (2018) introduces a new tool, PDFSense, to visualize the sensitivity of hadronic experiments to nucleon structure. The parameter-space of these experiments lies in 56 dimensions, $\delta \in \mathbb{R}^{56}$, and are visualized as 3D subspaces of the 10 first principal components in linear (PCA) and non-linear (t-SNE) embeddings.

Cook et al. (2018) illustrates how to learn more about the structures using a grand tour. Tours can better resolve the shape of clusters, intra-cluster detail, and better outlier detection than PDFSense & TFEP (TensorFlow embedded projections) or traditional static embeddings. This example builds from here, illustrating how the manual tour can be used to examine the sensitivity of structure in a projection to different parameters. The specific 2D projections passed to the manual tour were provided in their work.

The data has a hierarchical structure with top-level clusters; DIS, VBP, and jet. Each cluster is a particular class of experiments, each with many experimental datasets which, each have many observations of their own. In consideration of data density, we conduct manual tours on subsets of the DIS and jet clusters. This explores the sensitivity of the structure to each of the variables in turn and we present the subjectively best and worst variable to manipulate for identifying dimensionality of the clusters and describing the span of the clusters.

Jet cluster

The jet cluster resides in a smaller dimensionality than the full set of experiments with four principal components explaining 95% of the variation in the cluster (Cook et al., 2018). The data within this 4D embedding is further subsetted, to ATLAS7old and ATLAS7new, to focus on two groups that occupy different parts of the subspace. Radial manual tours controlling contributions from PC4 and PC3 are shown in Figures 5 and 6, respectively. The difference in shape can be interpreted as the experiments probing different phase-spaces. Back-transforming the principal components to the original variables can be done for a more detailed interpretation.

When PC4 is removed from the projection (Figure 5) the difference between the two groups is removed, indicating that it is important for distinguishing experiments. However, removing PC3 from the projection (Figure 6) does not affect the structure, indicating it is not important for distinguishing experiments. Animations for the remaining PCs can be viewed at the following links: [PC1](#), [PC2](#), [PC3](#), and [PC4](#). It can be seen that only PC4 is important for viewing the difference in these two experiments.

DIS cluster

Following Cook et al. (2018), to explore the DIS cluster, PCA is recomputed and the first six principal components, explaining 48% of the full sample variation, are used. The contributions of PC6 and PC2 are explored in Figures 7 and 8, respectively. Three experiments are examined: DIS HERA1+2 (green), dimuon SIDIS (purple), and charm SIDIS (orange).

Both PC2 and PC6 contribute to the projection similarly. When PC6 is rotated into the projection, variation in the DIS HERA1+2 is greatly reduced. When PC2 is removed from the projection, dimuon SIDIS becomes more clearly distinct. Even though both variables contribute similarly to the original projection, their contributions have quite different effects on the structure of each cluster, and the distinction between clusters. Animations of all of the principal components can be viewed from the links: [PC1](#), [PC2](#), [PC3](#), [PC4](#), [PC5](#), and [PC6](#).

Discussion

Dynamic linear projections of numeric multivariate data, tours, play an important role in data visualization; they extend the dimensionality of visuals to peek into high-dimensional data and parameter spaces. This research has taken the manual tour algorithm, specifically the radial rotation, used in GGobi (Swayne et al., 2003-08-28) to interactively rotate a variable into or out of a 2D projection, and modified it to create an animation that performs the same task. It is most useful for examining the importance of variables, and how the structure in the projection is sensitive or not to specific variables. This functionality available in package **spinfex**. The work complements the methods available in the **tourr** package.

This work was motivated by problems in physics, and thus the usage was illustrated on data comparing experiments of hadronic collisions, to explore the sensitivity of cluster structure to different principal components. These tools can be applied quite broadly to many multivariate data analysis problems.

The manual tour is constrained in the sense that the effect of one variable is dependent on the contributions of other variables in the manip space. However, this can be useful to simplify a projection by removing variables without affecting the visible structure. Defining a manual rotation in high dimensions is possible using Givens rotations and Householder reflections as outlined in Buja et al. (2005). This would provide more flexible manual rotation, but more difficult for a user because they have the choice (too much choice) of which directions to move.

Another future research topic could be to extend the algorithm for use on 3D projections. With the current popularity and availability of 3D virtual displays, this may benefit the detection and understanding of the higher dimensional structure, or enable the examination of functions.

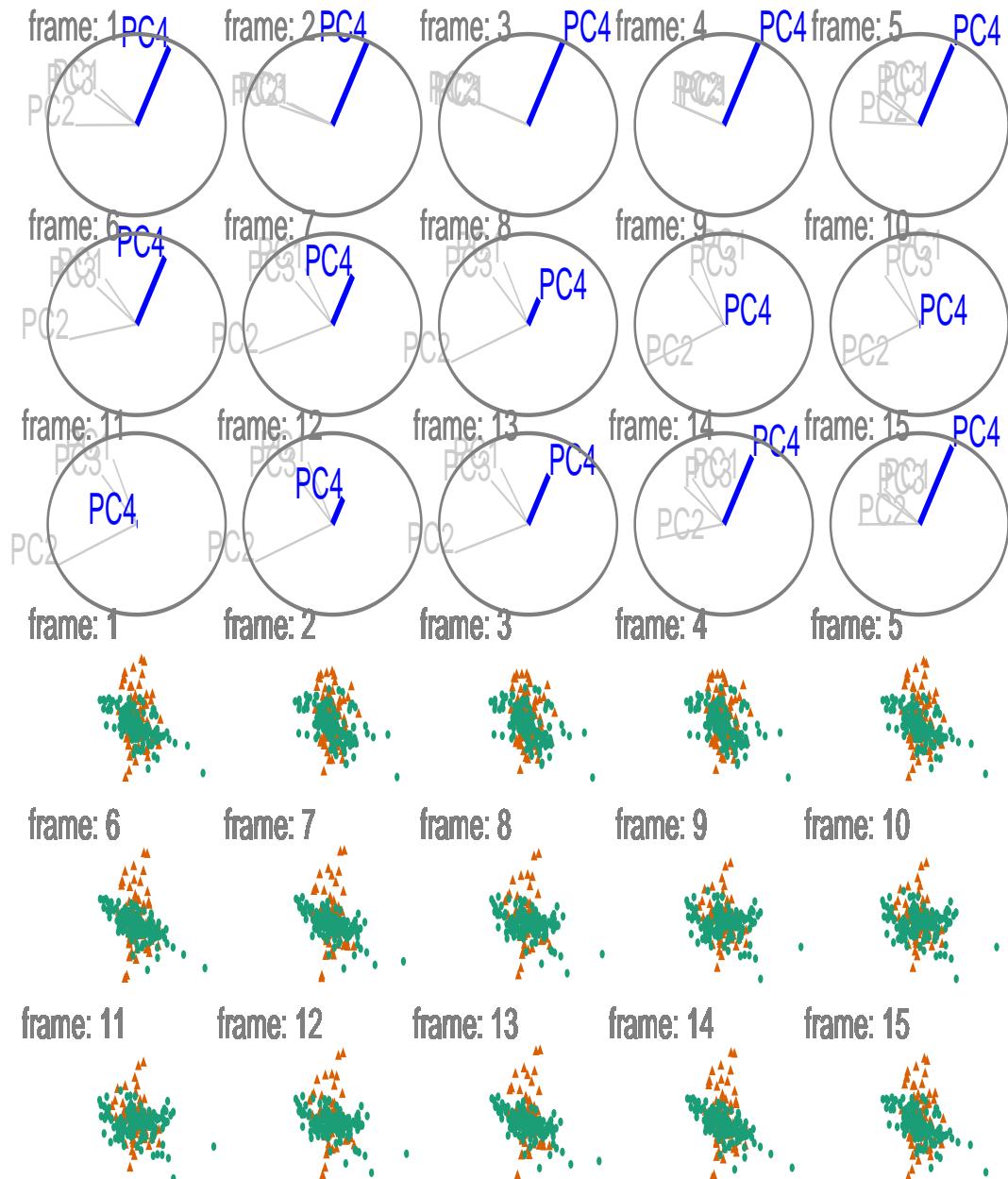


Figure 5: Snapshots of a radial manual tour of PC4 within the jet cluster, with color indicating experiment type: ATLAS7new (green) and ATLAS7old (orange). When PC4 is removed from the projection (frame 10) there is little difference between the groups, suggesting that PC4 is important for distinguishing the experiments.

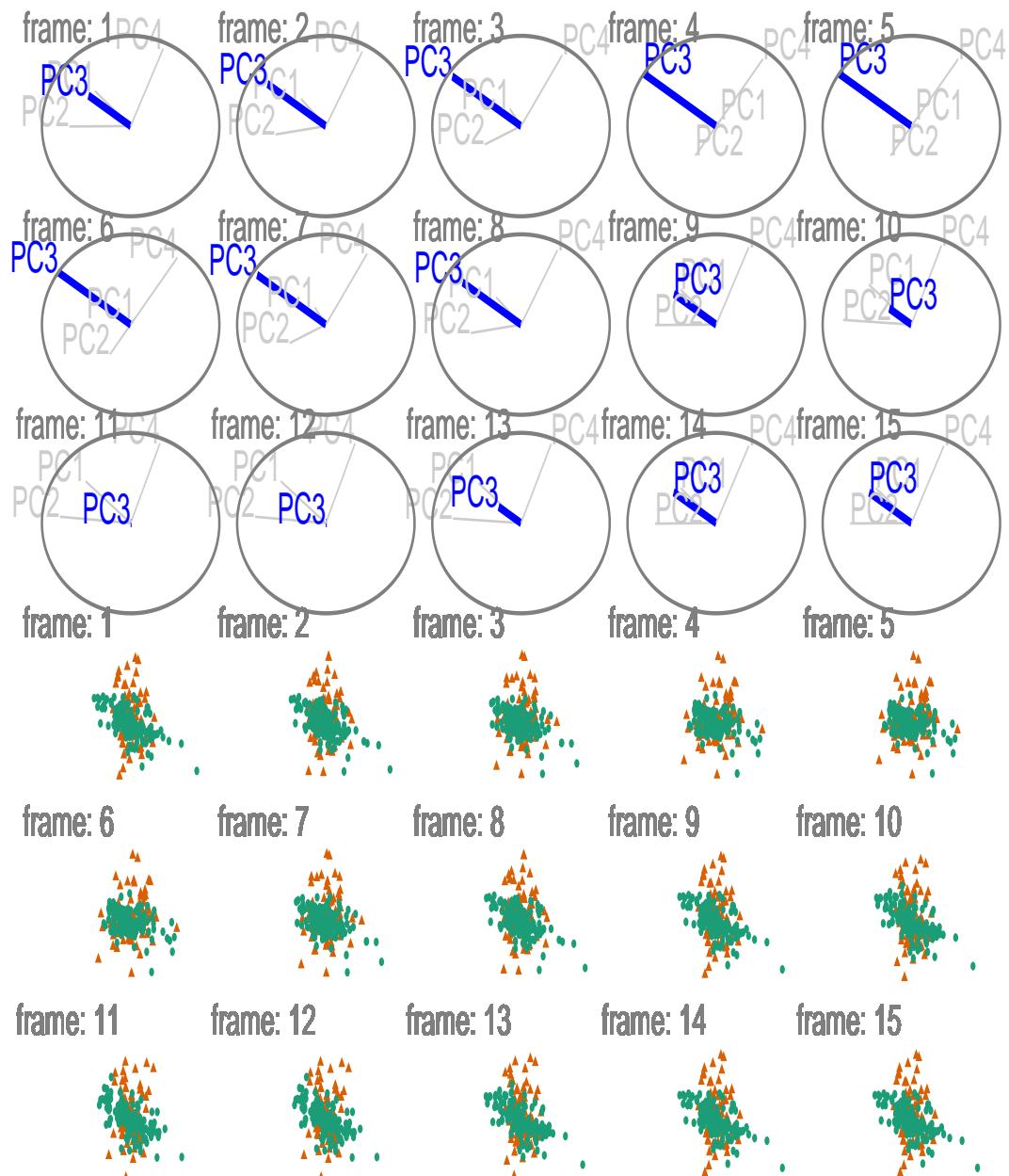


Figure 6: Snapshots of a radial manual tour of PC3 within the jet cluster, with color indicating experiment type: ATLAS7new (green) and ATLAS7old (orange). When the contribution from PC3 is changed there is little change to the structure of the two groups, suggesting that PC3 is not important for distinguishing the experiments.

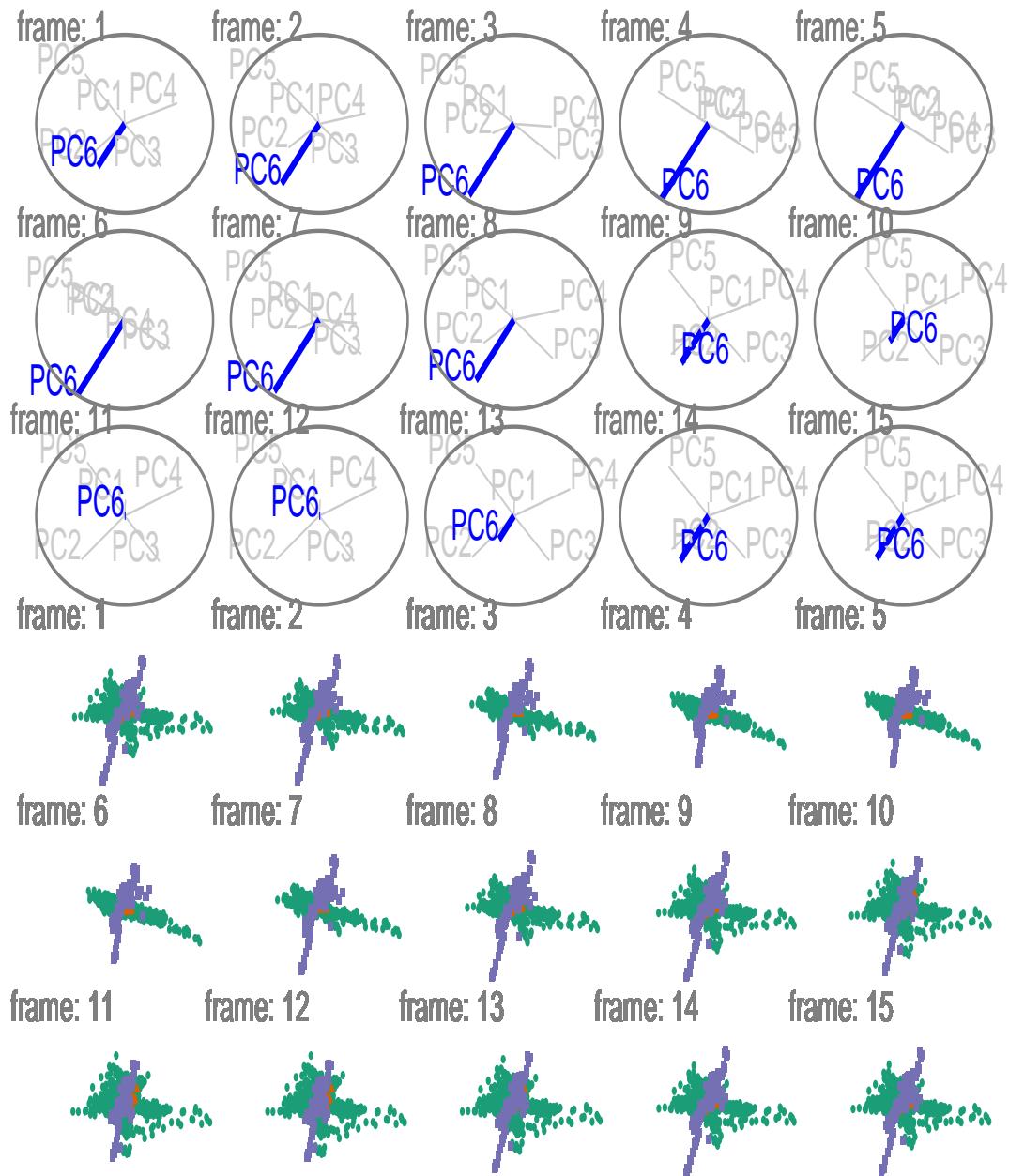


Figure 7: Snapshots of a radial manual tour exploring the sensitivity PC6 has on the structure of the DIS cluster, with color indicating experiment type: DIS HERA1+2 (green), dimuon SIDIS (purple), and charm SIDIS (orange). DIS HERA1+2 is distributed in a cross-shaped plane, charm SIDIS occupies the center of this cross, and dimuon SIDIS is a linear cluster crossing DIS HERA1+2. As the contribution of PC6 is increased, DIS HERA1+2 becomes almost singular in one direction (frame 5), indicating that this experiment has very little variability in the direction of PC6.

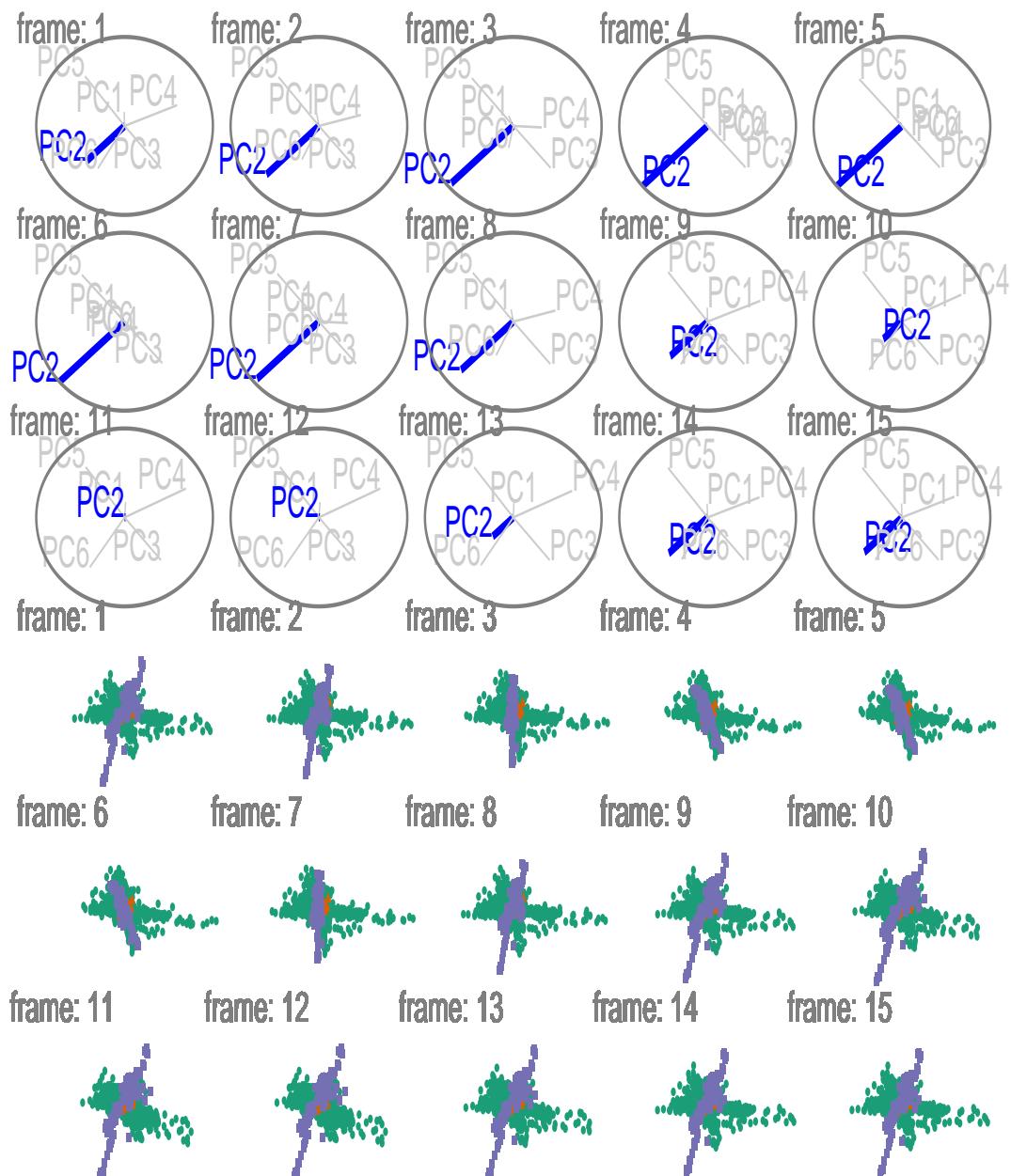


Figure 8: Snapshots of a radial manual tour exploring the sensitivity PC2 to the structure of the DIS cluster, with color indicating experiment type: DIS HERA1+2 (green), dimuon SIDIS (purple), and charm SIDIS (orange). As contribution from PC2 is decreased, dimuon SIDIS becomes more distinguishable from the other two clusters (frames 10-14), indicating that in its absence PC2 is important.

Having a graphical user interface would be useful for making it easier and more accessible to a general audience. This is possible to implement using `shiny` (Chang et al., 2020). The primary purposes of the interface would be to allow the user to interactively change the `manip` variable easily, and the interpolation step for more or less detailed views.

Acknowledgments

This article was created in R, using `knitr` (Xie, 2020) and `rmarkdown` (Allaire et al., 2020), with code generating the examples inline. The source files for this article be found at github.com/nspyrison/spinifex_paper/. The animated gifs can also be viewed at this site, and also in the supplementary material for this paper. The source code for the `spinifex` package can be found at github.com/nspyrison/spinifex/.

Bibliography

- J. J. Allaire, Y. Xie, J. McPherson, J. Luraschi, K. Ushey, A. Atkins, H. Wickham, J. Cheng, W. Chang, and R. Iannone. `rmarkdown`: Dynamic documents for r, 2020. URL <https://github.com/rstudio/rmarkdown>. [p]
- D. Asimov. The grand tour: a tool for viewing multidimensional data. *SIAM journal on scientific and statistical computing*, 6(1):128–143, 1985. doi: <https://doi.org/10.1137/0906011>. [p]
- A. Buja, D. Cook, D. Asimov, and C. Hurley. Computational methods for high-dimensional rotations in data visualization. In *Handbook of Statistics*, volume 24, pages 391–413. Elsevier, 2005. ISBN 978-0-444-51141-6. doi: [10.1016/S0169-7161\(04\)24014-7](https://doi.org/10.1016/S0169-7161(04)24014-7). URL <http://linkinghub.elsevier.com/retrieve/pii/S0169716104240147>. [p]
- W. Chang, J. Cheng, J. J. Allaire, Y. Xie, and J. McPherson. `shiny`: Web application framework for r, 2020. URL <https://CRAN.R-project.org/package=shiny>. [p]
- D. Cook and A. Buja. Manual controls for high-dimensional data projections. *Journal of Computational and Graphical Statistics*, 6(4):464–480, 1997. ISSN 1061-8600. doi: [10.2307/1390747](https://doi.org/10.2307/1390747). URL <http://www.jstor.org/stable/1390747>. [p]
- D. Cook, A. Buja, J. Cabrera, and C. Hurley. Grand tour and projection pursuit. *Journal of Computational and Graphical Statistics*, 4(3):155, 1995-09. ISSN 10618600. doi: [10.2307/1390844](https://doi.org/10.2307/1390844). URL <https://www.jstor.org/stable/1390844?origin=crossref>. [p]
- D. Cook, U. Laa, and G. Valencia. Dynamical projections for the visualization of PDFSense data. *Eur. Phys. J. C*, 78(9):742, 2018. doi: [10.1140/epjc/s10052-018-6205-2](https://doi.org/10.1140/epjc/s10052-018-6205-2). [p]
- S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *science*, 220 (4598):671–680, 1983. doi: [10.1126/science.220.4598.671](https://doi.org/10.1126/science.220.4598.671). [p]
- A. A. Lubischew. On the use of discriminant functions in taxonomy. *Biometrics*, pages 455–477, 1962. doi: [10.2307/2527894](https://doi.org/10.2307/2527894). [p]
- T. L. Pedersen and D. Robinson. `gganimate`: A grammar of animated graphics, 2020. URL <https://CRAN.R-project.org/package=gganimate>. [p]
- O. Rodrigues. Des lois géométriques qui régissent les déplacements d'un système solide dans l'espace: et de la variation des coordonnées provenant de ces déplacements considérés indépendamment des causes qui peuvent les produire. *Journal de Mathématiques Pures et Appliquées*, 5:380–440, 1840. [p]
- C. Sievert. *Interactive Web-Based Data Visualization with R, plotly, and shiny*. Chapman and Hall/CRC, 2020. ISBN 978-1-138-33145-7. URL <https://plotly-r.com>. [p]
- D. F. Swayne, D. T. Lang, A. Buja, and D. Cook. GGobi: evolving from XGobi into an extensible framework for interactive data visualization. *Computational Statistics & Data Analysis*, 43(4):423–444, 2003-08-28. ISSN 0167-9473. doi: [10.1016/S0167-9473\(02\)00286-4](https://doi.org/10.1016/S0167-9473(02)00286-4). URL <http://www.sciencedirect.com/science/article/pii/S0167947302002864>. [p]
- B.-T. Wang, T. J. Hobbs, S. Doyle, J. Gao, T.-J. Hou, P. M. Nadolsky, and F. I. Olness. Mapping the sensitivity of hadronic experiments to nucleon structure. *Physical Review D*, 98(9):094030, 2018. doi: [10.1103/PhysRevD.98.094030](https://doi.org/10.1103/PhysRevD.98.094030). [p]

- H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. ISBN 978-3-319-24277-4. URL <https://ggplot2.tidyverse.org>. [p]
- H. Wickham, D. Cook, H. Hofmann, and A. Buja. tourr: An r package for exploring multivariate data with projections. *Journal of Statistical Software*, 40(2), 2011. ISSN 1548-7660. doi: 10.18637/jss.v040.i02. URL <http://www.jstatsoft.org/v40/i02/>. [p]
- Y. Xie. knitr: A general-purpose package for dynamic report generation in r, 2020. URL <https://yihui.org/knitr/>. [p]

Nicholas Spyris
Monash University
Faculty of Information Technology

nicholas.spyrison@monash.edu

Dianne Cook
Monash University
Department of Econometrics and Business Statistics

dcook@monash.edu

SimilaR: R Code Clone and Plagiarism Detection

by Maciej Bartoszuk and Marek Gagolewski

Abstract Third-party software for assuring source code quality is becoming increasingly popular. Tools that evaluate the coverage of unit tests, perform static code analysis, or inspect run-time memory use are crucial in the software development life cycle. More sophisticated methods allow for performing meta-analyses of large software repositories, e.g., to discover abstract topics they relate to or common design patterns applied by their developers. They may be useful in gaining a better understanding of the component interdependencies, avoiding cloned code as well as detecting plagiarism in programming classes.

A meaningful measure of similarity of computer programs often forms the basis of such tools. While there are a few noteworthy instruments for similarity assessment, none of them turns out particularly suitable for analysing R code chunks. Existing solutions rely on rather simple techniques and heuristics and fail to provide a user with the kind of sensitivity and specificity required for working with R scripts. In order to fill this gap, we propose a new algorithm based on a Program Dependence Graph, implemented in the **SimilaR** package. It can serve as a tool not only for improving R code quality but also for detecting plagiarism, even when it has been masked by applying some obfuscation techniques or imputing dead code. We demonstrate its accuracy and efficiency in a real-world case study.

Introduction

In recent years there has been a rise in the availability of tools related to code quality, including inspecting run-time memory usage (Serebryany et al., 2012), evaluating unit tests coverage (Ammann and Offutt, 2013), discovering abstract topics to which source code is related (Grant et al., 2012; Tian et al., 2009; McBurney et al., 2014; Linstead et al., 2007; Maskeri et al., 2008), finding parts of code related to a particular bug submission (Lukins et al., 2008), and checking for similarities between programs. With regards to the latter, quantitative measures of similarity between source code chunks play a key role in such practically important areas as software engineering, where encapsulating duplicated code fragments into functions or methods is considered a good development practice or in computing education, where any cases of plagiarism should be brought to a tutor's attention, see (Misic et al., 2016; Mohd Noor et al., 2017; Roy et al., 2009; Rattan et al., 2013; Ali et al., 2011; Hage et al., 2011; Martins et al., 2014). Existing approaches towards code clone detection can be classified based on the abstraction level at which they inspect programs' listings.

- *Textual* – the most straightforward representation, where a listing is taken as-is, i.e., as raw text. Typically, string distance metrics (like the Levenshtein one; see, e.g., van der Loo, 2014) are applied to measure similarity between pairs of the entities tested. Then some (possibly approximate) nearest neighbour search data structures seek matches within a larger code base. Hash functions can be used for the same purpose, where fingerprints of code fragments might make the comparison faster, see, e.g., (Johnson, 1993; Manber, 1994; Rieger, 2005). Another noteworthy approach involves the use of Latent Semantic Analysis (Marcus and Maletic, 2001) for finding natural clusters of code chunks.
- *Lexical* (token-based) – where a listing is transformed into tokens, which are generated by the parser during the lexical analysis stage. This form is believed to be more robust than the textual one, as it is invariant to particular coding styles (indentation, layout, comments, etc.). Typically, algorithms to detect and analyse common token sub-sequences are used (Kamiya et al., 2002; Ueda et al., 2002; Li et al., 2006; Wise, 1992; Prechelt et al., 2000; Hummel et al., 2011; Schleimer et al., 2003).
- *Syntactic* – an approach based on abstract syntax trees (ASTs). Contrary to the previous representation, which is “flat” by its nature, here the code is represented in a hierarchical manner. This makes it possible to, for instance, distinguish between a top-level statement and a code block that is executed *within* the `else` clause of an `if-else` construct. One way to quantify the similarity of ASTs is to find common sub-trees. This might be an uneasy task, therefore, e.g., in (Baxter et al., 1998) a hash function is used to project a sub-tree into a discrete single value, so that only the sub-trees in the same bucket are compared against each other. Another way involves the computation of diverse tree metrics (based on node type count, their distribution, connectivity, etc.) so that each AST is represented as a feature vector. Then the feature vectors can be compared against each other directly in a pairwise

fashion (Mayrand et al., 1996; Patenaude et al., 1999; Fu et al., 2017) or by means of some cluster analysis-based approach (Jiang et al., 2007).

- *Semantic* – the most sophisticated representation involving a set of knowledge-based, language-dependent transformations of a program’s abstract syntax tree. Usually, a data structure commonly known as a Program Dependence Graph (PDG) is created, see below for more details. In such a data structure, the particular order of (control- or data-) *independent* code lines is negligible. A popular approach to measure similarity between a pair of PDGs concerns searching for (sub)isomorphisms of the graphs, see (Komondoor and Horwitz, 2001; Liu et al., 2006; Qu et al., 2014).

There are a few generally available software solutions whose purpose is to detect code clones, e.g., MOSS (see <http://theory.stanford.edu/~aiken/moss/> and Schleimer et al., 2003) and JPlag (see <http://jplag.de/> and Prechelt et al., 2000), see also (Misic et al., 2016; Vandana, 2018) for an overview. These tools are quite generic, offering built-in support for popular programming languages such as Java, C#, C++, C, or Python.

Unfortunately, there is no package of this kind that natively supports the R language, which is the GNU version of S (see, e.g., Becker et al., 1998; Venables and Ripley, 2000). It is a serious gap: R is amongst the most popular languages¹, and its use has a long, successful track record, particularly with respect to all broadly-conceived statistical computing, machine learning, and other data science activities (Wickham and Gromlund, 2017). With some pre-processing, MOSS and JPlag can be applied on R code chunks, but the accuracy of code clones detection is far from optimal. This is due to the fact that, while at a first glance being an imperative language, R allows plenty typical functional constructs (see the next section for more details and also, e.g., Chambers, 1998; Wickham, 2014; Chambers, 2008). On the one hand, its syntax resembles that of the C language, with curly braces to denote a nested code block and classical control-flow expressions such as `if..else` conditionals, or `while` and `for` (for each) loops. On the other hand, R’s semantics is based on the functional Scheme language (Abelson et al., 1996), which is derived from Lisp. Every expression (even one involving the execution of a `for` loop) is in fact a call to a function or any combination thereof, and each function is a first-class object that (as a rule of thumb) has no side effects. Moreover, users might choose to prefer applying *Map–Filter–Reduce*-like expressions on container objects instead of the classical control-flow constructs or even mix the two approaches. Also, the possibility of performing the so-called *nonstandard evaluation* (metaprogramming) allows to change the meaning of certain expressions during run-time. For instance, the popular forward-pipe operator, `%>%`, implemented in the `magrittr` (Bache and Wickham, 2014) package, allows for converting a pipeline of function calls to a mutually nested series of calls.

In this paper we describe a new algorithm that aims to fill the aforementioned gap (based on Bartoszuk, 2018). The method’s implementation is included in the `SimilaR`² package. It transforms the analysed code base into a Program Dependence Graph that takes into account the most common R language features as well as the most popular development patterns in data science. Due to this, the algorithm is able to detect cases of plagiarism quite accurately. Moreover, thanks to a novel, polynomial-time approximate graph comparison algorithm, its implementation has relatively low run-times. This enables to conduct an analysis of a software repository whose size is significant.

This paper is set out as follows. First we introduce the concept of a *Program Dependence Graph* along with its R language-specific customisations. Then we depict a novel algorithm for quantifying similarity of two graphs. Further on we provide some illustrative examples for the purpose of showing the effects of applying particular alterations to a Program Dependence Graph. What is more, we demonstrate the main features of the `SimilaR` package version 1.0.8. Then we perform an experiment involving the comparison of the complete code-base of two CRAN packages.

Program Dependence Graph

A *Program Dependence Graph* (PDG) is a directed graph representing various relations between individual expressions in a source code chunk. As we mentioned in the introduction, it is among the most sophisticated data structures used for the purpose of code clones detection. First proposed by Ferrante et al. (1987), it forms the basis of many algorithms, see, e.g., (Liu et al., 2006; Qu et al., 2014; Gabel et al., 2008; Krinke, 2001; Horwitz and Reps, 1991; Komondoor and Horwitz, 2001; Ghosh and Lee, 2018; Nasirloo and Azimzadeh, 2018).

¹For instance, the 2018 edition of the *IEEE Spectrum* ranking places R on the No. 7 spot, see <http://spectrum.ieee.org/at-work/innovation/the-2018-top-programming-languages>.

²See <https://CRAN.R-project.org/package=SimilaR>. `SimilaR` can be downloaded from the Comprehensive R Archive Network (CRAN) repository (Silge et al., 2018) and installed via a call to `install.packages("SimilaR")`.

Abstract Syntax Tree. To create a PDG, we first need to construct an *Abstract Syntax Tree* (AST) of a given program. In R, it is particularly easy to compute the AST corresponding to any expression, due to its built-in support for *reflection* that facilitates metaprogramming. For instance, the `parse()` function can be called to perform lexical analysis of a code fragment, yielding a sequence of language objects. Moreover, a basic version of a function to print an AST takes just few lines of code:

```
R> show_ast <- function(x) {
+   as.list_deep <- function(x) # convert to a plain list (recursively)
+   { if (is.call(x)) lapply(as.list(x), as.list_deep) else x }
+   x <- substitute(x) # expression that generated the argument
+   str(as.list_deep(x)) # pretty-print
+ }
```

Let us visualise the AST corresponding to expression `d <- sum((x-y)*(x-y))`.

```
R> show_ast(d <- sum((x-y)*(x-y)))
```

```
List of 3
$ : symbol <-
$ : symbol d
$ :List of 2
..$ : symbol sum
..$ :List of 3
...$ : symbol *
...$ :List of 2
....$ : symbol (
....$ :List of 3
.....$ : symbol -
.....$ : symbol x
.....$ : symbol y
....$ :List of 2
....$ : symbol (
....$ :List of 3
....$ : symbol -
....$ : symbol x
....$ : symbol y
```

In R, both a constant (numeric, logical, string, etc.) and a symbol (name) constitute what we call a *simple expression*. A *compound expression* is in turn a sequence of $n + 1$ expressions (simple or compound ones) $\langle f, a_1, \dots, a_n \rangle$, $n \geq 0$, which represents a call to f with arguments a_1, \dots, a_n (which we are typically used to denote as $f(a_1, \dots, a_n)$).

The above AST can be written in the Polish (prefix) notation as

$$\langle \text{'<-'}, \text{d}, \langle \text{sum}, \langle \text{'*'}, \langle \text{'-'}, \text{x}, \text{y} \rangle, \langle \text{'-'}, \text{x}, \text{y} \rangle \rangle \rangle \rangle$$

Such a notation is used in Scheme and Lisp; we skipped a call to ‘(’ for readability, as `(e)` is equivalent to `e` for each expression `e`. Alternatively, the above can be written as

$$\text{'<-'}(\text{d}, \text{sum}(\text{'*'}(\text{'-'}(\text{x}, \text{y}), \text{'-'}(\text{x}, \text{y}))))$$

in the “functional” form. Let us emphasise that even an application of a binary operator (here: `<-`, `*`, and `-`) corresponds to some function call. Hence `x-y` is just a syntactic sugar for `'-'(x,y)`. Moreover, other expressions such as `if..else` and loops also correspond to some function calls. For example:

```
R> show_ast(for(i in 1:5) {
+   print("i = ", i)
+   if (i %% 2 == 0) print(":") else print(":$")
+ })
```

```
List of 4
$ : symbol for
$ : symbol i
$ :List of 3
..$ : symbol :
```

```

..$ : num 1
..$ : num 5
$ :List of 3
..$ : symbol {
..$ :List of 3
...$ : symbol print
...$ : chr "i = "
...$ : symbol i
..$ :List of 4
...$ : symbol if
...$ :List of 3
... .$ : symbol ==
... . .$ :List of 3
... . . .$ : symbol %%
... . . . .$ : symbol i
... . . . . .$ : num 2
... . . . . .$ : num 0
... . .$ :List of 2
... . . .$ : symbol print
... . . . .$ : chr ":")
...$ :List of 2
... . . .$ : symbol print
... . . . .$ : chr ":$"

```

Vertex and edge types. The *vertices* of a PDG represent particular expressions, such as a variable assignment, a function call or a loop header. Each vertex is assigned its own *type*, reflecting the kind of expression it represents. The comprehensive list of vertex types for the R language code-base used in **SimilaR** is given in Table 1. The number of distinct types is a kind of compromise between the algorithm’s sensitivity and specificity. It was set empirically based on numerous experiments (Bartoszuk, 2018).

We may also distinguish two types of edges: *control dependency* and *data dependency* ones. The former represents the branches in a program’s control flow that result in a conditional execution of expressions such as **if-else**-constructs or loops. A subgraph of a PDG consisting of all the vertices

Id	Color	Type	Description
0	Olive	Entry	marks the beginning of a function
1	Light yellow	Header	loop
2	–	Next	<code>next</code>
3	–	Break	<code>break</code>
4	Orange	If	a conditional expression
5	Light blue	If_part	an expression to execute conditionally
6	Gray	assignment	an assignment expression, e.g., <code>name <- val</code>
7	Violet	parameter	a function parameter
8	–	oneBracketSingle	an expression involving <code>[i]</code> , e.g., <code>vector[-1]</code>
9	–	oneBracketDouble	an expression involving <code>[i,j]</code>
10	–	oneBracketTripleOrMore	an expression involving <code>[i,j,k,...]</code>
11	–	twoBrackets	an expression involving <code>[[i]]</code> , e.g., <code>list[["nameditem"]]</code>
12	–	dollar	an expression involving <code>\$</code> , e.g., <code>df\$column</code>
13	–	funNoArguments	a function call with no actual parameters
14	Red	funOneArgument	a function call with one actual parameter
15	–	funTwoArguments	a function call with two actual parameters
16	–	funThreeArguments	a function call with three actual parameters
17	–	funFourOrMoreArguments	a function call with four or more actual parameters
18	–	stopifnot	a call to <code>stopifnot()</code>
19	–	logicalOperator	a logical operator-based expression, e.g., <code>&</code> , <code> </code> or <code>!</code>
20	Green	arithmeticOperator	an arithmetic operator-based expression, e.g., <code>+</code> , <code>-</code> or <code>*</code>
21	Blue	comparisonOperator	a comparison operator-based expression, e.g., <code>==</code> , <code><=</code> or <code><</code>
22	Green	return	a call to <code>return()</code>
23	Cyan	colon	a colon operator-based expression, e.g., <code>from:to</code>
24	Dark green	symbol	a symbol (name)
25	Dark green	constant	a constant value, e.g., <code>1</code> , <code>"string"</code> or <code>NA</code>

Table 1: Assumed Program Dependence Graph vertex types for the R language.

and only the *control dependency* edges is called a *Control Dependence Subgraph* (CDS).

The latter edge vertex type is responsible for modelling data flow relations: there is an edge from a vertex v to a vertex u , whenever a variable assigned in the expression corresponding to v is used in the computation of the expression related to u . A spanning subgraph of a PDG that consists solely of the *data dependency* edges is called a *Data Dependence Subgraph* (DDS).

Hence, a PDG of a function $F()$ is a vertex- and edge-labelled directed graph $F = (V_F, E_F, \zeta_F, \xi_F)$, where V_F is the set of its vertices, $E_F \subseteq V_F \times V_F$ denotes the set of edges $((v, u) \in E_F \text{ whenever there is an edge from } v \text{ to } u)$, $\zeta_F : V_F \rightarrow \{\text{Entry, Header}, \dots, \text{constant}\} = \{0, \dots, 25\}$ gives the type of each vertex and $\xi_F : E_F \rightarrow \{\text{DATA, CONTROL}\}$ marks if an edge is a data- or control-dependency one. Note that each PDG is rooted – there exists one and only one vertex v with indegree 0 and $\zeta_F(v) = \text{Entry}$.

Example code chunks with the corresponding dependence graphs are depicted in Figures 1 and 2. The meaning of vertex colors is explained in Table 1.

```
sum <- function(x)
{
  s <- 0
  m <- 1
  for(i in x) {
    s <- s + i
    m <- m * i
  }

  if(s < 0) {
    s <- -s
    print("Negative s")
  }
  if(m < 0) {
    m <- -m
    print("Negative m")
  }
  return(s)
}
```

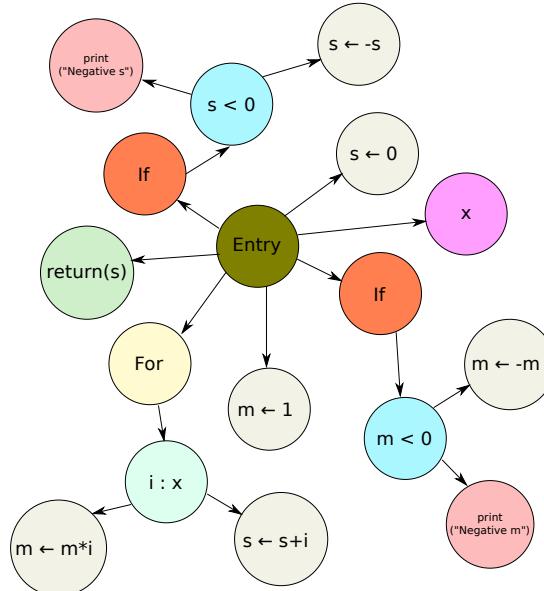


Figure 1: An example function and the respective Control Dependence Subgraph.

```
sum <- function(x)
{
  a <- 5
  b <- 6
  for(i in x)
  {
    c <- a + b - i
  }
}
```

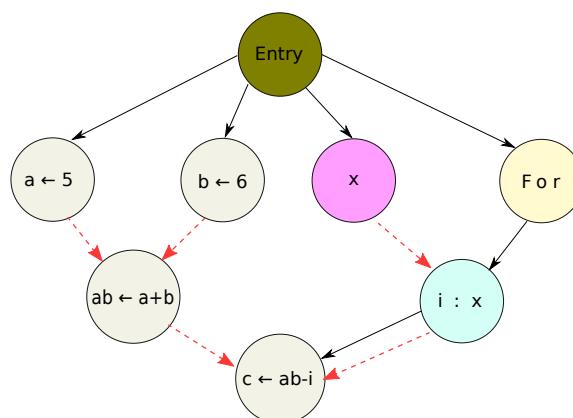


Figure 2: An example function and the corresponding Program Dependence Graph; solid and dashed arrows represent control and data dependency edges, respectively.

The most basic version of an algorithm to create a PDG based on an abstract syntax tree is described in (Harrold et al., 1993). Let us note the fact that a CDS is a subgraph of an AST: it provides the information about certain expressions being nested within other ones, e.g., that some assignment is part (child) of a loop’s body. Additionally, an AST includes a list of local variables and links them with expressions that rely on them. This is a crucial piece of information used to generate DDS.

Note that, however, a PDG created for the purpose of code clones detection cannot be treated as a straightforward extension of a raw AST. The post-processing procedure should be carefully customised taking into account the design patterns and coding practices of a particular programming language. Hence, below we describe the most noteworthy program transforms employed in the **SimilaR** package so that it is invariant to typical *attacks*, i.e., transforms changing the way the code is written yet not affecting its meaning.

Unwinding nested function calls. As mentioned above, in R, as in any functional language, functions play a key role. A code chunk can be thought of as a sequence of expressions, each of which is composed of function calls. Base or external library functions are used as a program’s building blocks and often very complex tasks can be written with only few lines of code.

For instance, given a matrix $\mathbf{X} \in \mathbb{R}^{d \times n}$ representing n vectors in \mathbb{R}^d and a vector $\mathbf{y} \in \mathbb{R}^d$, the closest vector in \mathbf{X} to \mathbf{y} with respect to the Euclidean metric can be determined by evaluating `X[,which.min(apply((X-y)^2, 2, sum))]`. This notation is very concise and we can come up with many equivalent forms of this expression written in a much more loquacious fashion.

Therefore, in **SimilaR**, hierarchies of nested calls, no matter their depth, are always recursively unwound by introducing as many auxiliary assignments as necessary. For instance, `f(g(x))` is decomposed as `gx <- g(x); f(gx)`. This guarantees that all their possible variants are represented in the same way in the PDG.

Forward-pipe operator, `%>%` Related to the above is the **magrittr**’s forward-pipe operator, `%>%`, which has recently gained much popularity within the R users’ community. Even though the operator is just a syntactic sugar for forwarding an object into the next function call/expression, at the time of writing of this manuscript, the package has been used as a direct dependency by over 700 other CRAN packages. Many consider it very convenient, as it mimics the “left-to-right” approach known from object-orientated languages like Java, Python or C++. Instead of writing (from the inside and out) `f(g(x), y)`, with **magrittr** we can use the syntax `x %>% g %>% f(y)` (which would normally be represented as `x.g().f(y)` in other languages). To assure proper similarity evaluation, **SimilaR** unwinds such expressions in the same manner as nested function calls.

Calls within conditions in control-flow expressions. An expression serving as a Boolean condition in an `if` or a `while` construct might be given as a composition of many function calls. The same might be true for an expression generating the container to iterate over in a `for` loop. PDG vertices representing such calls are placed on the same level as their corresponding control-flow expressions so that they can be unwound just as any other function call.

Canonicalization of conditional statements. The following code chunk:

```
if (cond) {
  return(A)
} else {
  return(B)
}
```

is semantically equivalent to:

```
if (cond)
  return(A)
return(B)
```

This exploits the fact that the code after the `if` statement is only executed whenever the logical condition is false. To avoid generating very different control dependencies, we always unwind the former to the latter by putting the code-richer branch outside of a conditional statement.

Tail call to `return()`. The return value that is generated by evaluating a sequence of expressions wrapped inside curly braces (the ``{`}` function) is determined by the value of its last expression. If a function’s body is comprised of such a code block, the call to `return()` is optional if it is used in the last expression. However, many users write it anyway. Therefore, a special vertex of type `return()` have been introduced to mark an expression that generates the output of a function.

Map-like functions. Base R supports numerous *Map*-like operations that are available in many programming languages. The aim of the members of the `*apply()` family (`apply()`, `lapply()`, `sapply()`, etc.) is to perform a given operation on each element/slice of a given container. These are unwound as an expression involving a `for` loop. For instance, a call to `ret <- lapply(l, fun, ...)` can be written as

```
ret <- list(); for (el in l) ret[[length(ret)+1]] <- fun(el, ...)
```

Variable duplication. To prevent redundant assignments such as `xcopy <- x` made just in order to refer to the original value under a new alias, a hierarchical variable dictionary is kept to generate *data dependency* edges properly.

Memoization. In pure functional languages it is assumed that functions have no side effects, i.e., the same arguments are mapped to the same return value. In R it is of course not always technically true (e.g., when the pseudo-random number generator is involved), but such an assumption turns out to be helpful in our context. Therefore, if a function call instance is invoked more than once, its value is memorised by introducing a new variable.

Dead code. Many plagiarism detection algorithms can be easily misled by adding random code that does not affect the main computations. In **SimilaR**, such *dead code* is identified and removed. This is done by iteratively deleting all vertices whose outdegree is zero (except those of type `return`).

To sum up, **SimilaR** guarantees that the Program Dependence Graph is the same regardless of the order of independent function calls, unwinding nested function calls, the use of the forward-pipe operator, etc. Hence, it is invariant to the most typical attacks. Moreover, it has been implemented in such a way that new kinds of transformations can be easily added in the future, as the R development practices and common program design patterns evolve.

Comparing Program Dependence Graphs

In our setting, code similarity assessment reduces to a comparison between a pair of Program Dependence Graphs. In this section we are interested in an algorithm μ such that $\mu(F, G) \in [0, 1]$ represents a similarity degree between two PDGs F and G . A similarity of 1 denotes that two PDGs are identical, while 0 means that they are totally different. Alternatively, we might be interested in a non-symmetric measure $\tilde{\mu}(F, G) \in [0, 1]$ representing the degree to which the source code of F is *contained* within G .

Ideally, an algorithm to compare two PDGs should enjoy the following properties:

- it should be *flexible* in the sense that introducing a “small difference” in one of the graphs should not affect the estimated similarity degree significantly;
- it should be *fast* to execute so that computing numerous pairwise similarities can be performed in a reasonable time span.

Due to the latter, we immediately lose our interest in all currently known *exact* algorithms to find subgraph isomorphisms or maximum common subgraphs because of their exponential-time complexity (the problems are NP-hard; see, e.g., [Wegener, 2005](#)). To recall, two graphs are isomorphic whenever there exists a mapping between the two graphs’ vertices preserving the node adjacencies.

In the **SimilaR** package, we use a modified version (for increased flexibility and better performance in the plagiarism detection problem) of an algorithm described in ([Shervashidze et al., 2011](#)), which itself is based on the Weisfeiler–Lehman isomorphism test ([Weisfeiler and Lehman, 1968](#)) and graphs kernels. Note that the base method has been successfully used in many applications, e.g., in cheminformatics ([Mapar, 2018](#)) and programming autonomous robots ([Luperto and Amigoni, 2019](#)).

In each of the h iterations of the SimilaR algorithm, we assign new labels to the vertices of a PDG based on their neighbours' labels. While in the original algorithm (Shervashidze et al., 2011), two vertices are considered diverse already when one of their neighbours has been assigned a different label, here we might still be assigning the same label if the vertices' adjacency differs only slightly. Our approach turns out to be more robust (Bartoszuk, 2018) against minor code changes or some vertices being missing in the graph.

SimilaR algorithm at a glance. Before we describe every step of the algorithm in detail, let us take a look at it from a bird's-eye perspective. If we are to assign each vertex a new label that is uniquely determined by their current type as well the labels allocated to their neighbours, two identical graphs will always be coloured the same way, no matter how many times we reiterate the labelling procedure. In particular, after h iterations, a vertex's label depends on the types of vertices whose distance from it is at most h .

We are of course interested in assigning equivalent labels to vertices in graphs that are not necessarily identical, but still *similar* to each other. Otherwise, two vertices which have all but one neighbour in common, would get distinct labels. After h iterations, all the vertices at distance at most h would all already be assigned different colours. This would negatively affect the overall graph similarity assessment.

In order to overcome this problem, we introduce the concept of vertex *importance*, which is based upon the number of vertices that depend on a given node. Only *important enough* differences in the vertex neighbourhoods will be considered as sufficient to trigger a different labelling. Then, after h iterations, two vectors of label type counts can be compared with each other to arrive at a final graph similarity degree.

SimilaR algorithm in detail. The following description of the SimilaR algorithm will be illustrated based on a comparison between two functions, `clamp1()` (whose PDG from now on we will denote with $F = (V_F, E_F, \zeta_F, \xi_F)$, see Fig. 3) and `standardise()` (denoted $G = (V_G, E_G, \zeta_G, \xi_G)$, see Fig. 4).

v	$Entry$	x	$min(x)$	$max(x)$	- (1)	$<$	If	If_part	$NULL$	- (2)	/
raw $\delta_F(v)$	7.80	3.04	1.35	1.12	0.93	0.65	0.30	0.20	0.10	0.21	0.10
normalised	0.49	0.19	0.09	0.07	0.06	0.04	0.02	0.01	0.01	0.01	0.01

Table 2: Vertex importance degrees in `clamp1()` (sum = 15.79, median = 0.04).

v	$Entry$	x	$sd(x)$	$<$	If	If_part	$NULL$	$mean(x)$	-	/
raw $\delta_G(v)$	4.33	1.71	0.93	0.65	0.30	0.20	0.10	0.33	0.21	0.1
normalised	0.49	0.19	0.10	0.07	0.03	0.02	0.01	0.04	0.02	0.01

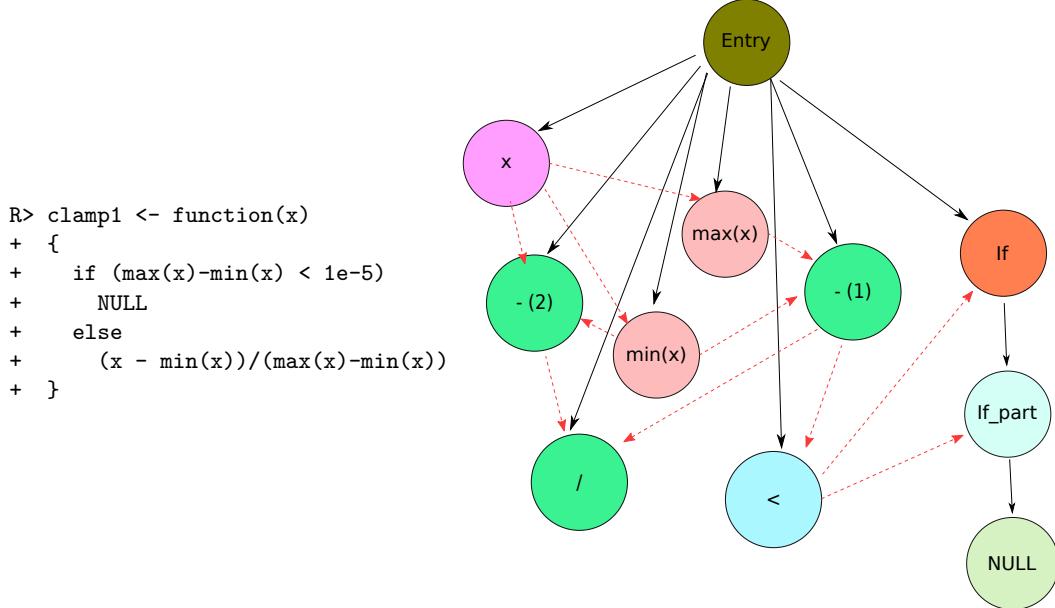
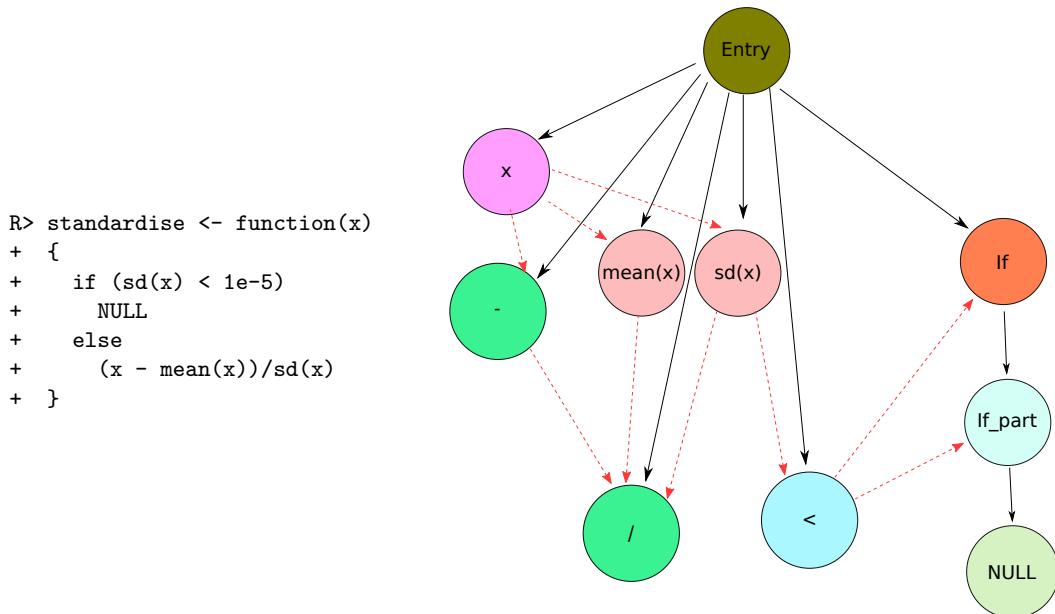
Table 3: Vertex importance degrees in `standardise()` (sum = 8.86, median = 0.03).

v	$Entry$	x	$min(x)$	$max(x)$	- (1)	$<$	If	If_part	$NULL$	- (2)	/
$\zeta_F(v)$	0	24	14	14	20	21	4	5	25	20	20
$\zeta_F^1(v)$	0	1	2	2	3	4	5	6	7	3	3
$\zeta_F^2(v)$	7	1	2	2	8	3	4	5	0	6	6
$\zeta_F^3(v)$	8	5	6	6	7	1	2	3	0	4	4

Table 4: `clamp1()`: Labels assigned to vertices in each iteration.

<i>v</i>	<i>Entry</i>	<i>x</i>	<i>sd(x)</i>	<	<i>If</i>	<i>If_part</i>	<i>NULL</i>	<i>mean(x)-</i>	/
$\zeta_G(v)$	0	24	14	21	4	5	25	14	20
$\zeta_G^1(v)$	0	1	2	4	5	6	7	2	3
$\zeta_G^2(v)$	7	1	9	3	4	5	0	2	6
$\zeta_G^3(v)$	12	9	10	1	2	3	0	11	4

Table 5: `standardise()`: Labels assigned to vertices in each iteration.

Figure 3: The `clamp1()` function with the respective PDG.Figure 4: The `standardise()` function with the respective PDG.

1. **Vertex importance degrees.** Firstly, each vertex in F is assigned an importance degree, $\delta_F : V_F \rightarrow \mathbb{R}_+$,

$$\delta_F(v) = 0.1 + \sum_{(v,u) \in E_F} (1 + 0.1\mathbb{I}_{\xi_F((v,u))=\text{DATA}}) \delta_F(u),$$

where \mathbb{I}_{cond} is an indicator function with value 1 if cond is true and 0 otherwise. In other words, a vertex v with outdegree equal to 0 has importance $\delta(v) = 0.1$. Otherwise, its importance degree is set to $\delta(v) = 0.1$ plus the sum of importances of its outgoing control-dependent neighbours plus the sum of importances of its outgoing data-dependent neighbours multiplied by 1.1.

Note that if F is an acyclic graph, then it has a topological ordering, i.e., an arrangement of the vertices such that every edge is directed from earlier to later in the sequence. In such a case δ_F is well-defined. Otherwise, we shall be computing the importance degrees in the depth-first manner.

Next, the importance degrees are normalised, $\delta_F(v) \mapsto \delta_F(v)/\sum_{u \in V_F} \delta_F(u)$. Tables 2 and 3 give the importance degrees of the vertices in the two graphs studied.

2. **Vertex labels.** Recall from the previous section that each vertex $v \in V_F, u \in V_G$ has been assigned a label, $\zeta_F(v), \zeta_G(u) \in \{0, \dots, 25\}$, based on the type of operation it represents (see Tables 4 and 5).

In the i -th (out of h in total, here we fix $h = 3$; see Bartoszuk, 2018 for discussion) iteration of the SimilaR algorithm, we assign new labels ζ_F^i, ζ_G^i according to the labels previously considered.

- (a) **Iteration $i = 1$.** In the first iteration, the initial labels, ζ_F, ζ_G , are simply remapped to consecutive integers. This yields ζ_F^1 and ζ_G^1 as given in Tables 4 and 5.
- (b) **Iterations $i = 2$ and $i = 3$.** In subsequent iterations, we seek groups of similar vertices so as to assign them the same label. Two vertices $v, u \in V_F \cup V_G$ are considered *similar* (with no loss in generality, we are assuming $v \in V_F$ and $u \in V_G$ below), whenever they have been assigned the same label in the previous iteration *and* have outgoing neighbours with the same labels. However, for greater flexibility, we allow for the neighbourhoods to differ slightly – unmatched neighbours of lesser importance degrees will not be considered significant. Formally, $v \in V_F$ and $u \in V_G$ are similar, whenever $\zeta_F^{i-1}(v) = \zeta_G^{i-1}(u)$ and:

$$\sum_{(v,w) \in E_F, \zeta_F^{i-1}(w) \notin C(v,u)} \delta_F(w) \leq \min\{M_F, M_G\},$$

where $C(v, u) = \{\zeta_F^{i-1}(w) : (v, w) \in E_F\} \cap \{\zeta_G^{i-1}(w) : (u, w) \in E_G\}$ denotes the *multiset* of common neighbours' vertex labels and M_F and M_G denote the medians of importance degrees of vertices in F and G , respectively.

The above similarity relation is obviously reflexive and symmetric. When we compute its transitive closure, we get an equivalence relation whose equivalence classes determine sets of vertices that shall obtain identical labels.

For instance, let $i = 2$ and v be the *Entry* vertex in F and u be the *Entry* vertex in u . We have $\zeta_F^1(v) = \zeta_G^1(u) = 0$. Outgoing neighbours of v have types: 1 (x), 3 (- (2)), 3 ($/$), 2 ($\min(x)$), 2 ($\max(x)$), 4 ($<$), 3 (- (1)) and 5 (If). Neighbours of u are labelled as 1 (x), 3 (-), 2 ($\text{mean}(x)$), 3 ($/$), 2 ($sd(x)$), 4 ($<$), 5 (If). Hence, v has one unmatched neighbour of type 3 (here: an arithmetic operation). However, as there are 3 such neighbours of v (importances: 0.01, 0.01, 0.06) and 2 of u (importances: 0.01, 0.02), we need a rule for determining the left-out importance degree. Here, for greater algorithm's flexibility, we always assume that the unmatched importances are considered in an increasing order. Therefore, we get that $0.01 \leq \min\{M_F, M_G\} = \min\{0.04, 0.03\} = 0.03$ and thus v and u are considered similar.

As another example, let $i = 3$ and v be the x vertex in F and u be the x vertex in G . We have $\zeta_F^2(v) = \zeta_G^2(u) = 1$. Their neighbours are: - (2) (importance=0.01; type=6), $\min(x)$ (importance=0.09; type=2), $\max(x)$ (importance=0.07; type=2) and - (importance=0.02; type=6), $\text{mean}(x)$ (importance=0.04; type=2), $sd(x)$ (importance=0.10; type=9). The sum of the importance degrees of the unmatched neighbours is now equal to $0.07 + 0.10 > 0.03$, hence, u and v are not considered similar. They are also not in the transitive closure with respect to the other similarities. Therefore, they will be assigned different labels.

3. **Partial similarity degrees.** Let m be the maximal integer label assigned above and $\mathbf{L}_F^i = (\mathbf{L}_{F,1}^i, \dots, \mathbf{L}_{F,m}^i)$ be a vector of label counts, where $\mathbf{L}_{F,j}^i = |\{v \in V_F : \zeta_F^i(v) = j\}|$, see Table 6. We define \mathbf{L}_G^i in much the same way, see Table 7. We introduce the following “partial” similarity measures of the label sequences – the symmetric:

$$\mu(\mathbf{L}_F^i, \mathbf{L}_G^i) = 1 - \frac{\sum_{k=1}^m |\mathbf{L}_{F,k}^i - \mathbf{L}_{G,k}^i|}{\sum_{k=1}^m \mathbf{L}_{F,k}^i + \sum_{k=1}^m \mathbf{L}_{G,k}^i}$$

and its nonsymmetric version:

$$\tilde{\mu}(\mathbf{L}_F^i, \mathbf{L}_G^i) = \frac{\sum_{k=1}^m \min(\mathbf{L}_{F,k}^i, \mathbf{L}_{G,k}^i)}{\sum_{k=1}^m \mathbf{L}_{F,k}^i}$$

The partial similarities for $i = 1, 2, 3$ are given in Table 8.

j	0	1	2	3	4	5	6	7	8	9	10	11	12
$\mathbf{L}_{F,j}^1$	1	1	2	3	1	1	1	1	0	0	0	0	0
$\mathbf{L}_{F,j}^2$	1	1	2	1	1	1	2	1	1	0	0	0	0
$\mathbf{L}_{F,j}^3$	1	1	1	1	2	1	2	1	1	0	0	0	0

Table 6: `clamp1()`: Label counts in each iteration.

j	0	1	2	3	4	5	6	7	8	9	10	11	12
$\mathbf{L}_{G,j}^1$	1	1	2	2	1	1	1	1	0	0	0	0	0
$\mathbf{L}_{G,j}^2$	1	1	1	1	1	1	2	1	0	1	0	0	0
$\mathbf{L}_{G,j}^3$	1	1	1	1	2	0	0	0	0	1	1	1	1

Table 7: `standardise()`: Label counts in each iteration.

i	$\mu(\mathbf{L}_F^i, \mathbf{L}_G^i)$	$\tilde{\mu}(\mathbf{L}_F^i, \mathbf{L}_G^i)$	$\tilde{\mu}(\mathbf{L}_G^i, \mathbf{L}_F^i)$
1	0.95	0.91	1.00
2	0.86	0.82	0.90
3	0.57	0.55	0.60
Final	0.79	0.76	0.83

Table 8: Similarity measures in each iteration.

4. **Final similarity degrees.** The overall similarity degree is defined as the arithmetic mean of the $h = 3$ partial similarities (reported in Table 8):

$$\mu(F, G) = \frac{1}{h} \sum_{i=1}^h \mu(\mathbf{L}_F^i, \mathbf{L}_G^i) = \frac{0.95 + 0.86 + 0.57}{3} = 0.79, \quad (1)$$

We obtain its nonsymmetric versions in the same way:

$$\begin{aligned} \tilde{\mu}(F, G) &= \frac{1}{h} \sum_{i=1}^h \tilde{\mu}(\mathbf{L}_F^i, \mathbf{L}_G^i) = \frac{0.91 + 0.82 + 0.55}{3} = 0.76, \\ \tilde{\mu}(G, F) &= \frac{1}{h} \sum_{i=1}^h \tilde{\mu}(\mathbf{L}_G^i, \mathbf{L}_F^i) = \frac{1.00 + 0.90 + 0.60}{3} = 0.83. \end{aligned} \quad (2)$$

Having discussed the algorithms behind the **SimilaR** package, let us proceed with the description of its user interface.

Illustrative examples

The **SimilaR** package can be downloaded from CRAN and installed on the local system via a call to:

```
R> install.packages("SimilaR")
```

Here we are working with version 1.0.8 of the package.

Once the package is loaded and its namespace is attached by calling:

```
R> library("SimilaR")
```

two functions are made available to a user. `SimilaR_fromTwoFunctions()` is responsible for assessing the similarity between a pair of function objects (R is a functional language, hence assuming that functions constitute basic units of code seem natural). Moreover, `SimilaR_fromDirectory()`, which we shall use in the next section, is a conveniently vectorised version of the former, performing the comparison of all the scripts in a given directory.

Let us evaluate the similarity between the $F = \text{clamp1}()$ and $G = \text{standardise}()$ functions defined above:

```
R> SimilaR_fromTwoFunctions(clamp1, standardise) # aggregation="sym"
   name1      name2  SimilaR decision
1 clamp1 standardise 0.7954545      1
```

Here, **SimilaR** denotes the (symmetric) measure $\mu(F, G) \in [0, 1]$ as given by Eq. (1). **decision** uses a built-in classifier model to assess whether such a similarity degree is considered significant (1) or not (0).

To obtain the non-symmetric measures, $\tilde{\mu}(F, G)$ and $\tilde{\mu}(G, F)$ (see Eq. (2)), we pass **aggregation="both"** as an argument:

```
R> SimilaR_fromTwoFunctions(clamp1, standardise, aggregation="both")
   name1      name2 SimilaR12 SimilaR21 decision
1 clamp1 standardise 0.7575758 0.8333333      1
```

Example: clamp2(). Let us recall the source code of the **clamp1()** function:

```
R> clamp1
function(x)
{
  if (max(x)-min(x) < 1e-5)
    NULL
  else
    (x - min(x))/(max(x)-min(x))
}
```

By applying numerous transformations described above, we may arrive at its following version:

```
R> library("magrittr")
R> clamp2 <- function(y)
+ {
+   longName <- y                      # variable duplication
+   longName2 <- min
+   z <- { sum(longName**2) }            # dead code
+   min_y <- longName %>% longName2 # forward-pipe
+   max_y <- y %>% max
+   max_y_min_y <- max_y-min_y # memoization
+   if(!(max_y_min_y >= 1e-5)) # canonicalization of the if statement
+   {
+     return(NULL)
+   }
+   ((y - min_y)/max_y_min_y) # tail call to return removed
+ }
```

SimilaR correctly identifies the two functions as equivalent:

```
R> SimilaR_fromTwoFunctions(clamp1, clamp2, aggregation="both")
   name1  name2 SimilaR12 SimilaR21 decision
1 clamp1 clamp2      1      1      1
```

Example: A vectorised version of clamp1(). Let us now consider two different vectorised versions of the **clamp1()** function for list-type inputs. The first one is based on a call to **lapply()**, which takes care of applying a given anonymous function on each list member:

```
R> clamp1_vectorised1 <- function(x) {
+   x %>% lapply(function(y) {
+     if (max(y)-min(y) < 1e-5) {
+       {{{NULL}}}
+     } else {
```

```

+
+           {{{{(y - min(y))/(max(y)-min(y))}}}}}
+
+       }
+
+   }

```

The second function unwinds the Map-like construct, adding a `for`-loop instead. It also relies on some domain knowledge, namely, that a new list is pre-allocated with `NULLs`.

```

R> clamp1_vectorised2 <- function(x) {
+   n <- length(x)
+   res <- vector("list", n) # NULLs
+   for (i in 1:n) { # assumed n>0
+     m <- min(x[[i]])
+     mm <- max(x[[i]])-m
+     if (mm >= 1e-5)
+       res[[i]] <- (x[[i]] - m)/mm
+   }
+   return(res)
+ }

```

Note that the function only works for non-empty input lists.

The pairwise comparison yields:

```

R> SimilaR_fromTwoFunctions(clamp1_vectorised1, clamp1_vectorised2,
+                             aggregation="both")

```

name1	name2	SimilaR12	SimilaR21	decision
clamp1_vectorised1	clamp1_vectorised2	0.7833333	0.9215686	1

Which indicates a significant degree of similarity, which is indeed the case.

A case study

In the previous section we illustrated that **SimilaR** is easily able to identify the code chunks that can be transformed *onto* each other. Now we shall demonstrate its usefulness in a real-world scenario: let us compare the code-base of two R packages: `nortest` (Gross and Ligges, 2015) and `DescTools` (Signorell et al., 2020). The former implements five significance tests for normality, while the latter advertises itself as

*A collection of miscellaneous basic statistic functions and convenience wrappers for efficiently describing data. [...] Many of the included functions can be found scattered in other packages and other sources written partly by Titans of R. The reason for collecting them here, was primarily to have them consolidated in ONE instead of dozens of packages (which themselves might depend on other packages which are not needed at all), and to provide a common and consistent interface as far as function and arguments naming, NA handling, recycling rules, etc. are concerned. [...] (**DescTools** package DESCRIPTION; Signorell et al., 2020)*

1. Set-up. First we attach the required packages and set up the directory where we shall store the data that we are going to feed the algorithm with at a later stage.

```

R> library("SimilaR")
R> dir_output <- tempfile("dir")
R> dir.create(dir_output)

```

2. Generate code-base. **SimilaR** does not offer direct support for comparing two R packages. Therefore, below we export each package's code-base to a single source file, getting rid of non-function objects:

```

R> for (pkg in c("DescTools", "nortest"))
+ {
+   library(pkg, character.only=TRUE) # attach the package
+   env <- as.environment(paste0("package:", pkg)) # package's environment

```

```

+   fun_names <- ls(envir=env) # list of all exported objects
+   file_text <- character(0) # the to-be code-base (1 function == 1 string)
+   for (fun in fun_names)
+   {
+     f <- get(fun, env) # get function object
+     if (!is.function(f)) next
+
+     f_char <- paste0(deparse(f), collapse="\n") # extract source code
+     file_text[length(file_text)+1] <- sprintf("`%s`<-%s", fun, f_char)
+   }
+   file_name <- file.path(dir_output, paste0(pkg, ".R"))
+   writeLines(file_text, file_name) # write source file
+   cat(sprintf("%s: %d functions processed.\n", pkg, length(file_text)))
+ }

DescTools: 549 functions processed.
nortest: 5 functions processed.

```

Here the list of the objects exported by both packages is determined by querying their corresponding `package:DescTools` and `package:nortest` environments. Moreover, a call to `deparse()` on a function object gives a plain-text representation of its source code.

3. Run the algorithm. Now we ask the algorithm to fetch the two source files in the output directory and execute all the pairwise comparisons between the functions defined therein.

```

R> time0 <- Sys.time()
R> results <- SimilaR_fromDirectory(dir_output,
+      fileTypes="file", aggregation="both")
R> print(Sys.time()-time0)

Time difference of 15.41459 secs

```

The above gives the time to execute all the 2745 pairwise comparisons on an Intel Core i7 laptop with 16GB RAM, running the GNU/Linux 4.19.30-041930-generic SMP x86_64 kernel.

4. Report results. Let us inspect the top 10 results returned by the algorithm (in terms of overall similarity).

```
R> print(head(results, 10))
```

For greater readability, the results are reported in Table 9.

Name1 (DescTools)	Name2 (nortest)	SimilaR12	SimilaR21	Decision
CramerVonMisesTest()	cvm.test()	1.0000000	1.0000000	1
LillieTest()	lillie.test()	1.0000000	1.0000000	1
PearsonTest()	pearson.test()	1.0000000	1.0000000	1
ShapiroFranciaTest()	sf.test()	1.0000000	1.0000000	1
CramerVonMisesTest()	ad.test()	0.9451477	0.9105691	1
CramerVonMisesTest()	lillie.test()	0.7299578	0.5339506	0
LillieTest()	cvm.test()	0.5339506	0.7299578	0
LillieTest()	ad.test()	0.5339506	0.7032520	0
Eps()	sf.test()	0.5333333	0.6315789	0
ShapiroFranciaTest()	ad.test()	0.7719298	0.3577236	0

Table 9: Similarity report (the top 10 results) for the comparison between the code-base of the `DescTools` and `nortest` packages.

Discussion. We observe that 5 function pairs were marked as similar (`Decision = 1`). The top 4 results accurately indicate the corresponding normality tests from the two packages – their sources are identical.

However, the 5th result is a *false positive*: `CramerVonMisesTest()` is reported as similar to `ad.test()`, which implements the Anderson–Darling normality test. Let us “visually” compare their sources:

```
CramerVonMisesTest <- function(x) {
  DNAME <- deparse(substitute(x))
  x <- sort(x[complete.cases(x)])
  n <- length(x)
  if (n < 8)
    stop("sample size must be greater than 7")
  p <- pnorm((x - mean(x))/sd(x))
  W<- (1/(12 * n) + sum((p - (2 * seq(1:n) - 1)/(2 * n))^2))
  WW <- (1 + 0.5/n) * W
  if (WW < 0.0275) {
    pval <- 1 - exp(-13.953 + 775.5 * WW - 12542.61 * WW^2)
  }
  else if (WW < 0.051) {
    pval <- 1 - exp(-5.903 + 179.546 * WW - 1515.29 * WW^2)
  }
  else if (WW < 0.092) {
    pval <- exp(0.886 - 31.62 * WW + 10.897 * WW^2)
  }
  else if (WW < 1.1) {
    pval <- exp(1.111 - 34.242 * WW + 12.832 * WW^2)
  }
  else {
    warning("p-value is smaller than 7.37e-10,
            cannot be computed more accurately")
    pval <- 7.37e-10
  }
  RVAL <- list(statistic = c(W = W), p.value = pval,
               method = "Cramer-von Mises normality test",
               data.name = DNAME)
  class(RVAL) <- "htest"
  return(RVAL)
}

ad.test <- function(x) {
  DNAME <- deparse(substitute(x))
  x <- sort(x[complete.cases(x)])
  n <- length(x)
  if (n < 8)
    stop("sample size must be greater than 7")
  logp1 <- pnorm((x - mean(x))/sd(x), log.p = TRUE)
  logp2 <- pnorm(-(x - mean(x))/sd(x), log.p = TRUE)
  h <- (2 * seq(1:n) - 1) * (logp1 + rev(logp2))
  A<- -n - mean(h)
  AA <- (1 + 0.75/n + 2.25/n^2) * A
  if (AA < 0.2) {
    pval <- 1 - exp(-13.436 + 101.14 * AA - 223.73 * AA^2)
  }
  else if (AA < 0.34) {
    pval <- 1 - exp(-8.318 + 42.796 * AA - 59.938 * AA^2)
  }
  else if (AA < 0.6) {
    pval <- exp(0.9177 - 4.279 * AA - 1.38 * AA^2)
  }
  else if (AA < 10) {
    pval <- exp(1.2937 - 5.709 * AA + 0.0186 * AA^2)
  }
  else pval <- 3.7e-24
  RVAL <- list(statistic = c(A = A), p.value = pval,
               method = "Anderson-Darling normality test",
               data.name = DNAME)
  class(RVAL) <- "htest"
```

```

    return(RVAL)
}

```

Basically, the main difference between the two functions is in the numeric constants used, which we know is a kind of detail purposefully ignored by **SimilaR**. Indeed, knowing that the Anderson–Darling test is a generalisation of the Cramér–von Mises test (both of them are based on the L_2 -distance between the empirical and true cumulative distribution function, the former is a weighted version of the latter), makes the reported decision concerning the *similarity* judgement quite justified.

Interestingly, **DescTools** does provide the `AndersonDarlingTest` function, but this is a version of the goodness-of-fit measure to test against *any* probability distribution provided. In other words, a c.d.f. of a normal distribution is not hard-coded in its source, and thus is significantly different from the code of `ad.test()`.

It is also worth noting that there are no false positives in terms of statistical tool types – all the functions deal with some form of goodness-of-fit testing and we recall that **DescTools** defines ca. 550 functions in total.

Discussion

We have introduced an algorithm to quantify the similarity between a pair of R source code chunks. The method is based on carefully prepared *Program Dependence Graphs*, which assure that semantically equivalent code pieces are represented in the same manner even if they are written in much different ways. This makes the algorithm robust with respect to the most typical *attacks*. In a few illustrative examples, we have demonstrated typical code alterations that the algorithm is invariant to, for instance, aliasing of variables, changing the order of independent code lines, unwinding nested function calls, etc.

In the presented case study we have analysed the similarities between the **DescTools** and **nortest** packages. Recall that most of the cloned function pairs are correctly identified, proving the practical usefulness of the **SimilaR** package. The reported above-threshold similarity between `CramerVonMisesTest()` and `ad.test()` is – strictly speaking – a false positive, nevertheless our tool correctly indicates that the two functions have been implemented in much the same way. This might serve as a hint to package developers that the two tests could be refactored so as to rely on a single internal function – de-duplication is among the most popular ways to increase the broadly conceived quality of software code.

On the other hand, the algorithm failed to match the very much-different (implementation-wise) `AndersonDarlingTest()` (generic distribution) with its specific version of `ad.test()` (normal distribution family). However, comparisons of such a kind, in order to be successful, would perhaps require the use of an extensive knowledge-base and are of course beyond the scope of our tool.

Finally, let us note that due to the use of a new polynomial-time algorithm, assessing the similarity of two Program Dependence Graphs is relatively fast. This makes **SimilaR** appropriate for mining software repositories even of quite considerable sizes. However, some pre-filtering of function pairs (e.g., based on cluster analysis) to avoid performing all the pairwise comparisons would make the system even more efficient and scalable.

Future versions of the **SimilaR** package will be equipped with standalone routines aiming at improving the quality and efficiency of R code, such as detecting dead or repeated code, measuring cyclomatic complexity, checking if the program is well structured, etc.

Acknowledgements. The authors would like to thank the Reviewers for valuable feedback that helped improve this manuscript.

Bibliography

- H. Abelson, G. J. Sussman, and J. Sussman. *Structure and Interpretation of Computer Programs*. MIT Press, 1996. [p]
- A. T. Ali, H. M. Abdulla, and V. Snasel. Overview and comparison of plagiarism detection tools. In *Proceedings of the Dateso 2011: Annual International Workshop on Databases, Texts, Specifications and Objects*, Dateso '11, pages 161–172, 2011. [p]
- P. Ammann and J. Offutt. *Introduction to Software Testing*. Cambridge University Press, 2013. [p]

- 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. [p]
- M. Bartoszuk. *A Source Code Similarity Assessment System for Functional Programming Languages Based on Machine Learning and Data Aggregation Methods*. PhD thesis, Warsaw University of Technology, Warsaw, Poland, 2018. [p]
- I. D. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier. Clone detection using abstract syntax trees. In *Proceedings of the International Conference on Software Maintenance*, ICSM '98, pages 368–378, 1998. [p]
- R. Becker, J. Chambers, and A. Wilks. *The New S Language*. Chapman & Hall, 1998. [p]
- J. Chambers. *Programming with Data*. Springer, 1998. [p]
- J. Chambers. *Software for Data Analysis. Programming with R*. Springer-Verlag, 2008. [p]
- J. Ferrante, K. J. Ottenstein, and J. D. Warren. The program dependence graph and its use in optimization. *ACM Transactions on Programming Languages and Systems*, 9(3):319–349, 1987. URL <https://doi.org/10.1145/24039.24041>. [p]
- D. Fu, Y. Xu, H. Yu, and B. Yang. WASTK: A weighted abstract syntax tree kernel method for source code plagiarism detection. *Scientific Programming*, 2017:7809047, 2017. [p]
- M. Gabel, L. Jiang, and Z. Su. Scalable detection of semantic clones. In *Proceedings of the 30th International Conference on Software Engineering*, ICSE '08, pages 321–330, 2008. URL <https://doi.org/10.1145/1368088.1368132>. [p]
- A. Ghosh and Y. Lee. An Empirical Study of a Hybrid Code Clone Detection Approach on Java Byte Code. *GSTF Journal on Computing*, 5(2):34–45, 2018. [p]
- S. Grant, J. R. Cordy, and D. B. Skillicorn. Using topic models to support software maintenance. In *2012 16th European Conference on Software Maintenance and Reengineering*, pages 403–408, 2012. URL <https://doi.org/10.1109/CSMR.2012.51>. [p]
- J. Gross and U. Ligges. *nortest: Tests for Normality*, 2015. URL <https://cran.r-project.org/package=nortest>. R package version 1.0-4. [p]
- J. Hage, P. Rademaker, and N. van Vugt. Plagiarism Detection for Java: A Tool Comparison. In *Computer Science Education Research Conference*, CSERC '11, pages 33–46, 2011. [p]
- M. J. Harrold, B. Malloy, and G. Rothermel. Efficient construction of Program Dependence Graphs. Technical report, ACM International Symposium on Software Testing and Analysis, 1993. [p]
- S. Horwitz and T. Reps. Efficient comparison of program slices. *Acta Informatica*, 28(8):713–732, 1991. URL <https://doi.org/10.1007/BF01261653>. [p]
- B. Hummel, E. Juergens, and D. Steidl. Index-based model clone detection. In *Proceedings of the 5th International Workshop on Software Clones*, IWSC '11, pages 21–27, 2011. URL <https://doi.org/10.1145/1985404.1985409>. [p]
- L. Jiang, G. Mishergi, Z. Su, and S. Glondu. Deckard: Scalable and accurate tree-based detection of code clones. In *Proceedings of the 29th International Conference on Software Engineering*, ICSE '07, pages 96–105, 2007. URL <https://doi.org/10.1109/ICSE.2007.30>. [p]
- J. H. Johnson. Identifying redundancy in source code using fingerprints. In *Proceedings of the 1993 Conference of the Centre for Advanced Studies on Collaborative Research: Software Engineering – Volume 1*, CASCON '93, pages 171–183, 1993. [p]
- T. Kamiya, S. Kusumoto, and K. Inoue. CCFinder: A multilingual token-based code clone detection system for large scale source code. *IEEE Transactions on Software Engineering*, 28(7):654–670, 2002. URL <https://doi.org/10.1109/TSE.2002.1019480>. [p]
- R. Komondoor and S. Horwitz. Using slicing to identify duplication in source code. In *Proceedings of the 8th International Symposium on Static Analysis*, SAS '01, pages 40–56, 2001. [p]
- J. Krinke. Identifying similar code with program dependence graphs. In *Proceedings of the Eighth Working Conference on Reverse Engineering (WCRE'01)*, WCRE '01, pages 301–307, 2001. [p]

- Z. Li, S. Lu, S. Myagmar, and Y. Zhou. CP-Miner: finding copy-paste and related bugs in large-scale software code. *IEEE Transactions on Software Engineering*, 32(3):176–192, 2006. URL <https://doi.org/10.1109/TSE.2006.28>. [p]
- E. Linstead, P. Rigor, S. Bajracharya, C. Lopes, and P. Baldi. Mining concepts from code with probabilistic topic models. In *Proceedings of the Twenty-second IEEE/ACM International Conference on Automated Software Engineering*, ASE ’07, pages 461–464, 2007. ISBN 978-1-59593-882-4. URL <https://doi.org/10.1145/1321631.1321709>. [p]
- C. Liu, C. Chen, J. Han, and P. S. Yu. GPLAG: Detection of software plagiarism by program dependence graph analysis. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’06, pages 872–881, 2006. URL <https://doi.org/10.1145/1150402.1150522>. [p]
- S. K. Lukins, N. A. Kraft, and L. H. Etzkorn. Source code retrieval for bug localization using latent Dirichlet allocation. In *Proceedings of the 2008 15th Working Conference on Reverse Engineering*, WCRE ’08, pages 155–164, 2008. ISBN 978-0-7695-3429-9. URL <https://doi.org/10.1109/WCRE.2008.33>. [p]
- M. Luperto and F. Amigoni. Predicting the global structure of indoor environments: A constructive machine learning approach. *Autonomous Robots*, 43(4):813–835, 2019. [p]
- U. Manber. Finding similar files in a large file system. In *USENIX Winter 1994 Technical Conference*, pages 1–10, 1994. [p]
- P. Mapar. Machine learning for enzyme promiscuity. Master’s thesis, Aalto University, Finland, 2018. [p]
- A. Marcus and J. I. Maletic. Identification of high-level concept clones in source code. In *Proceedings of the 16th IEEE International Conference on Automated Software Engineering*, ASE ’01, pages 107–114, 2001. [p]
- V. T. Martins, D. Fonte, P. R. Henriques, and D. da Cruz. Plagiarism Detection: A Tool Survey and Comparison. In M. J. V. Pereira et al., editors, *3rd Symposium on Languages, Applications and Technologies*, OpenAccess Series in Informatics (OASIcs), pages 143–158, Dagstuhl, Germany, 2014. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi: 10.4230/OASIcs.SLATE.2014.143. [p]
- G. Maskeri, S. Sarkar, and K. Heafield. Mining business topics in source code using latent Dirichlet allocation. In *Proceedings of the 1st India Software Engineering Conference*, ISEC ’08, pages 113–120, 2008. ISBN 978-1-59593-917-3. URL <https://doi.org/10.1145/1342211.1342234>. [p]
- J. Mayrand, C. Leblanc, and E. M. Merlo. Experiment on the automatic detection of function clones in a software system using metrics. In *1996 Proceedings of International Conference on Software Maintenance*, pages 244–253, 1996. URL <https://doi.org/10.1109/ICSM.1996.565012>. [p]
- P. W. McBurney, C. Liu, C. McMillan, and T. Weninger. Improving topic model source code summarization. In *Proc. 22nd International Conference on Program Comprehension*, ICPC 2014, pages 291–294, 2014. ISBN 978-1-4503-2879-1. URL <https://doi.org/10.1145/2597008.2597793>. [p]
- M. Misic, Z. Siustran, and J. Protic. A comparison of software tools for plagiarism detection in programming assignments. *International Journal of Engineering Education*, 32(2):738–748, 2016. [p]
- A. Mohd Noor et al. Programming similarity checking system. *Journal of Telecommunication, Electronic and Computer Engineering*, 9(3–5):89–94, 2017. [p]
- H. Nasirloo and F. Azimzadeh. Semantic code clone detection using abstract memory states and program dependency graphs. In *Proc. 4th International Conference on Web Research (ICWR)*, pages 19–27, 2018. [p]
- J. F. Patenaude, E. Merlo, M. Dagenais, and B. Lague. Extending software quality assessment techniques to Java systems. In *Proceedings Seventh International Workshop on Program Comprehension*, pages 49–56, 1999. URL <https://doi.org/10.1109/WPC.1999.777743>. [p]
- L. Prechelt, G. Malpohl, and M. Philippse. JPlag: Finding plagiarisms among a set of programs. Technical report, University of Karlsruhe, Department of Informatics, 2000. [p]
- W. Qu, Y. Jia, and M. Jiang. Pattern mining of cloned codes in software systems. *Information Sciences*, 259:544–554, 2014. URL <https://doi.org/10.1016/j.ins.2010.04.022>. [p]

- D. Rattan, R. Bhatia, and M. Singh. Software clone detection: A systematic review. *Information and Software Technology*, 55(7):1165–1199, 2013. URL <https://doi.org/10.1016/j.infsof.2013.01.008>. [p]
- M. Rieger. *Effective clone detection without language barriers*. PhD thesis, University of Bern, Switzerland, 2005. [p]
- C. K. Roy, J. R. Cordy, and R. Koschke. Comparison and evaluation of code clone detection techniques and tools: A qualitative approach. *Science of Computer Programming*, 74(7):470–495, 2009. URL <https://doi.org/10.1016/j.scico.2009.02.007>. [p]
- S. Schleimer, D. S. Wilkerson, and A. Aiken. Winnowing: Local algorithms for document fingerprinting. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, SIGMOD ’03, pages 76–85, 2003. URL <https://doi.org/10.1145/872757.872770>. [p]
- K. Serebryany, D. Bruening, A. Potapenko, and D. Vyukov. Addresssanitizer: A fast address sanity checker. In *Proc. USENIX Conference*, USENIX ATC’ 12, page 28, USA, 2012. USENIX Association. [p]
- N. Shervashidze, P. Schweitzer, E. J. van Leeuwen, K. Mehlhorn, and K. M. Borgwardt. Weisfeiler–Lehman graph kernels. *Journal of Machine Learning Research*, 12:2539–2561, 2011. [p]
- A. Signorell et al. *DescTools: Tools for Descriptive Statistics*, 2020. URL <https://cran.r-project.org/package=DescTools>. R package version 0.99.37. [p]
- J. Silge, J. C. Nash, and S. Graves. Navigating the R Package Universe. *The R Journal*, 10(2):558–563, 2018. URL <https://doi.org/10.32614/RJ-2018-058>. [p]
- K. Tian, M. Revelle, and D. Poshyvanyk. Using latent Dirichlet allocation for automatic categorization of software. In *2009 6th IEEE International Working Conference on Mining Software Repositories*, pages 163–166, 2009. URL <https://doi.org/10.1109/MSR.2009.5069496>. [p]
- Y. Ueda, T. Kamiya, S. Kusumoto, and K. Inoue. On detection of gapped code clones using gap locations. In *Ninth Asia-Pacific Software Engineering Conference, 2002.*, pages 327–336, 2002. URL <https://doi.org/10.1109/APSEC.2002.1183002>. [p]
- M. van der Loo. The stringdist package for approximate string matching. *The R Journal*, 6:111–122, 2014. [p]
- Vandana. A comparative study of plagiarism detection software. In *2018 5th International Symposium on Emerging Trends and Technologies in Libraries and Information Services*, pages 344–347. IEEE, 2018. URL <https://doi.org/10.1109/ETTLIS.2018.8485271>. [p]
- W. Venables and B. Ripley. *S Programming*. Springer, 2000. [p]
- I. Wegener. *Complexity Theory: Exploring the Limits of Efficient Algorithms*. Springer, 2005. [p]
- B. Weisfeiler and A. A. Lehman. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsia*, 2(9):12–16, 1968. [p]
- H. Wickham. *Advanced R*. Chapman & Hall/CRC, 2014. [p]
- H. Wickham and G. Grolemund. *R for Data Science*. O'Reilly, 2017. [p]
- M. J. Wise. Detection of similarities in student programs: YAP'ing may be preferable to Plague'ing. *ACM SIGCSE Bulletin*, 24(1):268–271, 1992. URL <https://doi.org/10.1145/135250.134564>. [p]

Maciej Bartoszuk
Faculty of Mathematics and Information Science,
Warsaw University of Technology
ul. Koszykowa 75, 00-662 Warsaw, Poland
<https://bartoszuk.rexamine.com>
<https://orcid.org/0000-0001-6088-8273>
m.bartoszuk@mini.pw.edu.pl

Marek Gagolewski
School of Information Technology,

*Deakin University
Geelong, VIC, Australia*
and
*Faculty of Mathematics and Information Science,
Warsaw University of Technology
ul. Koszykowa 75, 00-662 Warsaw, Poland*
<https://www.gagolewski.com>
<https://orcid.org/0000-0003-0637-6028>
m.gagolewski@deakin.edu.au

Linear Fractional Stable Motion with the **rlfsm** R Package

by Stepan Mazur and Dmitry Otryakhin

Abstract Linear fractional stable motion is a type of a stochastic integral driven by symmetric alpha-stable Lévy motion. The integral could be considered as a non-Gaussian analogue of the fractional Brownian motion. The present paper discusses R package **rlfsm** created for numerical procedures with the linear fractional stable motion. It is a set of tools for simulation of these processes as well as performing statistical inference and simulation studies on them. We introduce: tools that we developed to work with that type of motions as well as methods and ideas underlying them. Also we perform numerical experiments to show finite-sample behavior of certain estimators of the integral, and give an idea of how to envelope workflow related to the linear fractional stable motion in S4 classes and methods. Supplementary materials, including codes for numerical experiments, are available online. **rlfsm** could be found on CRAN and gitlab.

Introduction

The linear fractional stable motion (shortly, l fsm) $(X_t)_{t \in \mathbb{R}}$ on a filtered space $(\Omega, \mathcal{F}, (\mathcal{F}_t)_{t \in \mathbb{R}}, \mathbb{P})$ is defined via

$$X_t = \int_{\mathbb{R}} \left\{ (t-s)_+^{H-1/\alpha} - (-s)_+^{H-1/\alpha} \right\} dL_s, \quad x_+ := \max\{x, 0\}, \quad (1)$$

where L_s is a symmetric α -stable Lévy motion, $\alpha \in (0, 2)$, with the scaling parameter $\sigma > 0$ and the self-similarity parameter $H \in (0, 1)$. The l fsm is heavy-tailed process with infinite variance and long-range dependence. A good overview on the role which this process plays in natural sciences is done by [Watkins et al. \(2008\)](#). One could also find a review of stochastic properties of l fsm in [Mazur et al. \(2020\)](#).

We proceed with introduction to existing software, with interest towards study of numerical properties of statistical estimators for l fsm as the main motivation. So far, there is no standard approach for software development to operating the general class of stochastic processes driven by Lévy processes. Moreover, there was no systematic indexed and peer-reviewed software for simulating sample paths of l fsm and related estimators prior to **rlfsm**. There is a particularly simple and useful numerical algorithm for simulating lfsm developed by [Stoev and Taqqu \(2004\)](#). Other methods for simulation of the processes can be found in ([Wu et al., 2004](#)) and ([Biermé and Scheffler, 2008](#)). The paper ([Stoev and Taqqu, 2004](#)) contains a minimalistic implementation of l fsm generator as a MATLAB function. However, some useful packages, that could be used in numerical routines with Lévy-driven processes (e.g. to create l fsm generator and perform unit testing), exist and have been implemented in R. For instance, R package **somebm** ([Huang, 2013](#)) contains functions for generation of fractional Brownian motion (fBM). Currently archived by CRAN **dvfBm** ([Coeurjolly, 2009](#)) has routines for generation of fBm and estimator of the Hurst parameter of the latter. **stablist** ([Wuertz et al., 2016](#)) and **stable** ([Swihart et al., 2017](#)) contain different functions for stable distributions and random variables. A generator of random variables of the kind has been also implemented in MATLAB (see the code in Chapter 1.7 in ([Samorodnitsky and Taqqu, 1994](#))).

The paper is organized as follows. In Section 2 we present the simulation method for sample paths of l fsm and its implementation in our **path** function. Then, we present functions for finite sample studies of statistical estimators, and some other functions. Section 3 describes implementations of the high- and the low-frequency parameter estimators and discusses reasons behind their numerical behavior. Finally, in Section 4 we suggest an object oriented system that simplifies software programming of Lévy-driven integrals.

Basic R functions

Types of data we use

The latest version of the package (1.0.0) suggests that we work with two types of sample paths. In the low-frequency setting we only use points spaced 1 temporal index apart from each other, X_1, X_2, \dots, X_n . In the case of high-frequency, we use points with discretization equal to the length

of the path vector, $X_{1/n}, X_{2/n}, \dots, X_1$. This division is dictated by two issues: 1) the same division in the setting of limit theorems obtained by [Mazur et al. \(2020\)](#), and 2) the fact that there is no inference technique for an arbitrary mixture of the two frequencies. Consequently, temporal coordinates of low-frequency l fsm coincide with point index (compare `coordinates` and `point_num` in the example in Section 2.2) which varies from 0 to N . Analogously, in case of high-frequency scheme, temporal coordinates equal to point indexes divided by the total number of sampled points. When after sampling the index set is different from either $(1, 2, \dots, N)$ or $(1/n, 2/n, \dots, 1)$, rescaling in time should be performed using the equality $(a^H X_t)_{t \geq 0} \stackrel{d}{=} (X_{at})_{t \geq 0}$ with $a > 0$ provided that H is known or obtained via preliminary estimation.

Simulation method for the linear fractional stable motion

In this section, we start with a discussion on the simulation method of the l fsm proposed by [Stoev and Taqqu \(2004\)](#) which is implemented in R by us. In particular, simulation of sample paths is done via Riemann-sum approximations of its symmetric α -stable stochastic integral representation while Riemann-sums are computed efficiently by using the Fast Fourier Transform algorithm. In R, we introduce `path` function that creates sample paths of the l fsm. The idea underlying this sample path generator is that it should be always possible not only to obtain l fsm path, but also the underlying Lévy motion, generated during the procedure, and since the core function of l fsm is deterministic it should allow for l fsm path generation based on a given Lévy motion, and, in theory, otherwise (not always). For this reason generators of both processes were separated into independent parts (see Figure 1).

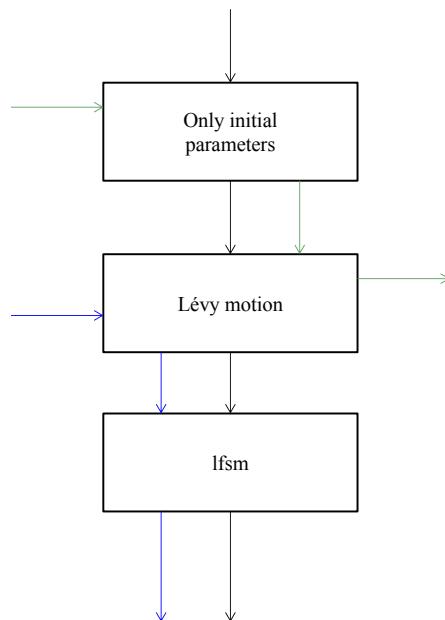


Figure 1: Scheme of generating Lévy motion and l fsm by path. Black arrows: when the algorithm initially is given the parameters, it generates Lévy motion, and then l fsm. Green arrows: when Lévy motion is needed without l fsm in order to save processing time, the algorithm bypasses computing of the later. Blue arrows: given a Lévy motion and some parameters, the generator computes the corresponding l fsm.

The function `path` can be used by

```
path(N,m,M,alpha,H,sigma,freq='L',disable_X=FALSE,
     levy_increments=NULL,seed=NULL)
```

Parameters `N`, `m`, `M` regard to the index of the process, or time, if applicable. `m` and `M` are the only means to control precision of the integral computation. `N` is a number of points of the l fsm to generate. `m` is a discretization parameter that corresponds to the number of points where Lévy motion is sampled between two nearby indexes (e.g. N and $N - 1$). `M` is the truncation parameter, i.e. number of points after which the integrated function is set to zero; `freq` stands for the frequency of the motion which can take two values: ‘`H`’ for high-frequency and ‘`L`’ for the low-frequency setting.

This is the switch between the two data types. `disable_X` is needed to disable computation of `X`, the default value is ‘`FALSE`’, when it is ‘`TRUE`’, only a Lévy motion is returned, which in turn reduces the computation time. `seed` is a parameter that performs seeding of the l fsm generator. Technically, in the `path` the seed is set just before Lévy increments are generated. The `path` function returns a list containing the l fsm, the underlying Lévy motion, the point number of the motions from 0 to N (`point_num`) and the corresponding coordinate which depends on the frequency, the parameters (σ, α, H) that were used to generate the l fsm, and the predefined frequency.

Generation of symmetric α -stable (sos) random variables is powered by function `rstable` from package `stabledist` with `S0` parametrization based on the Zolotarev’s representation for an α -stable distribution with some modifications. `S0` is used in order to make `sigma` a scale parameter of the motion and to get exempt from computing the normalization constant $C_{H,\alpha}$ presented in [Stoev and Taqqu \(2004\)](#) and is given by

$$C_{H,\alpha} := \left(\int_{\mathbb{R}} \left| (1-s)_+^{H-1/\alpha} - (-s)_+^{H-1/\alpha} \right|^{\alpha} ds \right)^{1/\alpha}.$$

The discrete convolution based algorithm and particularities of indexing

As it was mentioned in the beginning of Section 2.2, one of the features of `path` is the ability to operate on a pair l fsm - Lévy motion and to switch between them. We recall that direct computation of the sum approximating the integral in the definition of l fsm (1) would involve number of operations proportional to $N M m$, which makes the method slow. Instead, the original algorithm by [Stoev and Taqqu \(2004\)](#) suggests computing increments of l fsm with the help of

$$W(n) := \sum_{j=1}^{mM} a_{H,m}(j) Z_{\alpha}(n-j), \quad (2)$$

where $W(mk)$ is a discretized and truncated version of the increments of the l fsm, and in the limit has the same distribution as them

$$\{W(mk), k = 1, \dots, N\} \xrightarrow[m \rightarrow \infty; M \rightarrow \infty]{d} \{X(k) - X(k-1), k = 1, \dots, N\};$$

$Z_{\alpha}(k)$ are i.i.d. sos random variables that have indexes $-mM, \dots, mN - 1$ and scaling parameter equal to 1, and

$$a_{H,m}(j) := C_{H,\alpha}^{-1}(m, M) \left((j/m)^{H-1/\alpha} - (j/m-1)_+^{H-1/\alpha} \right) m^{-1/\alpha}, \quad j \in \mathbb{N}$$

with

$$C_{H,\alpha}(m, M) := m^{-1} \left(\sum_{j=1}^{mM} \left| (j/m)^{H-1/\alpha} - (j/m-1)_+^{H-1/\alpha} \right|^{\alpha} \right)^{1/\alpha}.$$

	-2	-1	0	1	2	3	4	5	index
Z	7	9	1	3	8	2	6	8	
a			4	9	2				
W(n)			99	39	61	86	58	90	

Figure 2: Example of direct computation of sum of the form (2) for 2 vectors. a corresponds to the kernel and Z - to the Lévy motion.

Let us consider an example which will recur and evolve throughout this section. Consider computing sum (2) where $m = 1$, $M = 3$, and $N = 6$ (see Figure 2). The two rightmost cells for $W(n)$ are left empty because there is no sense in computing them without truncation of a .

A method based on the discrete convolution theorem is used to obtain $W(mk)$. The theorem relies on Discrete Fourier Transform (DFT), which needs to perform a number of operations proportional to $(mN + mM) \log(mN + mM)$ instead of $N M m$. In order to understand how this method works, we review several definitions and theorems.

Definition 2.1 For any sequence x_n , $n \in \mathbb{N}$, Discrete-Time Fourier Transform (DTFT) is defined

as

$$X = \text{DTFT}\{x_n\}(\omega) = \sum_{n=-\infty}^{\infty} x_n \exp(-in\omega).$$

The reverse transform, IDTFT, is defined as

$$x_n = \text{IDTFT}\{X\} = \frac{1}{2\pi} \int_0^{2\pi} X(\omega) e^{i\omega n} d\omega.$$

Definition 2.2 Discrete convolution of two infinite sequences $\{A_n\}_{n \in \mathbb{N}}$ and $\{B_n\}_{n \in \mathbb{N}}$ is

$$(A * B)[n] := \sum_{m=-\infty}^{\infty} A[m]B[n-m].$$

There is a convolution theorem for discrete sequences which says that the discrete convolution of two sequences is equal to the Inverse Discrete Fourier Transform (IDFT) of the multiplication of the direct transforms of the sequences:

Theorem 2.3 For any discrete sequences x_n and y_n , $n \in \mathbb{N}$, it holds that

$$(x * y)[n] = \text{IDTFT}[\text{DTFT}\{x_n\}(\cdot) \times \text{DTFT}\{y_n\}(\cdot)].$$

Definition 2.4 Let x_n , $n \in \mathbb{N}$ be a sequence. Then $\{x_N\}[n]$, $n \in \mathbb{N}$ is called N -periodic summation of the sequence:

$$\{x_N\}[n] := \sum_{k \in \mathbb{N}} x[n + kN].$$

It is straightforward that the periodic summation in the definition above has period N . In our case, the latter theorem is applicable even though we will be interested in a finite sequence of length \tilde{N} . The sequence is padded with zeros to form an infinite one, and a periodic summation of a the length \tilde{N} is just a periodic extension of it.

Figure 3: Example of periodic summation of a zero-padded finite sequence where the period equals to the sequence length ($N = \tilde{N}$).

DTFT is not directly useful for simulation purpose, that is why we need a special case of Theorem 2.3, Circular Convolution Theorem which reduces DTFT to DFT.

Definition 2.5 The DFT of a finite sequence x_n of length N is defined as

$$X_k = \text{DFT}_k(x_n) := \sum_{n=0}^{N-1} x_n \exp(-2\pi i kn/N).$$

The IDFT is

$$x_n := \frac{1}{N} \sum_{k=0}^{N-1} X_k \exp(2\pi i kn/N).$$

Theorem 2.6

$$(x_N * y)[n] = \text{IDFT}\{\text{DFT}(x_N)\text{DFT}(y_N)\}$$

Returning to the task of computing the sum in (2), we consider two vectors: a of length mM and Z of length $m(M+N)$. Here, we again index vectors starting with zero, not one. If we extend Z periodically, pad a with zeros to make an infinite sequence, and compute $(a * Z_{m(N+M)})[n]$, values with indexes $[mM; m(N+M)-1]$ would coincide with the result of a convolution of a and Z . The first mM values would be meaningless. This gives an idea how to use Circular Convolution Theorem for computation of (2): instead of $a * Z$ we compute one period of $(a * Z_{m(N+M)})[n]$ through the left part of 2.6 and leave only meaningful values. Figure 4 illustrates the use of Circular Convolution Theorem with periodic extensions of Z and padded a to compute (2). In this case results with indexes -1 and -2 are meaningless and should be discarded.

-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	index
7	9	1	3	8	2	6	8	7	9	1	3	8	2	6	8	7	9	1	3	8	2	6	8	
0	0	4	9	2	0	0	0	0	0	4	9	2	0	0	0	0	0	4	9	2	0	0	0	
z a Z_N a_N																								

Figure 4: Example of transformation of vectors a and Z into sequences before computing their convolution.

Although the setup of the example as is on Figure 4 is fastest, it is impossible to use it directly, because in some situations truncation parameter M is larger than N , the number of points of l fsm sample path that is needed to be simulated. In this case `path` function performs an index shift using the following property:

$$(a * x_c)[n] := \sum_{k=-\infty}^{+\infty} a[k] \cdot x[n - k - c] = \sum_{k=-\infty}^{+\infty} a[k] \cdot x[\tilde{n} - k] = (a * x)[\tilde{n} - c] \quad (3)$$

This property is illustrated by Figure 5, wherein sequence $x[n]$ is shifted by 2 to the right, so $c = 2$. Accordingly, the resulting convolution also gets shifted 2 notches to the right (compare Figures 5 and 2). In general, according to (3), when $x[n]$ is shifted to assign index zero to the first value, the resulting convolution sequence also starts from the first meaningful value. Thus, `path` always keeps the first Nm as the result of convolution operation and discards the rest.

-8	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	index
7	9	1	3	8	2	6	8	7	9	1	3	8	2	6	8	7	9	1	3	8	2	6	8	
0	0	0	0	0	0	0	0	4	9	2	0	0	0	0	0	0	0	0	0	0	0	0	0	

Figure 5: Example of index shift in `path` function.

Examples

In the next example, we show how one can use the above function to generate a sample path and to provide its visualization. Compare the procedure with the similar one from Section 4.1.

```
# Path generation
List<-path(N=2^10-600,m=256,M=600,alpha=1.8,H=0.8,
            sigma=1,freq='L',disable_X=FALSE,seed=3)
str(List)

List of 7
$ point_num      : int [1:425] 0 1 2 3 4 5 6 7 8 9 ...
$ coordinates    : int [1:425] 0 1 2 3 4 5 6 7 8 9 ...
$ l fsm          : num [1:425] 0 -1.3969 0.0159 1.6487 1.87 ...
$ levy_motion    : num [1:425] 0 -21.8 28.3 42.1 38.1 ...
$ levy_increments: num [1:262144] -0.292 -0.708 -1.49 0.517 0.803 ...
$ pars           : Named num [1:3] 1.8 0.8 1
  ..- attr(*, "names")= chr [1:3] "alpha" "H" "sigma"
$ frequency      : chr "L"

# Normalized paths
Norm_l fsm<-List[['l fsm']] / max(abs(List[['l fsm']]))
Norm_oLm<-List[['levy_motion']] / max(abs(List[['levy_motion']]))

# Visualization of the paths
plot(Norm_l fsm, col=2, type="l", ylab="coordinate")
lines(Norm_oLm, col=3)
leg.txt <- c("l fsm", "oLm")
legend("topright", legend = leg.txt, col =c(2,3), pch=1)
```

The result of the chart rendering is shown on Figure 6. The following example shows how to switch `path` function in order to alter between simulation of l fsm from scratch and computing based

on an existing sample path of the Lévy motion.

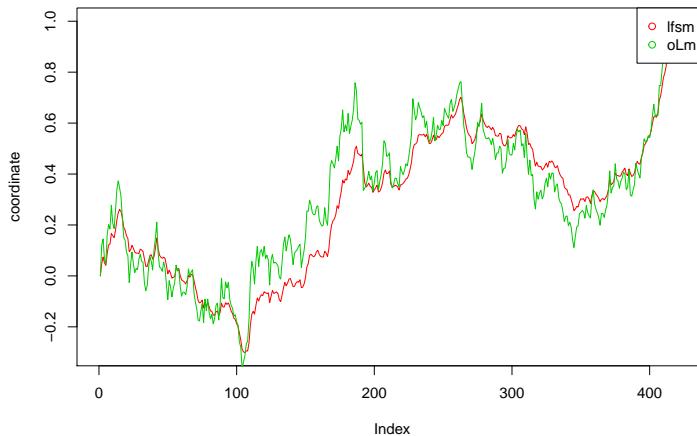


Figure 6: Plot of sample path and Lévy motion with seed=2

```
m<-256; M<-600; N<-2^12-M
alpha<-1.8; H<-0.8; sigma<-1.8
seed<-2

# Creating Levy motion
levyIncrems<-path(N=N, m=m, M=M, alpha, H, sigma, freq='L',
                    disable_X=T, levy_increments=NULL, seed=seed)

# Creating lfsm based on the levy motion
lfsm_full<-path(m=m, M=M, alpha=alpha,
                  H=H, sigma=sigma, freq='L',
                  disable_X=F,
                  levy_increments=levyIncrems$levy_increments,
                  seed=seed)

sum(levyIncrems$levy_increments==
    lfsm_full$levy_increments)==length(lfsm_full$levy_increments)

[1] TRUE
```

In the example the Lévy motion is generated without computing the lfsm, which was done by setting `disable_X=TRUE`, and saved to variable `levyIncrems`. After that, `path` was given the obtained Lévy increments and, basing on them, generated an lfsm path. As one can observe, the Lévy increments from the both objects produced by `path` are identical. The same holds when we obtain an lfsm path from the above procedure and one-step simulation of lfsm with seeding. These two facts are used in automated tests provided for `rlfsm` package.

MCestimLFSM and numerical properties of statistical estimators

In order to study numerical properties of the estimation procedures developed in Mazur et al. (2020), we created a technique, that could be used in solving this problem for any pair stochastic process and an estimator. The approach was implemented in `MCestimLFSM` function (Figure 9). The main motivation here is that for some estimators we have limit theorems, but we do not have theory which describes estimator behavior when the length of a path is relatively small, and thus, for instance, we cannot use closed-form expressions to obtain confidential intervals. In the following examples we show how to use functions `MCestimLFSM`, `PLot_vb`, and `Plot_dens` for studying empirical variance, bias and a density function of an estimator. In the first example, we study `GenLowEstim` estimator, and its bias and variance dependencies on the length of the sample paths. In particular, one would be able to determine starting from which path length the estimator loses significant bias influence.

```

library(rlfsm)
library(gridExtra)
registerDoParallel()

m<-25; M<-55
p<-.4; p_prime<-.2
t1<-1; t2<-2
k<-2

NmonteC<-5e2
alpha<-1.8; H<-0.8; sigma<-0.3

S<-seq(from = 100, to = 2e3, by =50)
tilda_est<-MCestimLFSM(s=S, fr='L', Nmc=NmonteC, m=m, M=M,
                        alpha=alpha,H=H,sigma=sigma,
                        GenLowEstim,t1=t1,t2=t2,p=p)

# Structure of tilda_est
names(tilda_est)
[1] "data"      "data_nor"   "means"      "sds"        "biases"    "Inference" "params"     "freq"

# Structure of BSdM is as follows

head(round(tilda_est$means,2))
  alpha      H sigma   s
1 1.76 0.67 0.25 100
2 1.81 0.70 0.27 150
3 1.81 0.71 0.27 200
4 1.82 0.73 0.28 250
5 1.83 0.74 0.28 300
6 1.83 0.75 0.29 350

head(round(tilda_est$biases,2))
  alpha      H sigma   s
1 -0.04 -0.13 -0.05 100
2  0.01 -0.10 -0.03 150
3  0.01 -0.09 -0.03 200
4  0.02 -0.07 -0.02 250
5  0.03 -0.06 -0.02 300
6  0.03 -0.05 -0.01 350

head(round(tilda_est$sds,2))
  alpha      H sigma   s
1 0.19 0.23 0.09 100
2 0.14 0.20 0.08 150
3 0.13 0.19 0.08 200
4 0.13 0.19 0.07 250
5 0.10 0.17 0.06 300
6 0.11 0.17 0.06 350

Plot_vb(tilda_est)

```

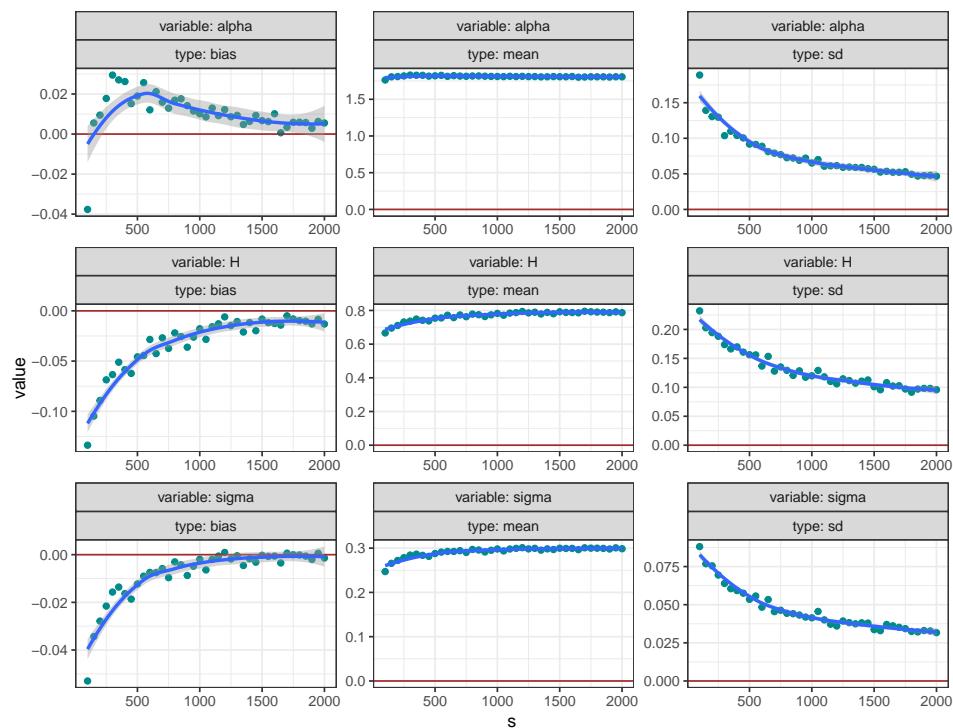


Figure 7: Variance and bias dependence on path length of tilde- estimators, described in Section 3.2.

Figure 7 shows that when $(\sigma, \alpha, H) = (0.3, 1.8, 0.8)$, estimator `GenLowEstim` could be considered unbiased starting approximately from 1000 points.

The second example compares empirical standardized densities of estimates, obtained by `GenLowEstim` with the limiting standard normal ones, Figure 8.

```
S<-c(1e2,1e3,1e4)
tilda_ests<-MCestimLFSM(s=S, fr='L', Nmc=NmonteC ,m=m, M=M,
                           alpha=alpha, H=H, sigma=sigma,
                           GenLowEstim,t1=t1,t2=t2,p=p)

l_plot<-Plot_dens(par_vec=c('sigma','alpha','H'), MC_data=tilda_ests,
                    Nnorm=1e7)
ggg<-grid.arrange(l_plot[[1]],l_plot[[2]],l_plot[[3]],nrow=1,ncol=3)
```

In short, in these examples for different path lengths s , `NmonteC` l fsm paths are simulated. To each path we apply tilde-statistic (see Section 3.2), therefore obtaining `NmonteC` estimates $(\tilde{\sigma}_{low}, \tilde{\alpha}_{low}, \tilde{H}_{low})$ for every s , which in turn, are used to calculate biases, standard deviations, and density functions (also, for each s separately).

MCestimLFSM architecture and optimization

It is important to notice that generation of l fsm is numerically heavy routine and also a large number of estimates is needed to compare their empirical distributions with the limiting ones. The latter task gave `MCestimLFSM` its name. Thus, in order to make computations feasible in terms of time and memory use, the architecture of `MCestimLFSM` must be well-optimized. Apparently, a multi-core setup is crucial for dealing with the task.

Having fixed a path length, the whole procedure behind `MCestimLFSM` could be split in two parts. First, we need to obtain samples for each estimator. Second, we obtain statistics of these samples (see Figure 9). Once finished, `MCestimLFSM` proceeds to the next length value until reaches the end of the vector of lengths.

In the first part, we generate $N_{Monte Carlo}$ l fsm paths of the length $s[i]$ via `path_fast` function. To each of the paths we apply all the estimators to obtain H , α , and σ estimates. During this stage, we use a `foreach`-based parallel loop, where each node simulates a path, computes and returns the statistics removing the path from memory. `path_fast` is an unavailable for users version of `path`

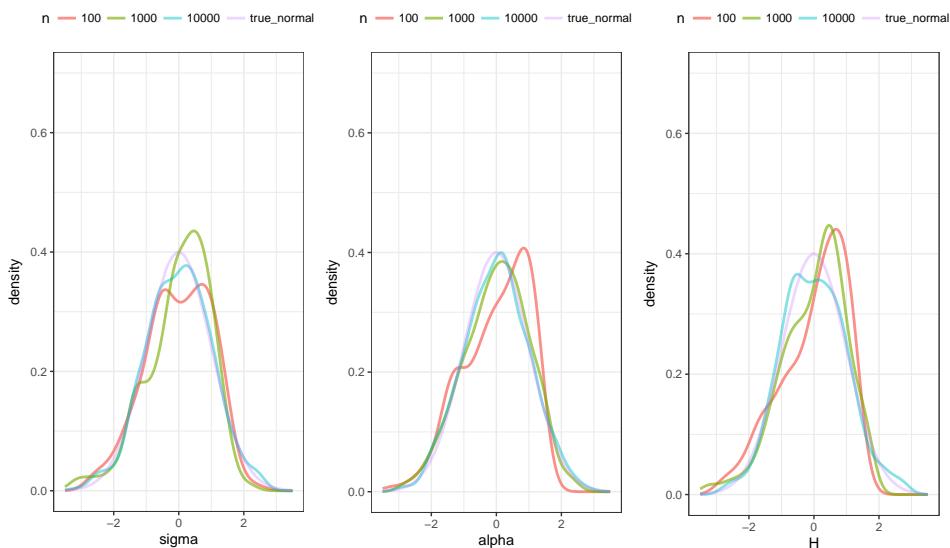


Figure 8: Empirical distributions of tilde-estimates, described in Section 3.2.

with significantly reduced functionality for the sake of saving execution time. The further desired enlargement of the node task by adding generation of the whole set of paths instead of just one, making the loop over `s[i]` parallel, leads to extreme memory consumption as well as unequal distribution of load among nodes. The number of numeric values in the set of paths equals to $N_{\text{Monte Carlo}} \times s[i]$. Simulations, performed in Mazur et al. (2020) showed that normal distribution is attained by estimators at $s = 10^3$. Given the fact that we need at least 10^5 Monte Carlo trials for a neat histogram of a distribution, one can obtain the amount of memory required to store a matrix of size $N_{\text{Monte Carlo}} \times s[i]$, which makes 763Mb, while some estimators require 80Gb per node. That is the reason why in the current version of `MCestimLFSM` the loop over `s` is sequential, and the one over `NmonteC` is parallel.

During the second part, averages and standard deviations of the samples are computed, and subsequently used to compute the standardized empirical distributions. So that, the three characteristics naturally come together within the same numerical procedure. So far there is no empirical evidence that parallel execution in this section makes `MCestimLFSM` more efficient.

Such architecture is of great use when the number of nodes available for computations exceeds the number of path length, and the length $s[i]$ differs significantly from $s[j]$ when $i \neq j$.

On some of the other basic functions

In this part, we will describe aspects of some of the other R functions implemented in the package.

Higher-order increments

These increments are the main building block for all statistics we use (see Section 3). They are defined as k -th iterated increments of step r of a sample path. In particular, $\Delta_{i,1}^{n,1} X := X_{\frac{i}{n}} - X_{\frac{i-1}{n}}$, and $\Delta_{i,2}^{n,1} X := X_{\frac{i}{n}} - 2X_{\frac{i-1}{n}} + X_{\frac{i-2}{n}}$. In `rlfsm`, we built two functions for computation of objects of this class- `increment()` and `increments()`. The former accepts a vector of points at which a user wants to evaluate higher-order increments, and computes them using formula

$$\Delta_{i,k}^{n,r} X := \sum_{j=0}^k (-1)^j \binom{k}{j} X_{(i-rj)/n}. \quad (4)$$

Before evaluation of (4), the function checks the condition $i < kr$. Evaluation of the increments on a sample path of length N takes $(k+1)(N - kr)$ operations- $k+1$ sums for $N - kr$ points. `increments()` computes increments iteratively on the whole set of path points. The first iteration gives $N - r$ increments, the second- $N - 2r$ and so on. Thus, the total number of performed

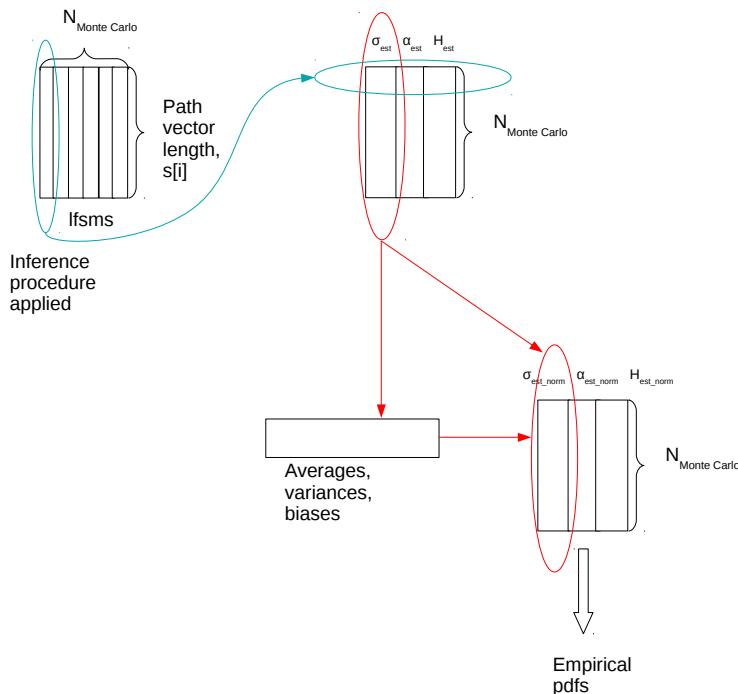


Figure 9: Scheme of extracting estimator statistics by function MCestimLFSM for a chosen path length.

operations is

$$\sum_{j=1}^k (N - jr) = kN - r(k+1)k/2.$$

It is clear that `increments()` is faster on sample paths with large number of points, but slower when the increment order is high. As we will show later, orders greater than ~ 10 are not usable for statistical inference. That is the reason why in all statistics we use either `increments()` or its hidden “relatives”.

A visualization method for sample paths

We introduce a pair of functions which makes a panel plot of sample paths produced by processes with different parameters. `Path_array` takes a set of α - H values, generates a path for each combination, and stacks the paths together in a data frame. In the produced data frame all the paths are tagged with α and H values. `Plot_list_paths()` takes the data frame as an argument and plots the sample paths on different panels based on their (α, H) values. This functionality is powered by `facet_wrap()` from `ggplot2` (Wickham, 2016). For discontinuous paths `Plot_list_paths()` draws an overlapping semitransparent line joining neighbouring points in order to highlight jumps.

```

l=list(H=c(0.2,0.5,0.8), alpha=c(0.5,1,1.5), freq="H")
arr<-Path_array(N=300, m=30, M=100, l=l, sigma=0.3)
head(arr)
      n          X alpha     H freq
1 1  0.0000000  0.5 0.2    H
2 2  0.2329891  0.5 0.2    H
3 3  1.1218238  0.5 0.2    H
4 4 -6.1284620  0.5 0.2    H
5 5 -2.2450357  0.5 0.2    H
6 6  3.4979978  0.5 0.2    H

str(arr)
'data.frame':      2709 obs. of  5 variables:
 $ n      : num  1 2 3 4 5 6 7 8 9 10 ...

```

```
$ X      : num  0 0.233 1.122 -6.128 -2.245 ...
$ alpha: Factor w/ 3 levels "0.5","1","1.5": 1 1 1 1 1 1 1 1 1 ...
$ H     : Factor w/ 3 levels "0.2","0.5","0.8": 1 1 1 1 1 1 1 1 1 ...
$ freq : Factor w/ 1 level "H": 1 1 1 1 1 1 1 1 1 ...

Plot_list_paths(arr)
```

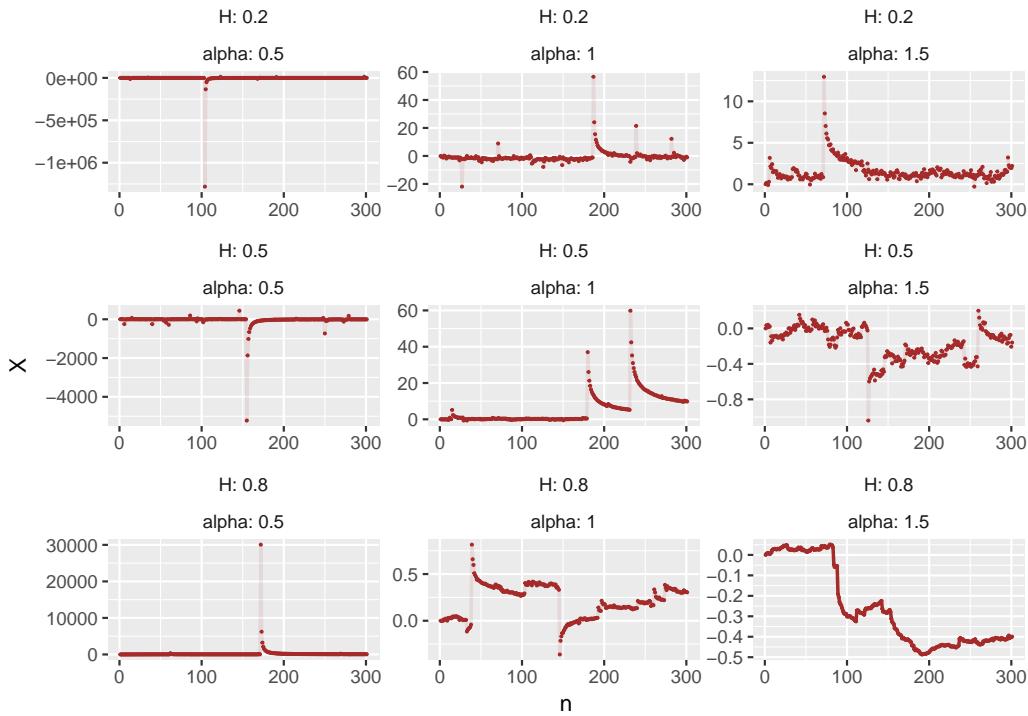


Figure 10: Graph rendered by Plot_list_paths

Parameter estimation of the linear fractional stable motion

In this section, we describe estimators for the parameters H , α , and σ that are obtained in the recent paper by [Mazur et al. \(2020\)](#), and their implementation in R.

Parameter estimation in the continuous case

First, we consider the case $H - 1/\alpha > 0$ which leads us to the important property that the lfsm $(X_t)_{t \in R}$ is locally Hölder continuous of any order up to $H - 1/\alpha$. Moreover, this condition implies the following restrictions

$$\alpha \in (1, 2) \quad \text{and} \quad H \in (1/2, 1) \tag{5}$$

that allow us to use the law of large numbers in Theorem 1.1 of ([Basse-O'Connor et al., 2017](#)) when $p < 1$, and the central limit theorem in Theorem 1.2 of ([Basse-O'Connor et al., 2017](#)) when $p < 1/2$, $k \geq 2$ and $H < k - 1/\alpha$.

Now, we consider consistent estimators for the self-similarity parameter H in high- and low-frequency setting, defined by

$$\begin{aligned} \widehat{H}_{high}(p, k)_n &:= \frac{1}{p} \log_2 \left(\frac{\sum_{i=2k}^n |\Delta_{i,k}^{n,2} X|^p}{\sum_{i=2k}^n |\Delta_{i,k}^{n,1} X|^p} \right), \\ \widehat{H}_{low}(p, k)_n &:= \frac{1}{p} \log_2 \left(\frac{\sum_{i=2k}^n |\Delta_{i,k}^2 X|^p}{\sum_{i=2k}^n |\Delta_{i,k}^1 X|^p} \right). \end{aligned}$$

Both estimators for H are based upon a ratio statistic that compares power variations at two different frequencies.

Let us define the following two statistics

$$V_{high}(f; k, r)_n := \frac{1}{n} \sum_{i=rk}^n f \left(n^H \Delta_{i,k}^{n,r} X \right) \quad V_{low}(f; k, r)_n := \frac{1}{n} \sum_{i=rk}^n f \left(n^H \Delta_{i,k}^r X \right), \quad (6)$$

where $f : \mathbb{R} \rightarrow \mathbb{R}$ is a measurable function. Estimators for the stability index α of the driving stable motion in high and low frequency setting are based on the empirical characteristic functions given by

$$\varphi_{high}(t; H, k)_n := V_{high}(\psi_t; k)_n \quad \text{and} \quad \varphi_{low}(t; k)_n := V_{low}(\psi_t; k)_n \quad (7)$$

with $\psi_t(x) := \cos(tx)$, for two different values t_1 and t_2 such that $t_2 > t_1 > 0$. Let us note that the empirical characteristic function $\varphi_{high}(t; H, k)_n$ depends on the parameter H while $\varphi_{low}(t; k)_n$ does not. Thus, we should infer the self-similarity parameter H by $\widehat{H}_{high}(p, k)_n$ and then we should use the plug-in estimator $\varphi_{high}(t; \widehat{H}_{high}(p, k)_n, k)_n$ to infer the stability index α in high-frequency setting. Estimators for the parameter α are given by

$$\begin{aligned} \widehat{\alpha}_{high} &:= \frac{\log |\log \varphi_{high}(t_2; \widehat{H}_{high}(p, k)_n, k)_n| - \log |\log \varphi_{high}(t_1; \widehat{H}_{high}(p, k)_n, k)_n|}{\log t_2 - \log t_1}, \\ \widehat{\alpha}_{low} &:= \frac{\log |\log \varphi_{low}(t_2; k)_n| - \log |\log \varphi_{low}(t_1; k)_n|}{\log t_2 - \log t_1}. \end{aligned}$$

Estimators for the scale parameter σ in high- and low-frequency are also based on the empirical characteristic functions which are defined for one value of $t > 0$. Further, we define a function $h_{k,r} : R \rightarrow R$ as follows:

$$h_{k,r}(x) = \sum_{j=0}^k (-1)^j \binom{k}{j} (x - rj)_+^{H-1/\alpha}, \quad x \in R, \quad (8)$$

where $k, r \in N$, and let $\|h_{k,r}\|_\alpha^\alpha := \int_R |h_{k,r}(s)|^\alpha ds$. Let us note that the function $h_{k,r}$ depends on two parameters α and H which need to be pre-estimated. Estimators for the parameter σ are expressed as

$$\begin{aligned} \widehat{\sigma}_{high} &:= \left(-\log \varphi_{high}(t_1; \widehat{H}_{high}(p, k)_n, k) \right)^{1/\widehat{\alpha}_{high}} / t_1 \|h_{k,1}\|_{\widehat{\alpha}_{high}}, \\ \widehat{\sigma}_{low} &:= \left(-\log \varphi_{low}(t_1; k) \right)^{1/\widehat{\alpha}_{low}} / t_1 \|h_{k,1}\|_{\widehat{\alpha}_{low}}. \end{aligned}$$

Parameter estimation in the general case

Here, we consider general case when an explicit lower bound for α is unknown. First, we consider estimators which are obtained in low frequency setting. Consistent estimator for parameter H for any $p \in (1, 1/2)$ is obtained by

$$\widehat{H}_{low}(-p, k)_n := \frac{1}{p} \log_2 \left(\frac{\sum_{i=2k}^n |\Delta_{i,k}^2 X|^{-p}}{\sum_{i=2k}^n |\Delta_{i,k}^1 X|^{-p}} \right).$$

Next, we consider two-step procedure to choose the order of increments k , since we should be in the domain of attraction of Theorem 1.2 of (Basse-O'Connor et al., 2017) that requires $k > H + 1/\alpha$. That's why we consider the preliminary estimator of α with $k = 1$ that is consistent given by

$$\widehat{\alpha}_{low}^0(t_1, t_2)_n = \frac{\log |\log \varphi_{low}(t_2; 1)_n| - \log |\log \varphi_{low}(t_1; 1)_n|}{\log t_2 - \log t_1}.$$

Since we do not know if $\widehat{\alpha}_{low}^0(t_1, t_2)_n$ is in the domain of attraction, we define the estimator of the parameter k as

$$\widehat{k}_{low}(t_1, t_2)_n := 2 + \lfloor \widehat{\alpha}_{low}^0(t_1, t_2)_n^{-1} \rfloor.$$

In the second step we use estimator $\hat{k}_{low} := \hat{k}_{low}(t_1, t_2)_n$ for the estimation of parameters H , α and σ . In particular, we get the following consistent estimators

$$\begin{aligned}\hat{H}_{low}(-p, \hat{k}_{low})_n &= \frac{1}{p} \log_2 \left(\frac{\sum_{i=2\hat{k}_{low}}^n |\Delta_{i,\hat{k}_{low}}^2 X|^{-p}}{\sum_{i=2\hat{k}_{low}}^n |\Delta_{i,\hat{k}_{low}}^1 X|^{-p}} \right), \\ \tilde{\alpha}_{low}(\hat{k}_{low}; t_1, t_2)_n &= \frac{\log |\log \varphi_{low}(t_2; \hat{k}_{low})_n| - \log |\log \varphi_{low}(t_1; \hat{k}_{low})_n|}{\log t_2 - \log t_1}, \\ \tilde{\sigma}_{low}(\hat{k}_{low}; t_1, t_2)_n &= (-\log \varphi_{low}(t_1; \hat{k}_{low}))^{1/\tilde{\alpha}_{low}} / t_1 \|h_{\hat{k}_{low}, 1}\|_{\tilde{\alpha}_{low}}.\end{aligned}$$

Next, we consider two-stage estimation procedure in the general case in high-frequency setting which is the same as in the low-frequency setting. For $p \in (0, 1/2)$ we compute $\hat{H}_{high}(-p)_n = \hat{H}_{high}(-p, 1)_n$ and, therefore, we can define the preliminary estimator of α by

$$\hat{\alpha}_{high}^0(p, p')_n = \phi^{-1} \left(\frac{V_{high}(f_{-p'}, \hat{H}_{high}(-p)_n)_n^p}{V_{high}(f_{-p}, \hat{H}_{high}(-p)_n)_n^{p'}} \right)$$

with

$$\phi(\hat{\alpha}_{high}^0(p, p')_n) := \frac{(2/\hat{\alpha}_{high}^0(p, p')_n)^{p-p'} a_{-p}^{p'} \Gamma(p'/\hat{\alpha}_{high}^0(p, p')_n)_n^p}{a_{-p}^p \Gamma(p/\hat{\alpha}_{high}^0(p, p')_n)_n^{p'}}$$

where $p, p' \in (0, 1/2)$ such that $p \neq p'$, and $V_{high}(f_{-p}, \hat{H}_{high}(-p)_n)_n$ is given in formula (6) with $k = 1$, $f_{-p}(x) = |x|^{-p}$ and preliminary estimator $\hat{H}_{high}(-p)_n$ for the parameter H . It is remarkable that $\phi(\cdot)$ is always invertible for all $p \neq p'$ (see Dang and Ista (2017)). Consequentially, we can define the estimator of k in high-frequency setting by

$$\hat{k}_{high} := \hat{k}_{high}(p, p')_n = 2 + [\hat{\alpha}_{high}^0(p, p')_n]^{-1}.$$

Thus, consistent estimators of H , α and σ , in high-frequency setting are given by

$$\begin{aligned}\hat{H}_{high}(-p, \hat{k}_{high})_n &= \frac{1}{p} \log_2 \left(\frac{\sum_{i=2\hat{k}_{high}}^n |\Delta_{i,\hat{k}_{high}}^{n,2} X|^{-p}}{\sum_{i=2\hat{k}_{high}}^n |\Delta_{i,\hat{k}_{high}}^{n,1} X|^{-p}} \right), \\ \tilde{\alpha}_{high}(\hat{k}_{high}; t_1, t_2)_n &= \phi^{-1} \left(\frac{V_{high}(f_{-p'}, \hat{H}_{high}(-p, \hat{k}_{high})_n; \hat{k}_{high})_n^p}{V_{high}(f_{-p}, \hat{H}_{high}(-p, \hat{k}_{high})_n; \hat{k}_{high})_n^{p'}} \right), \\ \tilde{\sigma}_{high}(\hat{k}_{high}; p, p')_n &= \left(\frac{\tilde{\alpha}_{high} a_{-p} V_{high}(f_{-p}, \hat{H}_{high}(-p)_n)_n}{2\Gamma(p/\tilde{\alpha}_{high})} \right)^{-\frac{1}{p}} / \|h_{\hat{k}_{high}, 1}\|_{\tilde{\alpha}_{high}}.\end{aligned}$$

Implementation in R

We introduce function `ContinEstim` for performing statistical inference according to Section 3.1 when $H - 1/\alpha > 0$.

```
ContinEstim(t1, t2, p, k, path, freq)
```

The function is basically comprised by simpler functions `alpha_hat`, `H_hat` and `sigma_hat` responsible for retrieving the corresponding parameters. `sigma_hat` is called using `tryCatch` as the former may return an error due to numerical integration in `Norm_alpha`.

General low-frequency estimation technique, described in Section 3.2 is implemented in `GenLowEstim`.

```
GenLowEstim(t1, t2, p, path, freq = "L")
```

This estimator first sets a preliminary k to be equal to 1, and uses it to compute preliminary parameters H_0 and α_0 . Using these H_0 and α_0 , a new k is obtained through `2+floor(alpha_0*(-1))`, and then the new k is used for the same estimation procedure as in `ContinEstim`. This approach induces an effect, which does not exist in the case when `ContinEstim` is applied. When α is smaller than, or close to $2/N$, where N is the observed Ifsm path length, the computational errors are more frequent. These extra errors occur when the preliminary estimation of k appears to exceed $N/2$, making it impossible to compute $\Delta_{i,\hat{k}_{low}}^2 X$ in statistic $\hat{H}_{low}(-p, \hat{k}_{low})_N$. In case of other sample path realizations $k < H + 1/\alpha$, and it is still possible to obtain the estimates which happen to

converge to the true value $(\hat{H}, \hat{\alpha}, \hat{\sigma})$, because in this case one would be in the domain of attraction of Theorem 2.2 of (Mazur et al., 2020). Though, the limiting distribution is not stable anymore, and the rate of convergence depends on α and H . Real distributions of estimates in this case are left unexplored.

High-frequency estimator from the same section was implemented in `GenHighEstim`.

```
GenHighEstim(p, p_prime, path, freq, low_bound = 0.01, up_bound = 4)
```

Estimate deterioration

Although the general high- and low-frequency estimators presented in Section 3.2 have important advantages, namely closed form expressions for distribution functions and non-suboptimal convergence rates, they also reveal two drawbacks in performance. Due to condition and error handling, the time performances of the general estimators are much worse than those of the continuous ones. On top of that, the plug-in estimators (because of their nature) have much less probability of obtaining an estimate at all. The main idea is as follows: the more statistics are used in a plug-in estimator, the higher the probability to stumble upon a numerical error during the estimation procedure. We illustrate this effect by the following experiment, wherein the general high- and low-frequency estimators are compared to the corresponding continuous ones. For each pair from a set of parameters (H, α) , `NmonteC` sample paths of the both frequencies were generated, and to each of them the relevant procedures `ContinEstim`, `GenLowEstim` and `GenHighEstim` were applied (see “Estimate deterioration experiment” in the supplementary materials). Then, the rates of successful computation results were computed. The result of estimation was considered “successful” if during the procedure all three parameters were obtained, no error occurred, and the estimates are meaningful, namely $(\hat{H}, \hat{\alpha}) \in (0, 1) \times (0, 2)$.

This experiment shows (Figures 11a and 11b) that in both high- and low-frequency cases `ContinEstim` gives much better precision than the corresponding general estimator. The outcome is rigorous in low-frequency technique since `ContinEstim` and `GenLowEstim` have the same set of tuning parameters. On the other hand, the high-frequency estimators have non-coinciding parameter sets, and thus, without fine tuning, the result is merely intuitive. One could observe that in general estimation near the boundaries of the interval $(\hat{H}, \hat{\alpha}) \in (0, 1) \times (0, 2)$ produces more errors, which is partly due to the fact that near the boundaries it is easier to obtain an estimate outside the interval. Such an estimate is removed by `Errfilter` function in the experiment.

Zones with different convergence regimes in the low-frequency case

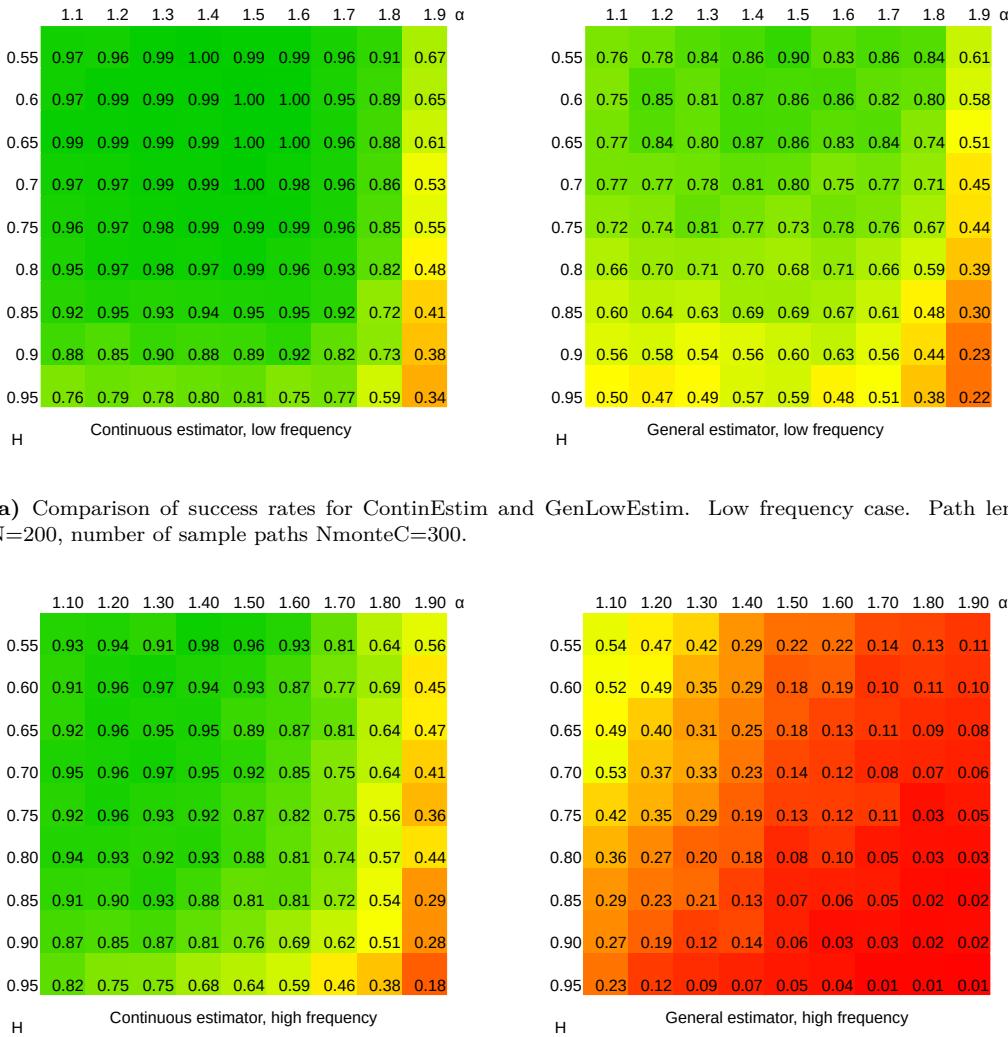
In order to show how the general low-frequency estimation works in practice, we perform a numerical experiment whose code could be found in section “Zones with different convergence” of the accompanying .R file. We set a constant σ and choose two sets of parameters- one for α and one for H . Then, for each combination of them a number $N_{mc} = 500$ of sample paths is created. All path lengths are set to a constant $N = 1000$. To each path we apply several statistics. One of them is `k_new<-2+floor(alpha_0(-1))` where α_0 is obtained via `alpha_hat` with parameters `k=1`, `freq='L'` plugged-in. This provides us simulated distribution of \hat{k}_{low} (Figure 12). Also, we fix a set `k_ind = seq(1,8,by=1)` and, given a path, for each of these k 's extract statistics $\varphi_{low}(t, k = k_{ind})_n$ and $\hat{\alpha}_{low}(t_1, t_2; k = k_{ind})_n$, see Figures 13 and 14.

Three regimes of performance of `GenLowEstim` (read, the general low-frequency estimator $\hat{\alpha}_{low}(k, t_1, t_2)_n$) are observed. To a large extend, only parameter α determines which regime is in presence.

Due to small variance of $\hat{\alpha}_{low}^0(t_1, t_2)_n$ (Figure 14), when $\alpha \in (1, 2)$ the estimation $\hat{k}_{low}(t_1 = 1, t_2 = 2)_n$ returns 2 except from the boundaries, where edge effects are observed. This results in the fact that in cases when statistics $\hat{k}_{low}(1, 2)_n$ can be computed without stumbling on numerical errors performances of `GenLowEstim` and low frequency `ContinEstim` are the same. At the same time, statistic $\hat{\alpha}_{low}(k, t_1, t_2)_n$ is not far from its limit value for $k < 3$, that's why the parameter estimation of the LFSM is technically possible by `ContinEstim` and `GenLowEstim` at such length of the sample path.

When α is near 1 there is a transition between the regime with values of $\hat{k}_{low}(1, 2)_n$ concentrated at point $k = 2$, and the regime where $\hat{k}_{low}(1, 2)_n$ is highly dispersed. This shift is characterized by only two values of $\hat{k}_{low}(1, 2)_n$: 2 and 3. Such behavior of the estimated order of increments is due to the fact that when $\alpha^{-1} \in N$

$$\mathbb{P}\left(\hat{k}_{low} = 2 + \alpha^{-1}\right) \rightarrow \lambda \quad \text{and} \quad \mathbb{P}\left(\hat{k}_{low} = 1 + \alpha^{-1}\right) \rightarrow 1 - \lambda$$



(a) Comparison of success rates for ContinEstim and GenLowEstim. Low frequency case. Path length N=200, number of sample paths NmonteC=300.

(b) Comparison of success rates for ContinEstim and GenHighEstim. High frequency case. Path length N=200, number of sample paths NmonteC=300.

Figure 11: Comparison of success rates of estimators

for some constant $\lambda \in (0, 1)$, see Mazur et al. (2020), Section 4.1. Surprisingly, λ is close to 0.5 throughout the whole set of H 's (Figure 12). There are no $\hat{k}_{low}(1, 2)_n$ higher than 3 observed because the preliminary estimation of α is still quite precise as one can see from the middle row on Figure 14. After obtaining $\hat{k}_{low}(1, 2)_n$ equal to either 2 or 3, $\hat{\alpha}_{low}(\hat{k}_{low}(1, 2)_n, t_1, t_2)_n$ is computed again quite precisely, but worse than in the continuous case.

At $\alpha < 1$ $\hat{\alpha}_{low}(k, t_1, t_2)_n$ has high variance regardless of what k is chosen, therefore different values are obtained when computing $\hat{k}_{low}(1, 2)_n$. These values plugged-in to $\hat{\alpha}_{low}(k, t_1, t_2)_n$ produce again very dispersed estimates of parameter α . This mechanism explains why $\hat{\alpha}_{low}$ has higher variance in discontinuous case ($H - 1/\alpha < 0$) than in continuous (see the numerical study in Section 5 in Mazur et al. (2020)).

The way $\hat{\alpha}_{low}$ behaves could be explained using pic.(13), where φ_n and $V_{low}(\psi_t, k)_n$ are plotted. Cases wherein $\hat{\alpha}_{low}$ performs poorly coincide with ones wherein φ_n and $V_{low}(\psi_t, k)_n$ are significantly distant from each other, so convergence $V_{low}(\psi_t, k)_n \xrightarrow{a.s.} \varphi_n(t; k)$ isn't observed at the given length of sample paths, which ruins the whole idea of (σ, α) estimation. Of course, this effect doesn't affect H -estimation because it is based on ratio statistic, which has a different form.

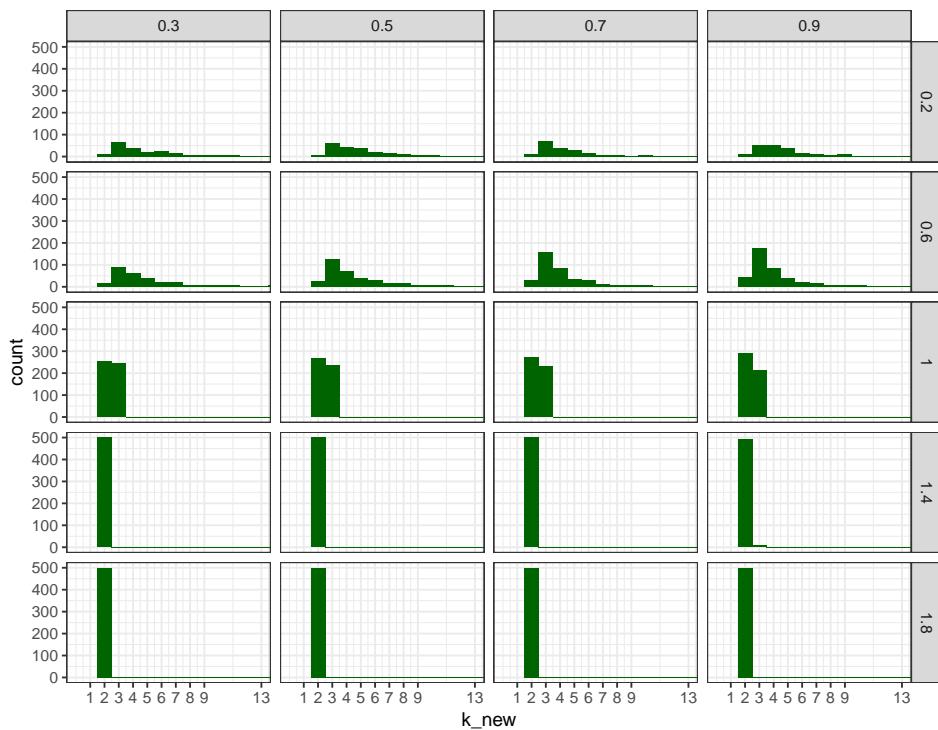


Figure 12: Histograms of preliminary estimations of k , $\hat{k}_{low}(1, 2)_n$. α 's are on vertical labels, H 's- on horizontal.

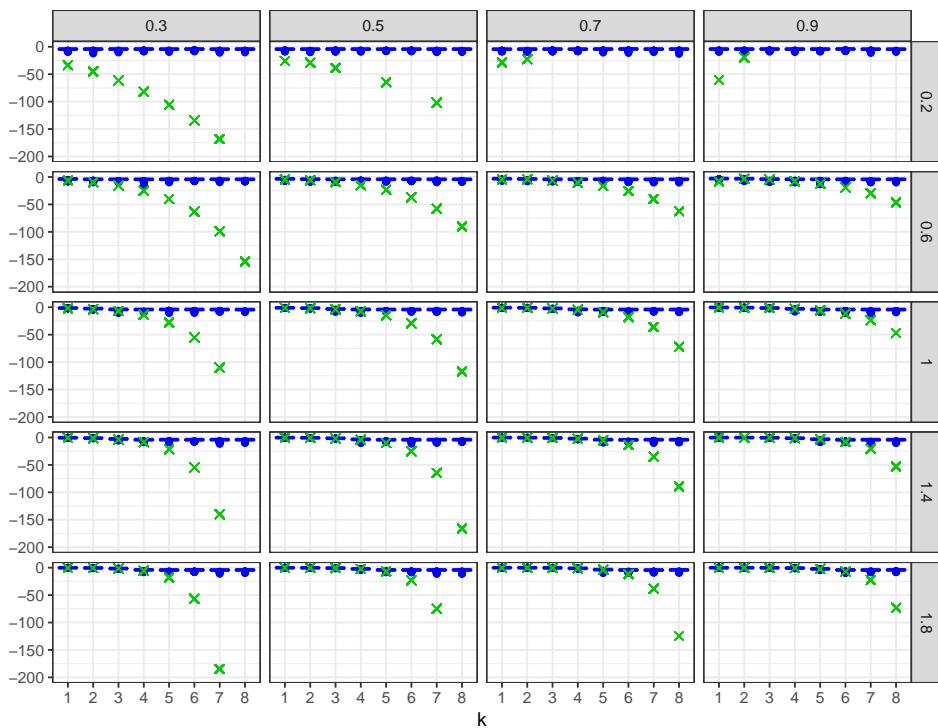


Figure 13: Comparison of the real $\varphi_n(t = 1; k)$ and the one estimated via $V_{low}(\psi_{t=1}, k)_n$ on the logarithmic scale. α 's are on vertical labels, H 's- on horizontal. The lower and upper box sides correspond to the 25th and 75th percentiles.

S4 classes for Lèvy-driven motions

Here we describe a simple S4 system (a short introduction to S4 classes is given in Wickham (2014), Chapter OO field guide) that could be used to simplify manipulations with the two types

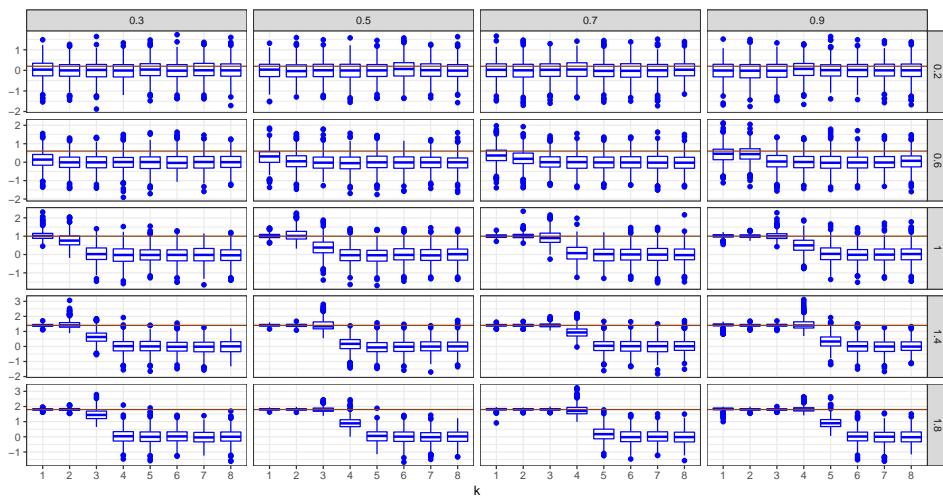


Figure 14: Convergence of $\hat{\alpha}_{low}(k, t_1, t_2)_n$ to the real α (red line) for different k . α 's are on vertical labels, H 's- on horizontal. The lower and upper box sides correspond to the 25th and 75th percentiles.

of observations of the linear fractional stable motion. Additionally, we present a possible way to extend the system so that it encompasses more general stochastic processes. The system aims to be helpful in

- passing “attributes” (frequency, σ , α , H) from objects to functions automatically (without additional developer's efforts).
- hiding complicated details of interfaces from users.
- using generics to protract functions on different objects by means of inheritance. For instance, plotting function written for l fsm could be used for other types of stochastic integral.

Classes for simulated l fsm

Here we describe the least general classes- “SimulatedL fsmLow” and “SimulatedL fsmHigh”, objects of which are obtained by simulating low- and high-frequency linear fractional stable motions. Figure

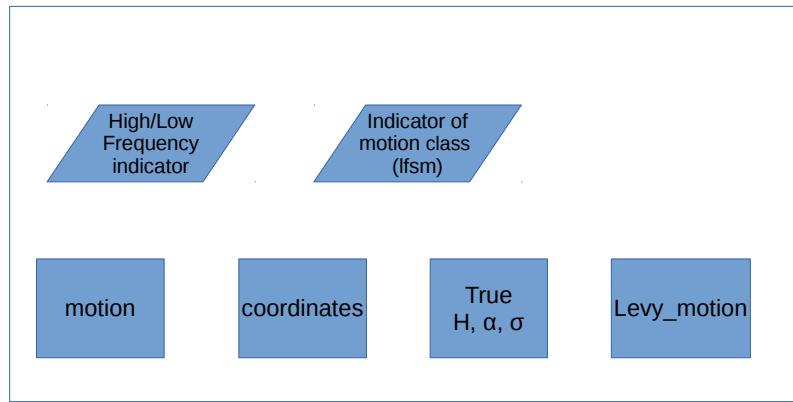


Figure 15: Structure of the classes of simulated l fsm. Frequency indicator and indicator of process type are included in the class name, whilst motion, coordinates, parameters for which the path was simulated and the Lévy motion are written in the slots.

15 shows their internal structure. Roughly speaking, these classes were designed to contain minimum information that could fully describe a simulated LFSM path. Indicators of frequency and a process type are included in the name of a class, which is supposed to make a method dispatch more straightforward, without additional condition blocks. Moreover, all generic functions distinct high- and low-frequency schemes of all types with the help of class names. The same holds for motion

types. Parameters H, α, σ , as well as Lévy motion, coordinates and the lfsm itself are written in corresponding slots.

Examples

In the following example we see how an instance of class “SimulatedLfsmLow” is created and then plotting and inference is performed using generic functions `plot` and `ContinInfer`. First, we register classes, methods and one generic from “S4 classes examples” in the supplementary materials.

```
N<-3000; m<-65; M<-300
sigma<-0.3; alpha<-1.8; H<-0.8
p<-.4; t1<-1; t2<-2; k<-2

# Make an object of S4 class SimulatedLfsmLow
List <- path(N,m,M,alpha,H,sigma,freq='L',disable_X=FALSE,seed=3)

# Make an object of parameters
prmts<-new("AlpaHSigma",alpha=List$pars[['alpha']],
H=List$pars[['H']],sigma=List$pars[['sigma']])
X_sim <- new("SimulatedLfsmLow", Process = List$lfsm,
coordinates = List$coordinates, pars = prmts,
levy_motion = List$levy_motion)

# structure of the instance
str(X_sim)

Formal class 'SimulatedLfsmLow' [package ".GlobalEnv"] with 4 slots
..@ pars      :Formal class 'AlpaHSigma' [package ".GlobalEnv"] with 3 slots
... . . . @ alpha: num 1.8
... . . . @ H   : num 1.8
... . . . @ sigma: num 1.8
..@ levy_motion: num [1:3497] 0 -15 -19.8 -21.2 -24.1 ...
..@ Process    : num [1:3497] 0 -0.542 -0.912 -1.12 -1.276 ...
..@ coordinates: int [1:3497] 0 1 2 3 4 5 6 7 8 9 ...

# plot the motion
plot(X_sim)
```

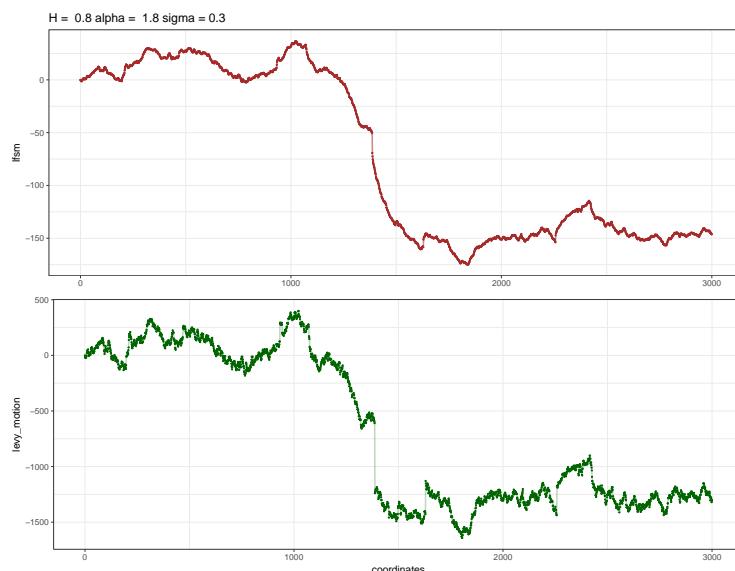


Figure 16: Output of plot method for simulated lfsm

```
ContinInfer(x=X_sim,t1=t1,t2=t2,k=k,p=p)
```

```
$alpha
```

```
[1] 1.870217
$H
[1] 0.8314528
$sigma
[1] 0.3227219
```

In this example, the plot function takes almost no effort, compared to the similar one from Section 2.2, which is due to the fact, that there has been a method defined for generic `plot` and object “SimulatedL fsmLow”. The last function, `Cont inInfer`, is a generic which has a registered method for class “StochasticProcLow”, general stochastic processes in low-frequency setting. Since “SimulatedL fsmLow” inherits from “StochasticProcLow”, the generic dispatched this method and performed statistical inference. `Cont inInfer` was designed to perform inference according to Theorem 3.1 from (Mazur et al., 2020) and is based on R function `Cont inInfer`. One can see that `plot` (and, less obviously, `Cont inInfer`) used “Low” from the name of the class to perform computations.

Acknowledgments

The authors acknowledge financial support from the project “Ambit fields: probabilistic properties and statistical inference” funded by Villum Fonden. Stepan Mazur acknowledges financial support from the internal research grants at Örebro University, and from the project “Models for macro and financial economics after the financial crisis” (Dnr: P18-0201) funded by Jan Wallander and Tom Hedelius Foundation. The authors would like to thank Prof. Mark Podolskij for significant discussions. Dmitry Otryakhin thanks Dr. Firuza Mamedova for valuable remarks on the draft of this paper.

Bibliography

- A. Basse-O’Connor, R. Lachièze-Rey, and M. Podolskij. Power variation for a class of stationary increments Lévy driven moving averages. *Annals of Probability*, 45(6B):4477–4528, 2017. URL <https://doi.org/10.1214/16-AOP1170>. [p]
- H. Biermé and H.-P. Scheffler. Fourier series approximation of linear fractional stable motion. *Journal of Fourier Analysis and Applications*, 14(2):180–202, 2008. URL <https://doi.org/10.1007/s00041-008-9011-7>. [p]
- J.-F. Coeurjolly. *dvfBm: Discrete variations of a fractional Brownian motion*, 2009. URL <https://CRAN.R-project.org/package=dvfBm>. R package version 1.0. [p]
- T. Dang and J. Istas. Estimation of the Hurst and the stability indices of a H -self-similar stable process. *Electronic Journal of Statistics*, 11(2):4103–4150, 2017. URL <https://doi.org/10.1214/17-EJS1357>. [p]
- J. Huang. *somebm: some Brownian motions simulation functions*, 2013. URL <https://CRAN.R-project.org/package=somebm>. R package version 0.1. [p]
- S. Mazur, D. Otryakhin, and M. Podolskij. Estimation of the linear fractional stable motion. *Bernoulli*, 26(1):226–252, 2020. URL <https://doi.org/10.3150/19-BEJ1124>. [p]
- G. Samorodnitsky and M. S. Taqqu. *Stable non-Gaussian random processes: stochastic models with infinite variance*, volume 1. CRC Press, 1994. [p]
- S. Stoev and M. Taqqu. Simulation methods for linear fractional stable motion and FARIMA using the fast Fourier transform. *Fractals*, 95(1):95–121, 2004. URL <https://doi.org/10.1142/S0218348X04002379>. [p]
- B. Swihart, J. Lindsey, and P. Lambert. *stable: Probability Functions and Generalized Regression Models for Stable Distributions*, 2017. URL <https://CRAN.R-project.org/package=stable>. R package version 1.1.2. [p]
- N. W. Watkins, D. Credgington, R. Sanchez, and S. C. Chapman. A kinetic equation for linear fractional stable motion with applications to space plasma physics. *ArXiv e-prints*, 2008. URL <https://arxiv.org/pdf/0803.2833.pdf>. [p]

- H. Wickham. *Advanced R*. CRC Press, 2014. URL <https://adv-r.hadley.nz/index.html>. [p]
- H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. ISBN 978-3-319-24277-4. URL <https://ggplot2.tidyverse.org>. [p]
- W. B. Wu, G. Michailidis, and D. Zhang. Simulating sample paths of linear fractional stable motion. *IEEE Transactions on Information Theory*, 50(6):1086–1096, 2004. URL <https://doi.org/10.1109/TIT.2004.828059>. [p]
- D. Wuertz, M. Maechler, and Rmetrics core team members. *stabledist: Stable Distribution Functions*, 2016. URL <https://CRAN.R-project.org/package=stabledist>. R package version 0.7-1. [p]

Stepan Mazur

School of Business, Örebro University

Fakultetsgatan 1, SE-701 82 Örebro

Sweden

ORCID: 0000-0002-1395-9427

stepan.mazur@oru.se

Dmitry Otryakhin¹

Department of Mathematics, Aarhus University

Ny Munkegade 118, DK-8000 Aarhus C

Denmark

ORCID: 0000-0002-4700-7221

d.otryakhin.acad@protonmail.ch

¹some parts of the work were done at the Department of Mathematics of Stockholm University.

The R package NonProbEst for estimation in non-probability surveys

by M. Rueda, R. Ferri-García, L. Castro

Abstract Different inference procedures are proposed in the literature to correct selection bias that might be introduced with non-random sampling mechanisms. The R package [NonProbEst](#) enables the estimation of parameters using some of these techniques to correct selection bias in non-probability surveys. The mean and the total of the target variable are estimated using Propensity Score Adjustment, calibration, statistical matching, model-based, model-assisted and model-calibrated techniques. Confidence intervals can also be obtained for each method. Machine learning algorithms can be used for estimating the propensities or for predicting the unknown values of the target variable for the non-sampled units. Variance of a given estimator is performed by two different Leave-One-Out jackknife procedures. The functionality of the package is illustrated with example data sets.

Introduction

Since sampling theory was formalized in the beginning of the 20th century, surveys have been the main tool to obtain information from society and nature. Traditional surveys used telephone or face-to-face interviews for questionnaire administration, as well as mailing lists. However, the increase of costs, linked to the decrease in response rates, and the development of information and communication technologies have favored the use of new survey modes such as online or smartphone questionnaires. These modes make the sampling process cheaper and faster, but tend to amplify bias from several sources. More precisely, online surveys are often performed through a non-probability sampling, using self-selection procedures without a defined sampling frame where the inclusion probabilities are known or with deficient sampling frames with coverage issues, leading to higher levels of selection bias ([Elliott and Valliant, 2017](#)).

Some techniques can be used to correct selection bias in online non-probability surveys. A good overview of the various methods is given in [Elliott and Valliant \(2017\)](#). There are three important approaches: the pseudo-design based inference (or pseudo-randomisation ([Buelens et al., 2018](#))), statistical matching and predictive inference.

In the pseudo-design based inference, the idea is to construct weights to correct for selection bias. The first method is estimating response probabilities and using them in Horvitz-Thompson or Hajek type estimators to account for unequal selection probabilities. The most used method to estimate response probabilities is Propensity Score Adjustment (see e.g. [Lee and Valliant \(2009\)](#)). This method uses a probability reference sample in addition to a non-probability convenience sample to construct a response propensity model. Sample matching is another approach also applied to tackle selection bias. A predictive model, with the target variable as the dependent variable, is built using data from the non-probability sample. This model is subsequently applied to a probability sample (where the target variable is not measured) to predict values of its individuals for an estimation of the population values. Similarly, predictive methods are based on superpopulation models. In this approach, a predictive model is fitted for the analysis variable from the sample and used to project the sample to the full population. This approach (that can be used with probability and non-probability samples) allows researchers to use the auxiliary information about covariates in different methods for predicting the unknown values. Most of these methods require special software for their implementation. The package [NonProbEst](#) implements some of these techniques.

The paper is structured as follows. First, we introduce the notation used throughout the paper and we discuss the different ways to do inference for non-probability surveys. In section 3 we briefly comment on the usefulness of Machine Learning (ML) Techniques in this context. Then, we describe the R package [NonProbEst](#). In section 5 we briefly describe the use of the functions, including suitable examples, for each method.

Statistical methodology

Let U denote a finite population with N units, $U = \{1, \dots, k, \dots, N\}$. Let s_V be a volunteer non-probability sample of size n_V , self-selected from an online population U_V which is a subset of the total target population U . Let y be the variable of interest in the survey estimation. Without any auxiliary information, the population total of y , Y , is usually estimated with the following

Horvitz-Thompson type estimator:

$$\hat{Y}_{HT} = \sum_{k \in s_V} w_{vk} y_k \quad (1)$$

being w_{vk} a weight of the unit k set by the researcher to adjust the lack of response, lack of coverage, voluntariness, ... (e.g. by means of post-stratification). A simple choice is $w_{vk} = N/n_V$, that is, consider the sample of volunteers as if it was obtained with a simple random sampling design of the population U .

This estimator has a bias induced by various mechanisms regarding their application. The most important are the selection bias (due to the difference between sampled and nonsampled individuals on the probability to participate in a survey) and the coverage bias (the online population U_v is not the same of the target population U).

The key to successful weighting to remove the bias in non-probability surveys lies in the use of powerful auxiliary information. Auxiliary information can be available in different forms. We distinguish three different cases, called InfoTP, InfoES and InfoEP, depending on the information at hand.

- InfoTP: Only the population totals of the auxiliary variables are known (often called control totals). Possible sources of information are a census of the target population, an administrative register, ... One of the simplest and most frequently used control totals occurs when the information consists of known counts for a set of population groups.
- InfoES: The auxiliary variable values are available for every element in a probability sample. This reference survey is conducted on the same target population than the non-probability survey, with the main difference that the former has a better coverage and higher response rates than the latter, thus it is adequate to represent the behavior that the target population should have when a probability survey is performed on it.
- InfoEP: The auxiliary variable values are available for every element in the whole population. An example of this is when statistical agencies use auxiliary variables specified in different existing registers, for all the elements in the population.

We will now explain the main methods used to treat these biases depending on the type of information that is available.

InfoTP

Calibration

Let \mathbf{x}_k be the value taken on unit k by a vector of auxiliary variables which population total is assumed to be known $\mathbf{X} = \sum_{k=1}^N \mathbf{x}_k$. The calibration estimation of Y consists in the computation of a new vector of weights w_k for $k \in s$ which modifies as little as possible the original sample weights, w_{vk} , which have the desirable property of producing unbiased estimations, respecting at the same time the calibration equations

$$\sum_{k \in s_V} w_k \mathbf{x}_k = \mathbf{X}. \quad (2)$$

Given a **pseudo-distance** $G(w_k, w_{vk})$, the calibration process consists in finding the solution to the minimization problem

$$\min_{w_k} \left\{ \sum_{k \in s_V} G(w_k, w_{vk}) \right\} \quad (3)$$

while respecting the calibration equation (2). Several distances were defined in Deville and Särndal (1992), being the linear distance one of the most commonly used. The resulting estimator of Y under the chi-square distance is the general regression estimator

$$Y_{reg} = \sum_{s_V} w_k y_k = \sum_{s_V} d_k y_k + (\mathbf{X} - \sum_{s_V} w_{vk} \mathbf{x}_k)' \hat{B}_{s_V} \quad (4)$$

where \hat{B}_s is

$$\hat{B}_{s_V} = T_s^{-1} \sum_{s_V} w_{vk} \mathbf{x}_k y_k \quad (5)$$

being $T_s = \sum_{s_V} w_{vk} \mathbf{x}_k \mathbf{x}'_k$.

It is proved in [Bethlehem \(2010\)](#) that bias can be reduced through calibration only when the non-response due to volunteering has a Missing At Random scheme, while it cannot be equally done in Not Missing at Random situations (which are the most frequent).

InfoSP

Propensity Score Adjustment

The Propensity Score Adjustment method was originally developed by [Rosenbaum and Rubin \(1983\)](#) which sought to reduce the confounding bias between treatment and control groups in experimental designs. This approach would be considered in sampling research as well in combination with a reference sample ([Rubin, 1986](#)), but it was not proposed for online surveys until the early 2000's ([Taylor et al., 2001](#)).

It is expected that a sample collected by online recruitment would not follow the principles of a probability sampling, especially in those cases that the survey is filled by volunteer respondents. In such a situation, every individual is associated to a probability of participating in the survey which depends on [her or his](#) characteristics.

The propensity for an individual to take part on the non-probability survey is obtained by training a predictive model (often a logistic regression) on the dichotomous variable, I_{sv} , which measures whether a respondent from the combination of both samples took part in the volunteer survey or in the reference survey. Covariates used in the model, \mathbf{x} , are measured in both samples (in contrast to the target variable which is only measured in the non-probability sample), thus the formula to compute the propensity of taking part in the volunteer survey with a logistic model, π , can be displayed as

$$\pi(\mathbf{x}) = \frac{1}{e^{-(\gamma^T \mathbf{x})} + 1} \quad (6)$$

for some vector γ , as a function of the model covariates.

We denote by s_R the reference sample and w_{Rk} the original design weight of the k individual in the reference sample

Several options for using the propensity scores in estimation are listed below:

- We can use the inverse of the estimated response propensity as a weight for constructing the estimator ([Valliant, 2019](#)):

$$\hat{Y}_{PSA1} = \sum_{k \in s_V} w_{Vk} y_k / \hat{\pi}(\mathbf{x}_k) = \sum_{k \in s_V} y_k w_k^{PSA1} \quad (7)$$

where $\hat{\pi}(\mathbf{x}_k)$ is the estimated response propensity for the individual k of the volunteer sample as predicted using covariates \mathbf{x} .

- Alternatively, the approach proposed in [Schonlau and Couper \(2017\)](#) can be used to obtain weights for a [Horvitz-Thompson](#) type estimator using propensity scores. Weights are defined as

$$w_k^{PSA2} = \frac{1 - \hat{\pi}(\mathbf{x}_k)}{\hat{\pi}(\mathbf{x}_k)} \quad (8)$$

and resulting estimator for the population total is given by

$$\hat{Y}_{PSA2} = \sum_{k \in s_V} y_k w_k^{PSA2} \quad (9)$$

- [Valliant and Dever \(2011\)](#) use the propensity scores to post-stratify the sample. The process is: sort the combined sample by $\hat{\pi}(\mathbf{x}_k)$; split the combined sample into g classes ($g = 5$ as the conventional choice following [Cochran \(1968\)](#)), each of which has about the same number of cases in the combined sample; and compute an average propensity, $\bar{\pi}_g$ within subclass g . Use $\bar{\pi}_g$ as the weight adjustment for every person in the subclass. Resulting estimator is:

$$\hat{Y}_{PSA3} = \sum_g \sum_{k \in s_{Vg}} w_{Vg} y_k / \bar{\pi}_g = \sum_g \sum_{k \in s_{Vg}} y_k w_k^{PSA3} \quad (10)$$

- Following the approach described in [Lee and Valliant \(2009\)](#) propensity scores are divided in g classes, where all units may have the same propensity score or at least be in a very narrow

range and an adjustment factor is calculated as:

$$f_g = \frac{\sum_{k \in s_R g} w_{Rk} / \sum_{k \in s_R} w_{Rk}}{\sum_{k \in s_V g} w_{Vk} / \sum_{k \in s_V} w_{Vk}} \quad (11)$$

where s_{Rg} is the set of individuals in the reference sample that are in the g th class of propensity scores and s_{Vg} is the set of individuals in the volunteer sample that are in the g th class of propensity scores. Finally, the adjusted weights w_k^{PSA4} are the product of the original weights and the adjustment factor; following the same notation, the adjusted weight for individual k in s_{Vg} (i. e. the individual k of the g th propensity class in the volunteer sample) is computed as

$$w_k^{PSA4} = w_{Vk} f_g \quad (12)$$

and the estimator is given by

$$\hat{Y}_{PSA4} = \sum_g \sum_{k \in s_{Vg}} y_k w_k^{PSA4} \quad (13)$$

Research findings have shown that PSA successfully removes bias in some situations, but at the cost of increasing the variance (Lee and Valliant, 2009). Valliant and Dever (2011) showed that the estimation of a variable using PSA must be complemented with further weighting adjustment in order to make estimates less biased. The use of PSA with further calibration is studied in Lee and Valliant (2009) and Ferri-García and Rueda (2018), concluding that calibration adjustments are helpful if they are applied using the right covariates.

Variance estimation in PSA is not a simple issue. Valliant (2019) proposes an estimator of the variance for an estimator of a mean, \hat{y} , based on linearization, but this estimator does not take into account the randomness of weight estimation, therefore it will tend to underestimate the variance.

Jackknife's variance estimator (Quenouille (1956)) can be seen as an acceptable alternative in nonprobability samples after applying PSA. Let $\hat{y} = \frac{1}{N} \sum_{k \in s_V} w_k^{PSA} y_k$ be the estimator of the mean of y , his Leave-One-Out Jackknife estimator of the variance is given by:

$$\hat{V}(\hat{y}) = \frac{n-1}{n} \sum_{j=1}^n (\bar{y}_{(j)} - \bar{y})^2 \quad (14)$$

where $\bar{y}_{(j)}$ is the value of the estimator \hat{y} after dropping unit j from s_V and where \bar{y} is the mean of values $\bar{y}_{(j)}$.

Given that PSA weights are estimated from the available data, the exclusion of one unit can have an impact on the values of w_i and affect the variability of the estimator. This variability can be taken into account if propensities are recalculated for each of the n Leave-One-Out partitions. Thus a Jackknife estimator with recalculating weights is defined as:

$$\hat{V}_{rw}(\hat{y}) = \frac{n-1}{n} \sum_{j=1}^n (\bar{y}_{rw(j)} - \bar{y}_{rw})^2 \quad (15)$$

where $\bar{y}_{rw(j)} = \frac{1}{N} \sum_{k \in s_V - \{j\}} w_k^{PSA}(j) y_k$, with $w_k^{PSA}(j)$ the PSA weight obtained from the sample $s_V - \{j\}$ and \bar{y}_{rw} is the mean of values $\bar{y}_{rw(j)}$.

Statistical matching

The statistical matching method was introduced by Rivers (2007). The idea is to model the relationship between y_k and \mathbf{x}_k using the volunteer sample s_V in order to predict y_k for the reference sample. That is, the matching estimator is given by:

$$\hat{Y}_{SM} = \sum_{s_R} \hat{y}_k w_{Rk}$$

being \hat{y}_k the predict value of y_k .

The key is how to predict the values y_k . Usually $\hat{y}_k = \mathbf{x}'_k \hat{\beta}$ being $\hat{\beta} = \sum_{k \in s_V} y_k \mathbf{x}_k / \sum_{k \in s_V} \mathbf{x}'_k \mathbf{x}_k$ but other methods can be considered as donor imputation (Rivers, 2007) or fractional donor imputation (Kim and Fuller, 2004).

A major drawback of matching is that the precision of the non-probability sample reduces

to the standard error of the reference sample (Buelens et al., 2018). These authors also justify that matching is based on strong ignorability assumptions and can lead biased estimators if the assumptions are not met.

InfoUP

The prediction approach is based on superpopulation models, which assume that the population under study $\mathbf{y} = (y_1, \dots, y_N)'$ is a realization of super-population random variables $\mathbf{Y} = (Y_1, \dots, Y_N)'$ having a superpopulation model ξ . To incorporate auxiliary information \mathbf{x}_k available for all $k \in U$ on assume a superpopulation for y built on some mean function of \mathbf{x} :

$$Y_k = m(\mathbf{x}_k) + e_k, \quad k = 1, \dots, N. \quad (16)$$

The random vector $e = (e_1, \dots, e_N)'$ is assumed to have zero mean and a positive definite covariance matrix which is diagonal (Y_k are mutually independent).

Using a set of covariates, \mathbf{x} , measured in s_V and $\bar{s}_V = U - s_V$ it is possible to estimate the values of y in \bar{s}_V with regression modeling such that the estimated value of y for an individual k can be calculated through the following expression:

$$\hat{y}_k = E_m(y_k | \mathbf{x}_k) \quad (17)$$

m alludes to the specific model which provides the expectation of y_k , and \mathbf{x}_k are the values of the k -th individual in the covariates \mathbf{x} .

We can use the auxiliary information in several ways to define several estimators:

- the model-based estimator:

$$\hat{Y}_m = \sum_{k \in s_V} y_k + \sum_{k \in \bar{s}_V} \hat{y}_k \quad (18)$$

- the model-assisted estimator:

$$\hat{Y}_{ma} = \sum_{k \in U} \hat{y}_k + \sum_{k \in s_V} (y_k - \hat{y}_k) w_{V,k} \quad (19)$$

- the model-calibrated estimator:

$$\hat{Y}_{mcal} = \sum_{k \in s_V} y_k w_k^{CAL} \quad (20)$$

where w_k^{CAL} are such that they minimize $\sum_{k \in s} G(w_k^{CAL}, w_{V,k})$, where $G(\cdot, \cdot)$ is a particular distance function, subject to

$$\sum_{k \in s_V} w_k^{CAL} \hat{y}_k = \sum_{k \in U} \hat{y}_k.$$

Usually the linear regression model is used, $E_m(y_k | \mathbf{x}_k) = \mathbf{x}'_k \beta$ and the above estimators can be rewritten as a type of regression estimators.

Prediction estimators need complete information about the auxiliary variables (InfoEP) and can fail if the model is not true, but might potentially be fruitful to correct for selection bias in informative sampling (Buelens et al., 2018).

Use of machine learning algorithms in non-probability samples

The emerging data sources like Big Data can be used in combination to traditional survey samples for construct more valid inferences. Machine Learning (ML) methods can be used for the matter, given their known advantages in high dimensional environments. There are several types of learning algorithms but for this package we focus on classification and regression. Classification aims to identify the category to which a new observation belongs while regression is used for prediction in real-valuated variables. Both are trained with known observations to make predictions based on some covariates.

There is a vast spectrum of classification and regression algorithms to take into account, starting from the basic linear and logistic regressions and its extensions, like Ridge regression (Hoerl and Kennard, 1970). Other examples are decision trees which uses tree-like graphs , like the C4.5

(Quinlan, 1993). More modern approaches even build ensembles of decision trees with outstanding results, like XGBoost (Chen and Guestrin, 2016). During the last few years, deep learning models have been dramatically improving the state-of-the-art (LeCun et al., 2015). However, many other techniques are still being widely used and developed, like some bayesian methods (Park and Casella, 2008). Having so many different options, choosing the right learning algorithm for each problem is key for obtaining optimal results.

Regarding survey research, the use of ML algorithms has been studied in the last few years for deriving model-assisted estimators (Montanari and Ranalli (2007); Baffetta et al. (2009); Breidt et al. (2017)). In the prediction approach ML algorithms uses the sample to train a model capturing the behaviour of a target variable which is to be estimated, and applies it to the nonsampled individuals to obtain population-level estimates. Applications of machine learning algorithms in PSA for nonresponse propensity have been studied for classification and regression trees (Phipps et al., 2012) and Random Forests (Buskirk and Kolenikov, 2015); their efficacy on reducing nonresponse bias in comparison to logistic regression depends on the available covariates and the complexity of the relationships. (Chen et al., 2019) use LASSO for calibrating non-probability surveys. (Buelens et al., 2018) review existing inference methods to correct for selection bias and recommend adding ML methods to deal with non-probability samples.

NonProbEst allows the use of a wide variety of classification and regression algorithms for model-based, model-assisted and model-calibrated estimators, matching and PSA (which only works with classification). It offers so many alternatives by relying on `caret` (Kuhn, 2018), a well known machine learning package.

The R package NonProbEst

The package `NonProbEst` implements in R a set of techniques for estimation in non-probability surveys, using various approaches which correspond to several frameworks. Functions in the package allow to obtain calibration weights via `calib_weights`, propensity scores via `propensities` and matching predictions for a reference sample via `matching`. Propensity scores can be transformed into weights by all of the approaches mentioned in previous sections via functions `lee_weights`, `sc_weights`, `valliant_weights`, `vd_weights`. These weights can be used for estimation of total, mean and proportion of a given target variable measured in a sample using functions `total_estimation`, `mean_estimation`, `prop_estimation`. Alternatively, total and mean can also be calculated using a model-based, a model-assisted or a model-calibrated approach with the functions `model_based`, `model_assisted` and `model_calibrated` respectively. The variance of the estimators can be calculated using the Leave-One-Out Jackknife method, this is, recalculating the set of weights after subtracting one unit or not, by means of the functions `generic_jackknife_variance` and `jackknife_variance`, and without recalculating the weights via `fast_jackknife_variance`. Frequentist confidence intervals of the estimates can be directly computed with the `confidence_interval` function.

Calibration weights are obtained using the `calib` function of the `sampling` package (Tillé and Matei, 2016) for g-weights computation. `calib_weights` offers a wrapper for calculation of final weights straight from the dataset. Functions that require prediction techniques, such as `propensities`, `matching`, `model_based`, `model_assisted`, `model_calibrated` and `jackknife_variance`, use the `train` function from the `caret` package (Kuhn, 2018). This function allows the user to use any of the algorithms in the large list of functions which are covered by `train`, with the possibility of optimizing hyperparameters for a better performance of the predictors. For propensity estimation, only classification algorithms should be used as the target variable is binary (participation in the probability sample vs participation in the non-probability sample). Case weights are used to balance both classes (for models that accept them). For matching, model-based and model-assisted estimations, algorithms should account for the type of variable of the target feature.

Note that weighting formulas for PSA from Lee (2006) and Valliant and Dever (2011) require applying a stratification procedure. In both `lee_weights` and `vd_weights` the same procedure is applied: the vector of propensities is sorted increasingly, and the individuals are equally divided in g strata of the same length according to their position in the sorted vector. g is defined by the user, and the procedure results in a vector with the strata number (from 1 to g) to which a given individual corresponds. This stratification avoids errors that could arise from the lack of unique values.

Three datasets are available in the package: `sampleP`, `sampleNP` and `population`. These fictitious datasets were created as described in Ferri-García and Rueda (2018); `sampleP` represents a probability sample of size $n_r = 500$ extracted by simple random sampling from a frame covering the entire population, while `sampleNP` represents a non-probability sample of size $n_v = 1000$ ex-

tracted by simple random sampling from a frame covering only the subpopulation of individuals who have access to Internet. The dataset of the complete population of size $N = 50000$ is available in `population`. Variables available in each dataset differ, with `sampleNP` having the largest amount of variables. In the aforementioned dataset, three variables (`vote_gen`, `vote_pens`, `vote_pir`) measuring whether an individual would vote to a given party ("gen", "pens" or "pir") in an election or not. Probabilities of voting to party "gen", "pens" or "pir" are higher if the individual is a woman, and elder person and has access to the Internet, respectively. These variables are only measured in `sampleNP`, meaning that adjustment methods have to be applied in order to produce reliable estimates of voting intentions. For the matter, the rest of the available variables in the dataset, which are also included in `sampleP` (except for the language) and `population`, can be used. `education_primaria`, `education_secundaria`, `education_terciaria` are three disjunct variables measuring the education level of the individual (Primary, Secondary or Tertiary Education), while `age` and `sex` measures the numeric age and the gender (0 female, 1 male). Finally, `language` measures whether the individual's native language is the official language or not. The absence of certain variables in the datasets accounts for real situations where not all the information is available at individual level.

It must be mentioned that the use of `jackknife_variance` for calculating the variance of the estimators via Leave-One-Out Jackknife will be computationally slower than the `fast_jackknife_variance` alternative. Recalculating the weights in each iteration means that the weighting procedure has to be repeated as many times as individuals are in the non-probability sample. If Propensity Score Adjustment is used for weighting, the models have to be rebuilt in each iteration, resulting in larger computation times which will depend on the computational costs of the algorithms used for propensity estimation. Note that `generic_jackknife_variance` will behave similarly if the estimator passed as argument involves predictive modelling algorithms or other costly procedures. To show the difference of procedures, we calculated the Leave-One-Out Jackknife estimated variance of the estimator of the mean for the variable `vote_pir` in a non-probability sample of size $n_v = 100$ extracted by simple random sampling on the `sampleNP` dataset, using a probability sample of size $n_r = 100$ extracted by simple random sampling on the `sampleP` dataset as the reference sample data. Considering a population of $N = 50000$, variance estimates of the estimator weighted by PSA using different algorithms were computed, measuring the computation elapsed time. All the calculations were performed in a Intel(R) Core(TM) i7-3770 CPU up to 3.40GHz. Results can be consulted in Table 1

Weight recalculation	PSA algorithm	R function	Elapsed time (seconds)
No	Logistic regression	glm	0.004999876
Yes	Logistic regression	glm	75.56034
Yes	CART	rpart	102.3409
Yes	Random Forest	rf	203.7737
Yes	GBM	gbm	453.731
Yes	Neural Network	nnet	719.733

Table 1: Total elapsed time of Leave-One-Out Jackknife variance estimation under recalculations of weights in each iteration for a set of predictive models, with sample sizes of 100 for both the probability and the non-probability sample

In this example, the variance estimation with recalculations takes more than 15000 times the seconds that it takes without recalculations if logistic regression is the method used for propensity estimation, and almost 144000 times if feed-forward neural networks are used. Time differences might be different depending on the data, the estimator and the algorithm, but they will be largely appreciable in all cases.

In order to illustrate how the resources in the package can be used for estimation in non-probability surveys, some examples of each adjustment covered by the package are developed in the following section.

Inference in non-probability samples with NonProbEst

InfoTP: Calibration

Suppose that a non-probability sample of 1000 individuals recruited via online surveying is available for estimating the vote intention in a given election. For the matter, `sampleNP` will be used as the non-probability sample data.

```
> library(NonProbEst)
> head(sampleNP)
  vote_gen vote_pens vote_pir education_primaria education_secundaria education_terciaria age sex language
1          0         1         0                  1                  0                  0   66   1     1
2          0         0         1                  0                  0                  1   30   1     1
3          1         0         0                  0                  1                  0   62   0     1
4          0         0         1                  1                  0                  0   33   0     1
5          0         0         1                  0                  1                  0   30   0     1
6          0         0         0                  1                  0                  0   69   1     1
```

Some auxiliary information is available in the sample; more precisely, individual data on education, age, gender and language (as described in the previous Section) can be used for mitigating the effects of coverage error. Population totals are available for all of these auxiliar variables, as they have been measured for the entire population. They can be retrieved from the `population` dataset:

```
> head(population)
  education_primaria education_secundaria education_terciaria age sex language
1                  0                  1                  0   39   1     1
2                  0                  0                  1   55   0     1
3                  1                  0                  0   35   0     1
4                  1                  0                  0   58   1     1
5                  1                  0                  0   36   1     1
6                  0                  1                  0   61   1     1
> totals <- colSums(population)
> totals
  education_primaria education_secundaria education_terciaria      age      sex      language
25287           10546           14167 2539340 244430 45429
```

If the variables of which population totals are available are not disjunct, Raking calibration can be applied in order to estimate cell counts and account for the lack of information. This can be done with the `calib_weights` function; in this case, the `Xs` argument were the dataset `sampleNP` selecting the auxiliar variables only. Other arguments involve the totals previously obtained and the initial weights, which allows the user to specify whether sampling design weights were used or not. In the latter case, unitary weights should be provided as a vector of ones of length equal to the number of individuals in the non-probability sample. Population size and method to be used by the `calib` function from `sampling` have to be specified.

```
> covariates <- colnames(sampleNP)[4:9]
> initial_weights <- rep(1, nrow(sampleNP))
> w <- calib_weights(sampleNP[, covariates], totals, initial_weights,
  N = 50000, method = "raking")
```

Once we obtain the weights, estimates for the mean (proportion if the variable is binary) or the total of any variable present in the non-probability sample can be obtained using `mean_estimation` or `total_estimation` respectively. For example, the estimated proportion of votes for each party can be obtained with the following code:

```
> mean_estimation(sampleNP, w, "vote_gen", N = 50000)
  vote_gen
0.09824163
> mean_estimation(sampleNP, w, "vote_pens", N = 50000)
  vote_pens
0.3726149
> mean_estimation(sampleNP, w, "vote_pir", N = 50000)
  vote_pir
0.3905399
```

If these estimates are compared to those which would be obtained if no adjustment was used, the effect of calibration is notorious. As the presence of "gen" voters in the sample is MCAR, estimates do not differ, but in the case of "pens" voters whose presence is MAR, the calibration approach gives a larger estimate which can be explained by the fact that the overrepresentation of younger people in the sample has been corrected up to a point. To a much lesser extent, this correction is also noticeable in the estimation of vote to "pir" (presence of their voters in the sample is NMAR).

```
> sum(sampleNP$vote_gen)/nrow(sampleNP)
[1] 0.096
> sum(sampleNP$vote_pens)/nrow(sampleNP)
[1] 0.346
> sum(sampleNP$vote_pir)/nrow(sampleNP)
[1] 0.404
> sum(sampleNP$vote_gen)/nrow(sampleNP) -
+     mean_estimation(sampleNP, w, "vote_gen", N = 50000)
  vote_gen
```

```

-0.00224163
> sum(sampleNP$vote_pens)/nrow(sampleNP) -
+      mean_estimation(sampleNP, w, "vote_pens", N = 50000)
  vote_pens
-0.02661494
> sum(sampleNP$vote_pir)/nrow(sampleNP) -
+      mean_estimation(sampleNP, w, "vote_pir", N = 50000)
  vote_pir
0.01346014

```

The variance of the estimates can be assessed through Leave-One-Out Jackknife, both with or without reweighting in each iteration. In the former case, a function must be created by the user for such a task. In the following lines, a function example is developed for estimating the variance on the estimation of the proportion of votes for the "pir" party:

```

### Leave-One-Out Jackknife variance estimation with reweighting
> estimator <- function(s){
  initial_weights <- rep(1, nrow(s))
  w <- calib_weights(s[,covariates], totals, initial_weights, N = 50000,
    method = "raking")
  return(mean_estimation(s, w, "vote_pir", N = 50000))
}
> v_r <- generic_jackknife_variance(sampleNP, estimator, N = 50000)
> v_r
[1] 0.0003352199
### Leave-One-Out Jackknife variance estimation without reweighting
> v_nr <- fast_jackknife_variance(sampleNP, w, estimated_vars = "vote_pir", N = 50000)
> v_nr
  vote_pir
0.0003189449

```

These estimates of the variance can be used for the construction of confidence intervals for the estimation of the proportion via `confidence_interval` function. This function requires the point estimator and the standard deviation as arguments, with the option to fix the confidence level. If not specified by the user, the confidence interval is calculated at 95% confidence level.

```

> ic_r <- confidence_interval( mean_estimation(sampleNP, w, "vote_pir", N = 50000),
  sqrt(v_r)
)
> ic_r
lower.vote_pir upper.vote_pir
  0.3546549   0.4264249
> ic_nr <- confidence_interval( mean_estimation(sampleNP, w, "vote_pir", N = 50000),
  sqrt(v_nr)
)
> ic_nr
lower.vote_pir upper.vote_pir
  0.3555368   0.4255429

```

InfoSP: Propensity Score Adjustment

Suppose that, in addition to the non-probability sample, a probability sample of the same target population is available as auxiliary information. The target variable is not measured, but some other variables which are also available in the non-probability sample have been measured on it. For the matter, `sampleP` will be used as data from the probability sample.

```

> head(sampleP)
  education_primaria education_secundaria education_terciaria age sex
1                  1                      0                   0  35   1
2                  0                      0                   1  64   0
3                  1                      0                   0  55   1
4                  0                      1                   0  61   1
5                  0                      0                   1  35   0
6                  1                      0                   0  51   1

```

In order to reduce the selection bias, Propensity Score Adjustment can be used in this case for reweighting. This procedure is implemented in the **propensities** function; it requires both samples, the list of covariates to be used to build the models for propensity estimation, and three arguments regarding technical aspects of the adjustment: the prediction algorithm (must match any of the list of **caret** supported algorithms), a boolean indicating whether smoothing of propensities is applied or not, and a vector of strings specifying the preprocessing procedures to be passed to **train** (by default, preprocessing is not applied). Further arguments to be passed to **train** can be specified.

In this example, the propensity of participating will be estimated using k-Nearest Neighbors with further smoothing and a parameter grid of all the odd numbers between 3 and 11 for optimization of k . The covariates will be all the variables measured in **sampleP**. The result will be a list with two vectors: the estimated propensities for individuals in the non-probability (convenience) and the probability (reference) sample respectively.

```
> covariates <- colnames(sampleP)
> pi <- propensities(sampleNP, sampleP, covariates,
  algorithm = "knn", smooth = T, tuneGrid = data.frame(k = seq(3, 11, by = 2)))
> summary(pi$convenience)
  Min. 1st Qu. Median Mean 3rd Qu. Max.
0.3079 0.6249 0.6873 0.6834 0.7584 0.9995
> summary(pi$reference)
  Min. 1st Qu. Median Mean 3rd Qu. Max.
0.3079 0.5384 0.6388 0.6236 0.6998 0.9469
```

The propensities must be subsequently transformed into weights for their application in survey estimation. Transformations available in **NonProbEst** include approaches developed by Lee (2006) and Lee and Valliant (2009) in the **lee_weights** function, Valliant and Dever (2011) in the **vd_weights** function, Schonlau and Couper (2017) in the **sc_weights** function and Valliant (2019) in the **valliant_weights** function. **lee_weights** and **vd_weights** require propensities of both samples and a number of strata (5 by default), while **sc_weights** and **valliant_weights** only require propensities of the non-probability sample.

For example, if we want to apply propensities via weights developed in Valliant and Dever (2011) for the estimation of voting intention to party "pir", we can do it with the following code:

```
> wi <- vd_weights(convenience_propensities = pi$convenience,
  reference_propensities = pi$reference)
> summary(wi)
  Min. 1st Qu. Median Mean 3rd Qu. Max.
1.233 1.376 1.493 1.505 1.632 2.011
> mean_estimation(sample = sampleNP, weights = wi,
  estimated_vars = "vote_pir")
  vote_pir
0.4006072
#Estimation of the 95% confidence interval
> estim <- mean_estimation(sample = sampleNP, weights = wi,
  estimated_vars = "vote_pir")
> std_dev <- fast_jackknife_variance(sample = sampleNP, weights = wi,
  estimated_vars = "vote_pir", N = 50000)
> confidence_interval(estimation = estim, std_dev = std_dev, confidence = 0.95)
lower.vote_pir upper.vote_pir
0.4001341 0.4010803
```

Note that for those weights that are calculated by means of propensity stratification, propensities of the individuals in the convenience and reference sample are needed. If they are calculated by inverting propensities, only those for the individuals in the convenience sample are needed. For example, if we calculate weights via the formula developed in Schonlau and Couper (2017), the code is:

```
> wi <- sc_weights(propensities = pi$convenience)
> summary(wi)
  Min. 1st Qu. Median Mean 3rd Qu. Max.
0.0004998 0.3185741 0.4549419 0.5044062 0.6003197 2.2479720
```

Apart from direct estimation, resulting weights can be used as inputs in the **initial_weights** argument of the **calib_weights** function for the estimation with PSA and calibration, or with the package **survey** (Lumley, 2018) for more complex analysis.

InfoUP: superpopulation estimators

In this case, in addition to the non-probability sample, the population itself is available for some covariates. However, the target variable is only measured in the non-probability sample. For the matter, `sampleNP` will be used as the non-probability sample data and `population` will be used as the population data.

The model-based estimator can be used to estimate the population total (or mean) for the target variable. In this example, the expected number of votes for "pens" will be estimated with [regularized logistic regression](#) as learning algorithm. This procedure is implemented in the `model_based` function. It requires the sample, the population, the covariates names and the target variable as arguments. In our example, the specific algorithm and a normalization preprocessing are passed to change default behaviour. Since no optimization strategy is specified in this case, a default bootstrap will be applied.

```
> covariates <- c("education_primaria", "education_secundaria",
+                  "education_terciaria", "age", "sex", "language")
> mySample = sampleNP
> mySample$vote_pens = factor(mySample$vote_pens, c(0, 1), c('F', 'T'))
> model_based(mySample, population, covariates, "vote_pens",
+              positive_label = 'T', algorithm = "glmnet", proc = c("center", "scale"))
[1] 18282.51
```

If the proportion of votes has to be estimated, rather than the total, it would be as simple as adding the `estimate_mean` argument as follows:

```
> model_based(mySample, population, covariates, "vote_pens", positive_label = 'T',
+              algorithm = "glmnet", proc = c("center", "scale"), estimate_mean = TRUE)
[1] 0.366757
```

Alternatively, model-calibrated estimator can be used to achieve higher efficiency in some situations. In that case, design weights have to be specified in the argument "weights", in addition to the rest of arguments previously described. If no sampling design was followed in data collection, which is the case that we suppose in our example, we can specify unitary weights by turning the parameter to 1, as it is done in the following code:

```
> model_calibrated(sample_data = mySample, weights = 1, full_data = population,
+                   covariates = covariates, estimated_var = "vote_pens", positive_label = 'T',
+                   algorithm = "glmnet", proc = c("center", "scale"),
+                   estimate_mean = TRUE)
[1] 0.365945
```

Conclusion and future developments

In this paper we show how the [NonProbEst](#) package can simplify the application of different weighting methods to correct selection bias in non-probability surveys. This package is, to the best of our knowledge, the first package that supports the user beyond estimation in PSA, PSA+calibration, statistical matching or model-calibration. Another important feature is that a wide range of ML techniques can be used to optimize the information provided by the auxiliary variables.

Additional features will be integrated in future versions of the package. Some simplified wrappers will be developed for some methods so non-expert users can also easily apply them, more parameters will be available for estimation and further support for weighted models will be added. Also, other techniques for variance estimation can be considered. Many of these features can already be applied combining [NonProbEst](#) with the [survey](#) package, as noted before.

Regarding Machine Learning, methods for variable selection will be studied as well as the use of more advanced deep learning libraries outside of [caret](#)'s scope. Variable selection would help explaining the bias and choosing the best covariates for its correction. Better deep learning libraries would allow the use of state-of-the-art algorithms.

Acknowledgments

This work is partially supported by Ministerio de Economía y Competitividad of Spain (grant MTM2015-63609-R) and by Ministerio de Ciencia, Innovación y Universidades (grant FPU17/02177).

Bibliography

- F. Baffetta, L. Fattorini, S. Franceschi, and P. Corona. Design-based approach to k-nearest neighbours technique for coupling field and remotely sensed data in forest surveys. *Remote Sensing of Environment*, 113(3):463–475, 2009. [p]
- J. Bethlehem. Selection bias in web surveys. *International Statistical Review*, 78(2):161–188, 2010. [p]
- F. J. Breidt, J. D. Opsomer, et al. Model-assisted survey estimation with modern prediction techniques. *Statistical Science*, 32(2):190–205, 2017. [p]
- B. Buelens, J. Burger, and J. A. van den Brakel. Comparing inference methods for non-probability samples. *International Statistical Review*, 86(2):322–343, 2018. [p]
- T. D. Buskirk and S. Kolenikov. Finding respondents in the forest: A comparison of logistic regression and random forest models for response propensity weighting and stratification. *Survey Methods: Insights from the Field*, page 17, 2015. [p]
- J. K. T. Chen, R. L. Valliant, and M. R. Elliott. Calibrating non-probability surveys to estimated control totals using lasso, with an application to political polling. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 68(3):657–681, 2019. [p]
- T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 785–794. ACM, 2016. [p]
- W. G. Cochran. The effectiveness of adjustment by subclassification in removing bias in observational studies. *Biometrics*, pages 295–313, 1968. [p]
- J.-C. Deville and C. E. Särndal. Calibration estimators in survey sampling. *Journal of the American Statistical Association*, 87(418):376–382, 1992. [p]
- M. R. Elliott and R. Valliant. Inference for nonprobability samples. *Statistical Science*, 32(2):249–264, 2017. [p]
- R. Ferri-García and M. d. M. Rueda. Efficiency of propensity score adjustment and calibration on the estimation from non-probabilistic online surveys. *SORT-Statistics and Operations Research Transactions*, 1(2):159–182, 2018. [p]
- A. E. Hoerl and R. W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970. [p]
- J. Kim and W. Fuller. Fractional hot deck imputation. *Biometrika*, 91(3):559–578, 2004. [p]
- M. Kuhn. *caret: Classification and Regression Training*, 2018. URL <https://CRAN.R-project.org/package=caret>. R package version 6.0-81. [p]
- Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436–444, 2015. [p]
- S. Lee. Propensity score adjustment as a weighting scheme for volunteer panel web surveys. *Journal of official statistics*, 22(2):329–349, 2006. [p]
- S. Lee and R. Valliant. Estimation for volunteer panel web surveys using propensity score adjustment and calibration adjustment. *Sociological Methods & Research*, 37(3):319–343, 2009. [p]
- T. Lumley. *survey: Analysis of Complex Survey Samples*, 2018. URL <https://CRAN.R-project.org/package=survey>. [p]
- G. E. Montanari and M. G. Ranalli. Multiple and ridge model calibration for sample surveys. In *Proceedings of the Workshop in Calibration and estimation in surveys, Ottawa*, 2007. [p]
- T. Park and G. Casella. The bayesian lasso. *Journal of the American Statistical Association*, 103(482):681–686, 2008. [p]
- P. Phipps, D. Toth, et al. Analyzing establishment nonresponse using an interpretable regression tree model with linked administrative data. *The Annals of Applied Statistics*, 6(2):772–794, 2012. [p]
- M. H. Quenouille. Notes on bias in estimation. *Biometrika*, 43(3/4):353–360, 1956. [p]

- J. R. Quinlan. *C4. 5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., 1993. [p]
- D. Rivers. Sampling for web surveys. *Presented in Joint Statistical Meetings*, 2007. Salt Lake City, UT. [p]
- P. R. Rosenbaum and D. B. Rubin. The central role of the propensity score in observational studies for causal effects. *Biometrika*, 70(1):41–55, 1983. [p]
- D. B. Rubin. Statistical matching using file concatenation with adjusted weights and multiple imputations. *Journal of Business & Economic Statistics*, 4(1):87–94, 1986. [p]
- M. Schonlau and M. P. Couper. Options for conducting web surveys. *Statistical Science*, 32(2):279–292, 2017. [p]
- H. Taylor, J. Bremer, C. Overmeyer, J. W. Siegel, and G. Terhanian. The record of internet-based opinion polls in predicting the results of 72 races in the november 2000 us elections. *International Journal of Market Research*, 43(2):127–135, 2001. [p]
- Y. Tillé and A. Matei. *sampling: Survey Sampling*, 2016. URL <https://CRAN.R-project.org/package=sampling>. R package version 2.8. [p]
- R. Valliant. Comparing alternatives for estimation from nonprobability samples. *Journal of Survey Statistics and Methodology*, 2019. [p]
- R. Valliant and J. A. Dever. Estimating propensity adjustments for volunteer web surveys. *Sociological Methods & Research*, 40(1):105–137, 2011. [p]

María del Mar Rueda
Department of Statistics and Operations Research
University of Granada
Spain
ORCID: 0000-0002-2903-8745
mrueda@ugr.es

Ramón Ferri-García
Department of Statistics and Operations Research
University of Granada
Spain
ORCID: 0000-0002-9655-933X
rferri@ugr.es

Luis Castro
Department of Statistics and Operations Research
University of Granada
Spain
luiscastro193@correo.ugr.es

ProjectManagement: an R Package for Managing Projects

by Juan Carlos Gonçalves-Dosantos¹, Ignacio García-Jurado and Julián Costa

Abstract Project management is an important body of knowledge and practices that comprises the planning, organisation and control of resources to achieve one or more pre-determined objectives. In this paper, we introduce **ProjectManagement**, a new R package that provides the necessary tools to manage projects in a broad sense, and illustrate its use by examples.

Introduction

Project management is an important body of knowledge and practices that comprises the planning, organisation and control of resources to achieve one or more pre-determined objectives. The most commonly used methods for project planning are PERT (Program Evaluation and Review Technique model) and CPM (Critical Path Method). PERT/CPM analyses the tasks involved in completing a project, especially the time needed to complete each task, and computes the minimum time needed to complete the total project. Through the data obtained in the analysis of the project, PERT/CPM identifies the critical activities, which are those for which any disturbance in its duration modifies the minimum time of execution of the project. Also, it obtains the times that can be assigned to non-critical activities, called slacks, in addition to their fixed durations, to give them flexibility. Project management often deals with the problem of redistribution of resources. Sometimes it is convenient to reduce the time of an activity by increasing the assigned costs. Other times, when the availability of resources is limited in a period of time, it may be necessary to level the use of those resources. These situations require a re-planning of the project.

Even with good project management, once the project has been carried out and the actual durations of the activities are known, there can be a delay in the completion time of the project. When the delay generates an additional cost, ways are needed to distribute the cost of the delay among the different tasks involved. To solve this problem we can use cooperative game theory and rules based on bankruptcy problems.

The essential elements related to project management can be found in Castro et al. (2007) or in Hillier and Lieberman (2001). Project management techniques have been widely used in all fields of engineering. Hall (2012) reviews the impact that such techniques have in various fields and their broad business opportunities. Their fields of application vary from classical construction and engineering to information technology and software development, including modern agile methods. Schmitz et al. (2019) also argue the usefulness of traditional project management techniques in the context of agile methodology. Evdokimov et al. (2018) include a case study that shows the current relevance of project management techniques in software development. Özdamar and Ulusoy (1995) present a survey of the problem of resource constraints. To distribute the delay cost of the project among the activities, Brânzei et al. (2002) provide two rules using, respectively, a game theoretical and a bankruptcy-based approach, and Bergantiños et al. (2018) introduce and analyse a consistent rule based on the Shapley value.

A well-known project management software is Microsoft Project. This tool is designed to create and control a project, through the allocation of resources to tasks, the management of budget and workloads, as well as monitoring developments. Microsoft Project is not open source and its license is fee-based. Other project management applications have been created as free software, such as OpenProj, PpcProject or ProMes (Gregoriou et al., 2013). In Salas-Morera et al. (2013) we can see a useful comparison of these applications.

The aforementioned tools are written in Java or Phyton. To the best of our knowledge, there are only two packages in R available for project management. **PlotPrjNetworks** (Muñoz, 2015) and **plan** (Kelley, 2018) are packages that offer the user the creation of a Gantt diagram for the visualization of the project structure. In our opinion, a tool was missing to manage a project from its development to its control. We believe that such a tool would be useful for the user community because it could be integrated with other tools developed in R, it could be easily modified to suit the specific needs of each user, and it could be wrapped into a graphical interface.

In this paper, we introduce **ProjectManagement**¹ (Gonçalves-Dosantos et al., 2020b), a new R package that provides the necessary tools to manage projects in a broad sense. It calculates the critical activities, the slack of each activity, the minimum duration of the project and the early and

¹<https://github.com/Juan-Goncalves-Dosantos/ProjectManagement.git>

last times of each activity. It plots a graph of the project and the schedule. The package also allows cost management to reduce the minimum project time, as well as resource management. Once the actual durations of the activities are known, it is possible to distribute the delay generated in the project among the different activities. When activity durations are considered random variables, the package provides additional functionality. In particular, it calculates the average duration of the project and the criticality index of each activity. It plots a representation of the project duration distribution and the early and last times of the activities. And it calculates several allocation proposals of the delay cost when the project has been completed and the actual duration of the activities is known.

The paper is organized as follows. First, we recall the basic definitions of project management and present different ways to distribute the delay cost when durations are assumed to be known and when they are random variables. Then, we provide a description of **ProjectManagement**. Finally, we illustrate the use of the package by way of examples.

Project management

In this section we discuss the basic concepts of deterministic and stochastic projects with a special focus on allocating the delay cost among the project activities. The aim of this section is to provide a brief (and quick) survey of the methodologies implemented in the R package **ProjectManagement** that we introduce later, as well as to indicate the main bibliographical sources in which interested readers can deepen their knowledge of each of these methodologies.

Let X be a finite non-empty set and N be a set of ordered pairs (x_1, x_2) , with $x_1, x_2 \in X$ and $|N| = n$. A directed graph is a pair $G = (X, N)$, where X is the set of nodes and N is the set of arcs. We say that an arc $i = (x_{i,1}, x_{i,2}) \in N$ starts at node $x_{i,1} \in X$ and ends at $x_{i,2} \in X$. A node $x_s \in X$ is a source node if there is no arc $i \in N$ such that $x_{i,2} = x_s$. A node $x_e \in X$ is a sink node if there is no arc $i \in N$ such that $x_{i,1} = x_e$. A cycle is a set of arcs $i_0, i_1, \dots, i_m \in N$ such that $x_{i_j,2} = x_{i_{j+1},1}$, with $j \in \{0, \dots, m-1\}$, and $x_{i_m,2} = x_{i_0,1}$. To illustrate the concept of directed graph consider the following example. Take graph $G = (X, N)$ given by $X = \{a, b, c, d\}$ and $N = \{1 = (a, b), 2 = (a, c), 3 = (b, d), 4 = (c, d)\}$. The diagram representing this graph is depicted in Figure 1. This graph has one source (a) and one sink (d). Arc 3, for instance, starts at node b and ends at node d . This graph has no cycles. However, if we add an arc $5 = (d, a)$, the resulting graph has two cycles: 1, 3, 5 and 2, 4, 5.

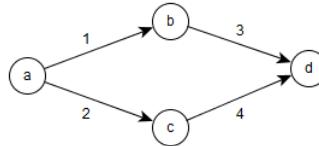


Figure 1: Diagram of the directed graph $G = (X, N)$. The circles represent the nodes and the arrows represent the arcs. This is the standard way of depicting a graph.

A deterministic project P is a tuple $P = (G, x^0)$, where $G = (X, N)$ is a directed graph without cycles, with one source node and one sink node, and $x^0 \in \mathbb{R}_+^n$ is the vector of non-negative planned durations. In this context, N represents the set of activities in the project. We denote by \mathcal{P}^N the family of all deterministic projects with set of activities N , and by \mathcal{P} the family of all deterministic projects.

In a deterministic project $P = (G, x^0) \in \mathcal{P}^N$, you can calculate the minimum duration of P , denoted by $D(G, x^0)$, i.e. the minimum time the project needs to complete all activities taking into account the structure of the graph. This time can be obtained as the solution of a linear programming problem, and thus, can be easily computed. Alternatively, $D(G, x^0)$ can be calculated using a project planning methodology like PERT (see, for instance, Hillier and Lieberman (2001) for details on project planning).

Given a node $x \in X$, we define the set of immediate predecessors of x as the set of activities ending in x , $Pred(x) = \{i \in N / x_{i,2} = x\}$, and the immediate successors of x as $Suc(x) = \{i \in N / x = x_{i,1}\}$. We define the earliest time $D_i^E(G, x^0)$ of an activity $i \in N$ as the minimum time required to complete all immediate predecessor activities of $x_{i,1}$, i.e. the *earliest start time* the activity i can start taking into account the graph

$$D_i^E(G, x^0) = \max_{j \in Pred(x_{i,1})} \{D_j^E(G, x^0) + x_j^0\}.$$

The *latest completion time* $D_i^L(G, x^0)$ of an activity $i \in N$ is the latest point in time when the activity can end without delaying the project

$$D_i^L(G, x^0) = \begin{cases} \max_{j \in N; \text{Suc}(x_{j,2})=\emptyset} \{D_j^E(G, x^0) + x_j^0\} & \text{if } \text{Suc}(x_{i,2}) = \emptyset, \\ \min_{j \in \text{Suc}(x_{i,2})} \{D_j^L(G, x^0) - x_j^0\} & \text{otherwise.} \end{cases}$$

It is easy to see that $D_i^E(G, x^0) \leq D_i^L(G, x^0)$ for all $i \in N$. Also, we can calculate the minimum duration of a project, using the earliest start times, as $D(G, x^0) = \max_{i \in N} \{D_i^E(G, x^0) + x_i^0\}$.

We define the slack $S_i(G, x^0)$ of an activity $i \in N$ as the maximum time, in addition to x_i^0 , that i can use to complete its task without delaying the project

$$S_i(G, x^0) = D_i^L(G, x^0) - D_i^E(G, x^0) - x_i^0.$$

If the slack for an activity is equal to 0, then this activity is critical, i.e. any perturbation in its time modifies the duration of the project. We can also define two other types of slack. The free slack of an activity is the maximum amount of time that this activity can be delayed without causing a delay in the project or in the earliest time of the other activities. The free slack of an activity can be calculated as

$$FS_i(G, x^0) = \min_{j \in \text{Suc}(x_{i,2})} \{D_j^E(G, x^0)\} - D_i^E(G, x^0) - x_i^0.$$

The independent slack of an activity is the maximum time that the activity duration can be increased without affecting the times of others activities

$$IS_i(G, x^0) = \max \left\{ \min_{j \in \text{Suc}(x_{i,2})} \{D_j^E(G, x^0)\} - D_i^L(G, x^0) - x_i^0, 0 \right\}.$$

Given the slack of an activity, we define the *latest start time* as the latest time that an activity can start without delaying the project

$$D_i^{EL}(G, x^0) = D_i^E(G, x^0) + S_i(G, x^0)$$

and the *earliest completion time* as the earliest time in which an activity can end if it starts in its earliest start time

$$D_i^{LE}(G, x^0) = D_i^L(G, x^0) - S_i(G, x^0).$$

Besides the schedule of a project, we can manage the resources allocated to the activities. The minimal cost expediting or MCE method (Kelley, 1961) considers that the duration of some activities can be reduced by increasing the resources allocated to them and thus the implementation costs. An MCE problem is a tuple (P, \bar{x}^0, c, D) , where P is a deterministic project, $\bar{x}^0 \in \mathbb{R}_+^n$ is the vector of minimum durations, that is, for each activity $i \in N$, \bar{x}_i^0 is the minimum duration that the activity can take if the resources allocated to carry it out are increased, $c \in \mathbb{R}^n$ is the vector of unit costs, that is, for each activity $i \in N$, c_i is the cost of accelerating a unit of time the duration of i , and D is the minimum duration of the project we are trying to achieve, with $D < D(G, x^0)$. This problem can be solved as a linear programming problem.

Two other interesting problems that arise from the management of resources are the levelling and the allocation (Hegazy, 1999). These problems take into account that in order for activities to be carried out in the estimated time, a certain level of resources must be used. The problem of levelling of resources is to find a schedule that allows to execute the project in its minimum duration time $D(G, x^0)$ whilst the use of resources is as uniform as possible over time. In the problem of allocation of resources, the level of resources available in each period of time is limited. The aim is to find the minimum duration time and a schedule for the execution of the project taking into account this resource constraint. Given the complex nature of these problems, their exact resolution is computationally demanding. The most common practice is to use heuristic methods to solve them.

Once the project is completed, we can know the actual (observed) duration of the activities and, therefore, whether there has been a delay in the project, that is, whether the actual duration of the project has been different than expected. We define a deterministic project with delays as a tuple $CP = (G, x^0, x, C)$, where (G, x^0) is a deterministic project, $x \in \mathbb{R}_+^n$ is the vector of actual duration of the activities, and $C : \mathbb{R}_+ \rightarrow \mathbb{R}$ is the delay cost function. We assume that C only depends on the duration of the project, it is a non-decreasing function, and $C(D(G, x^0)) = 0$. In practice, the most commonly used functions, for a vector $y \in \mathbb{R}_+^n$, are

$$C(D(G, y)) = D(G, y) - \delta \tag{1}$$

with $\delta \in \mathbb{R}_+$, for example $\delta = D(G, x^0)$.

We denote by \mathcal{CP}^N the family of all deterministic projects with delays with set of activities N , and by \mathcal{CP} the family of all deterministic projects with delays.

In a deterministic project with delays $CP \in \mathcal{CP}^N$, we may need to allocate $C(D(G, x))$ among the activities. This can be useful for several reasons. For example, it can serve as an incentive for those responsible for the activities that have been delayed to be more diligent in similar projects that we may carry out with them in the future; or it can be a mechanism to distribute among those responsible for the activities that have been delayed the financial penalty that the project manager has contractually guaranteed. Brânzei et al. (2002) propose two rules based on bankruptcy problems to address this problem: the Proportional rule and the Truncated Proportional rule. Although they define these rules for the case $x_i \geq x_i^0$, we do not consider this restriction. These rules are only defined when the sum of the individual delays is not zero.

The Proportional rule for deterministic scheduling problems with delays ϕ is defined, for each $i \in N$, by

$$\phi_i = \frac{x_i - x_i^0}{\sum_{j \in N} x_j - x_j^0} \cdot C(D(G, x)).$$

The Truncated Proportional rule for deterministic scheduling problems with delays $\bar{\phi}$ is defined, for each $i \in N$, by

$$\bar{\phi}_i = \frac{\min\{x_i - x_i^0, C(D(G, x))\}}{\sum_{j \in N} \min\{x_j - x_j^0, C(D(G, x))\}} \cdot C(D(G, x)).$$

In Bergantiños et al. (2018), the problem of allocating the delay costs is addressed in the context of cooperative game theory using a Shapley rule. As we illustrate later in an example, the Shapley rule allocates the delay costs in a more sensible way than the proportional rules, at least in some cases. It is much more costly to compute it but, in general, the extra effort is worthwhile. A TU-game is a pair (N, v) where N is a finite non-empty set, and v is a map from 2^N to \mathbb{R} with $v(\emptyset) = 0$. We say that N is the player (activity) set of the game and v is the characteristic function of the game, and we usually identify (N, v) with its characteristic function v . The Shapley value, an allocation rule in cooperative game theory, is a map Φ that associates to each TU-game (N, v) a vector $\Phi(v) \in \mathbb{R}^N$ satisfying $\sum_{i \in N} \Phi_i(v) = v(N)$ and providing a fair allocation of $v(N)$ among the players in N . The explicit formula of the Shapley value for every TU-game (N, v) and every $i \in N$ is given by

$$\Phi_i(v) = \sum_{S \subseteq N \setminus \{i\}} \frac{(|N| - |S| - 1)! |S|!}{|N|!} (v(S \cup \{i\}) - v(S)).$$

Since its introduction by Shapley (1953), the Shapley value has proved to be one of the most important rules in cooperative game theory and to have applications in many practical problems (see, for instance, Moretti and Patroni (2008)).

Bergantiños et al. (2018) define the Shapley rule for deterministic projects with delays Sh as $Sh(CP) = \Phi(v^{CP})$, where for all $CP = (G, x^0, x, C) \in \mathcal{CP}^N$:

- v^{CP} is the TU-game with set of players N given by

$$v^{CP}(S) = C(D(G, (x_S, x_{N \setminus S}^0)))$$

for all $S \subseteq N$, where $(x_S, x_{N \setminus S}^0)$ denotes the vector in \mathbb{R}^N whose i -th component is x_i if $i \in S$ or x_i^0 if $i \in N \setminus S$, and

- $\Phi(v^{CP})$ denotes the proposal of the Shapley value for v^{CP} .

The calculation of the Shapley value has, in general, an exponential complexity. In this context, its exact calculation is impossible in practice, even for a moderate number of activities. As an alternative to exact calculation, Castro et al. (2009) proposed an estimate of the Shapley value in polynomial time using a sampling process. In practical terms, this estimate is a reasonable solution.

Next, we introduce a generalization of the model and the rules described above. It follows the results in Gonçalves-Dosantos et al. (2020a). If instead of x_i^0 , the planned duration of activity $i \in N$, we consider the non-negative random variable X_i^0 describing the duration of i , we can define a stochastic project SP as tuple $SP = (G, X^0)$. Unlike in the deterministic setting, the duration of activities, the duration of the project, as well as the early and last times are now random variables instead of fixed numbers.

A stochastic project with delays is a tuple $SCP = (G, X^0, x, C)$, where (G, X^0) is a stochastic project, x is the vector of actual durations, and $C : \mathbb{R}_+ \rightarrow \mathbb{R}$ is the delay cost function. We assume that C is non-decreasing and $C(D(G, 0)) = 0$, where $0 \in R^n$ is the vector with all components equal to zero. Proportional rules can be extended to stochastic projects with delays in a straightforward way.

The Stochastic Proportional rule for deterministic scheduling problems with delays ϕ is defined, for each $i \in N$, by

$$\phi_i = \frac{x_i - E(X_i^0)}{\sum_{j \in N} x_j - E(X_j^0)} \cdot C(D(G, x)).$$

The Stochastic Truncated Proportional rule for deterministic scheduling problems with delays $\bar{\phi}$ is defined, for each $i \in N$, by

$$\bar{\phi}_i = \frac{\min\{x_i - E(X_i^0), C(D(G, x))\}}{\sum_{j \in N} \min\{x_j - E(X_j^0), C(D(G, x))\}} \cdot C(D(G, x)).$$

Also, we can extend the Shapley rule to the stochastic context. Let us see two extensions of the rule. The Shapley rule for stochastic projects with delays SSh is defined by $SSh(SCP) = \Phi(v^{SCP})$, where

- v^{SCP} is the TU-game with set of players N given by

$$v^{SCP}(S) = E(C(D(G, (x_S, X_{N \setminus S}^0))))$$

for all non-empty $S \subseteq N$,² and

- $\Phi(v^{SCP})$ denotes the proposal of the Shapley value for v^{SCP} .

As an alternative to the previous rule, the Shapley rule in two steps for stochastic projects with delays $SSh2$ is defined by $SSh2(SCP) = \Phi(v_1^{SCP}) + \Phi(v_2^{SCP})$, where

- v_1^{SCP} is the TU-game with set of players N given by

$$v_1^{SCP}(S) = E(C(D(G, (x_S, X_{N \setminus S}^0)))) - E(C(D(G, X^0)))$$

for all $S \subseteq N$,

- v_2^{SCP} is the TU-game with set of players N given by

$$v_2^{SCP}(S) = E(C(D(G, (X_S^0, 0_{N \setminus S}))))$$

for all $S \subseteq N$, where $0 \in R^n$ is the vector with all components equal to zero, and

- $\Phi(v_1^{SCP})$ and $\Phi(v_2^{SCP})$ denote the proposal of the Shapley value for v_1^{SCP} and v_2^{SCP} .

In general, the calculation of v^{SCP} , v_1^{SCP} and v_2^{SCP} is very complex. In our package, we use simulations to approximate these characteristic functions.

ProjectManagement package

ProjectManagement is a new R package that allows the user to address different tasks in project management. The user can obtain the duration of a project and a schedule of activities, and can plot this schedule for a better understanding of the problem. When the actual durations of each activity are observed, the package proposes several allocations of the delay cost, if there was any, among the activities. In the stochastic context, the package estimates the average duration of the project and plots the density functions of the following random variables: duration of the project, and early and last times of the activities. As in the deterministic case, it can make an allocation of the delay cost, if any.

The following dependencies of the package must be taken into account: **triangle** (Carnell, 2019), **plotly** (Sievert, 2020), **igraph** (Csardi and Nepusz, 2006), **kappalab** (Grabisch et al., 2015), **GameTheory** (Cano-Berlanga, 2017) and **lpSolveAPI** (lp_solve et al., 2020). The first one is used for calculations with triangular distributions, the second one to plot interactive graphics, the third one

²As in all TU-games, we define $v^{SCP}(\emptyset) = 0$.

to plot graphs, the next two are related to game-theoretic concepts and the last one to solve linear programming problems.

The functions incorporated in the package can be seen in Table 1. Note that for projects of more than 10 activities, functions `delay.pert` and `delay.stochastic.pert` will approximate the Shapley value through a sampling process. Table 2 describes the complete list of parameters used by the functions. Tables 3, 4, 5, and 6 state which arguments use each function.

Function	Description
<code>dag.plot</code>	Plots the AON graph of a project.
<code>delay.pert</code>	Calculates the delay cost of a deterministic project and allocates it among the activities.
<code>delay.stochastic.pert</code>	Calculates the delay cost of a stochastic project and allocates it among the activities.
<code>early.time</code>	Calculates the earliest start time for each activity.
<code>last.time</code>	Calculates the latest completion time for each activity.
<code>levelling.resources</code>	Calculates the schedule of the project so that the consumption of resources is as uniform as possible.
<code>mce</code>	Calculates the costs per activity needed to accelerate the project.
<code>organize</code>	Relabels the activities of a project (if i precedes j then $i \leq j$).
<code>rebuild</code>	Builds a type 1 precedence matrix.
<code>resource.allocation</code>	Calculates the project schedule so that resource consumption does not exceed the maximum available per time period.
<code>schedule.pert</code>	Calculates the duration of a project and the schedule of each activity, and plots the schedule and the AON graph.
<code>stochastic.pert</code>	Calculates the average duration of a stochastic project, the criticality index of each activity, and the density functions of the duration of the project, early times and last times.

Table 1: Summary of functions in **ProjectManagement**.

ProjectManagement allows the user to plot the activities on nodes graph of the Project (AON). Originally, in the PERT methodology, projects are represented by activities on arcs graphs (AOA). This is the representation we have used in this paper up to now. Both AON and AOA representations are widely used in the literature, each having some advantages over the other in particular circumstances. For automatically drawing the network of a project, the AON representation is more appropriate because it is computationally much more efficient. This is why we have incorporated it into the `dag.plot` function. This representation will be useful mainly for the user to check that he has entered the precedence matrices correctly, which are the ones that really characterize the project.

ProjectManagement also allows the user to choose from four different types of immediate precedences between the activities.

- Type 1: Finish to start (FS). If an activity $i \in N$ precedes type 1 to $j \in N$, then j cannot start until activity i has finished.
- Type 2: Start to start (SS). If an activity $i \in N$ precedes type 2 to $j \in N$, then j cannot start until activity i has started.
- Type 3: Finish to finish (FF). If an activity $i \in N$ precedes type 3 to $j \in N$, then j cannot finish until activity i has finished.
- Type 4: Start to finish (SF). If an activity $i \in N$ precedes type 4 to $j \in N$, then j cannot finish until activity i has started.

The relationships between the types of dependencies are as follows: Type 1 implies type 2, type 2 implies type 4, type 1 implies type 3 and finally type 3 implies type 4. Considering these relations, if one activity precedes another by more than one type, it is only necessary to indicate the one with the strongest character.

The user can indicate types 1 or 2 in the "prec1and2" parameter (see Table 2) using the values 1 or 2 respectively, and types 3 or 4 in "prec3and4" using 3 or 4 respectively. Note that cycles can not exist.

Parameter	Description
<code>duration</code>	Vector with the expected duration for each activity.
<code>prec1and2</code>	Matrix indicating precedence type 1 or type 2 between the activities (Default= <code>matrix(0)</code>).
<code>prec3and4</code>	Matrix indicating precedence type 3 or type 4 between the activities (Default= <code>matrix(0)</code>).
<code>observed.duration</code>	Vector with the actual duration for each activity.
<code>delta</code>	Value indicating the maximum time that the project can take without delay (see equation 1). This value is only used with the default cost function.
<code>distribution</code>	Vector with the distribution function of the duration for each activity. It can be normal, triangular, exponential, uniform, beta, t-Student, F distribution, chi-squared, gamma, Weibull, binomial, Poisson, geometric, hypergeometric, and empirical.
<code>values</code>	Matrix with the arguments of the distribution function of the duration for each activity. By rows the activities, and by columns the arguments.
<code>percentile</code>	Value used to calculate the maximum time allowed for the duration of the project without delay. This value is only used if no <code>delta</code> value is assigned.
<code>compilations</code>	Number of simulations that the function uses for estimation (Default=1000).
<code>cost.function</code>	Delay cost function. If this value is not added, the package uses equation 1.
<code>early.times</code>	Vector with the early time for each activity.
<code>PRINT</code>	Logical parameter indicating if the schedule and the AON graph are depicted (Default=TRUE).
<code>plot.activities.times</code>	Vector of selected activities from which it is shown the distribution of their early and last times (Default=NULL).
<code>minimum.durations</code>	Vector with the minimum duration an activity can take even if the resources are increased.
<code>critical.activities</code>	Vector with the critical activities to represent them in a different color in the AON graph (Default=NULL).
<code>duration.project</code>	Value indicating the minimum time sought in the project (Default=NULL).
<code>activities.costs</code>	Vector indicating the cost of accelerating a unit of time the duration for each activity.
<code>resources</code>	Vector indicating the necessary resources for each activity per period of time.
<code>int</code>	Value indicating the duration of each period of time (Default=1).
<code>max.resources</code>	Value indicating the maximum number of resources that can be used in each period of time.

Table 2: Summary of parameters in `ProjectManagement`.

Function	duration	prec1and2	prec3and4	observed.duration	delta
dag.plot		✓	✓		
delay.pert	✓	✓	✓	✓	✓
delay.stochastic.pert		✓	✓	✓	✓
early.time	✓	✓	✓		
last.time	✓	✓	✓		
levelling.resources	✓	✓	✓		
mce	✓	✓	✓		
organize		✓	✓		
resource.allocation	✓	✓	✓		
schedule.pert	✓	✓	✓		
stochastic.pert		✓	✓		

Table 3: Arguments used by each function in **ProjectManagement**.

Function	distribution	values	percentile	cost.function	compilations
dag.plot					
delay.pert			✓	✓	
delay.stochastic.pert	✓	✓	✓	✓	✓
early.time					
last.time					
levelling.resources					
mce					
organize					
resource.allocation					
schedule.pert	✓	✓			✓
stochastic.pert					

Table 4: Arguments used by each function in **ProjectManagement**.

Function	early.times	PRINT	plot.activities.times	minimum.durations	critical.activities
dag.plot					✓
delay.pert					
delay.stochastic.pert					
early.time					
last.time	✓				
levelling.resources					
mce				✓	
organize					
resource.allocation					
schedule.pert		✓			
stochastic.pert			✓		

Table 5: Arguments used by each function in **ProjectManagement**.

Function	duration.project	activities.costs	resources	int	max.resources
dag.plot					
delay.pert					
delay.stochastic.pert					
early.time					
last.time					
levelling.resources			✓	✓	
mce	✓	✓			
organize					
resource.allocation			✓	✓	✓
schedule.pert					
stochastic.pert					

Table 6: Arguments used by each function in **ProjectManagement**.

Examples

ProjectManagement is available for download from CRAN. To use the package you will need to load it at the beginning of the session, usually by typing

```
> library("ProjectManagement")
```

Next we analyse the following deterministic project with 10 activities. Their durations and precedence relations are given in Table 7.

N	1	2	3	4	5	6	7	8	9	10
Immediate precedence type 1	-	-	-	2	3	3	1, 4	2	5, 8	6
Immediate precedence type 2	-	-	-	-	4	-	-	-	-	-
Immediate precedence type 3	-	-	-	-	-	8	-	-	-	-
Immediate precedence type 4	-	-	-	-	-	-	9	-	-	-
Durations	2	1.5	1	4.5	2	2.5	3	4	2	5

Table 7: Example of a deterministic project.

We start by introducing the data set characterizing the project. We use the function `dag.plot` for depicting its AON graph. Figure 2 shows it; the green blocks contain the activities and the precedences are represented by arrows. The blocks *S* and *E* are the source and sink nodes, respectively. Note that the precedences type 1 are arrows without label, precedences type 2 are labeled as SS, precedences type 3 as FF, and precedences type 4 as SF.

```
> prec1and2<-matrix(0,nrow=10,ncol=10)
> prec1and2[1,7]<-1; prec1and2[2,4]<-1; prec1and2[2,8]<-1; prec1and2[3,5]<-1;
> prec1and2[3,6]<-1; prec1and2[4,7]<-1; prec1and2[5,9]<-1; prec1and2[6,10]<-1;
> prec1and2[8,9]<-1; prec1and2[4,5]<-2
> prec3and4<-matrix(0,nrow=10,ncol=10)
> prec3and4[8,6]<-3; prec3and4[9,7]<-4
> dag.plot(prec1and2,prec3and4)
```

Using `schedule.pert`, we obtain the project schedule, i.e. the minimum time needed to complete all activities and the early and last times. Also, we can plot the schedule. Let us see it:

```
> duration<-c(2,1.5,1,1.5,2,2.5,3,4,2,5)
> schedule.pert(duration,prec1and2,prec3and4)
`Total duration of the project`
[1] 10.5
```

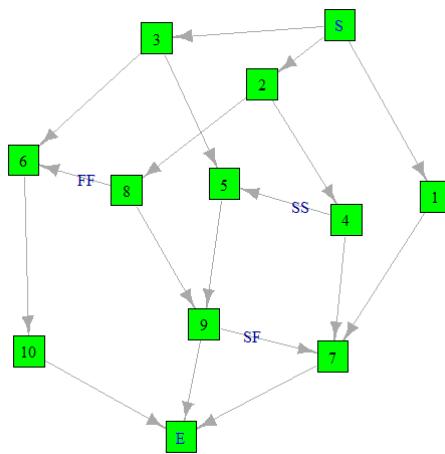


Figure 2: AON graph of the project. In an AON graph the activities are embodied in the nodes (squares) and the precedences of the various types, FS, SS, FF, SF, in the arcs (arrows).

[[2]]

Activities	Duration	Earliest start time	Latest start time	Earliest completion time
1	2.0	0.0	5.5	2.0
2	1.5	0.0	0.0	1.5
3	1.0	0.0	2.0	1.0
4	1.5	1.5	6.0	3.0
5	2.0	1.5	6.5	3.5
6	2.5	2.0	3.0	5.5
7	3.0	3.0	7.5	6.0
8	4.0	1.5	1.5	5.5
9	2.0	5.5	8.5	7.5
10	5.0	5.5	5.5	10.5

Activities	Latest completion time	Slack	Free Slack	Independent Slack
1	7.5	5.5	1.0	0.0
2	1.5	0.0	0.0	0.0
3	3.0	2.0	0.0	0.0
4	7.5	4.5	0.0	0.0
5	8.5	5.0	2.0	0.0
6	5.5	2.0	2.0	0.0
7	10.5	4.5	4.5	0.0
8	5.5	0.0	0.0	0.0
9	10.5	3.0	3.0	0.0
10	10.5	0.0	0.0	0.0

[[3]]

In this output we can see the total duration (10.5 units) of the project as well as other relevant information for each activity. Figure 3 depicts the different times of each activity using a colour coding. If we click on the points on the graph, a label indicates which activity and time it belongs to. Also, if we double click on a section of the legend we can see the data related to it, as in Figure 4 with the last times for each activity (another double click restarts the graph). In the output, the plot depicted in Figure 3 is saved as an object on [[3]]; this allows the users to manipulate the plot according to their needs. Finally, Figure 5 shows the AON graph of the project where critical activities are represented in red.

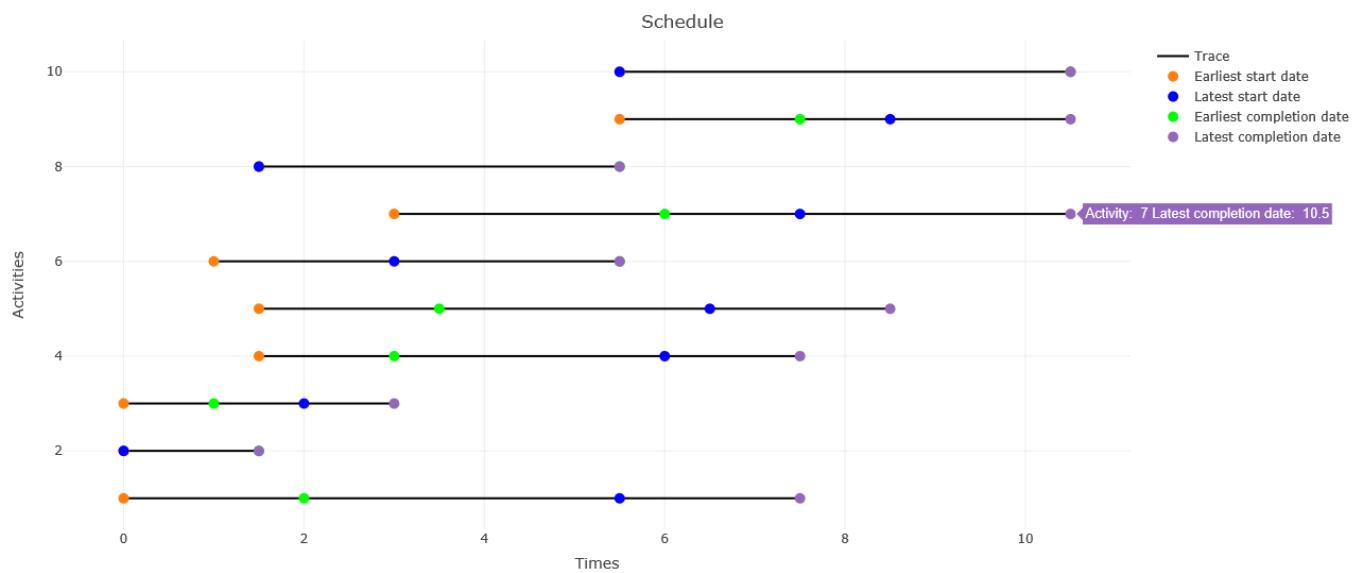


Figure 3: This figure shows an interactive graphic that displays the schedule of the project. If we move the mouse over the highlighted points of the segments, pop-up tags are generated with information about the activities.

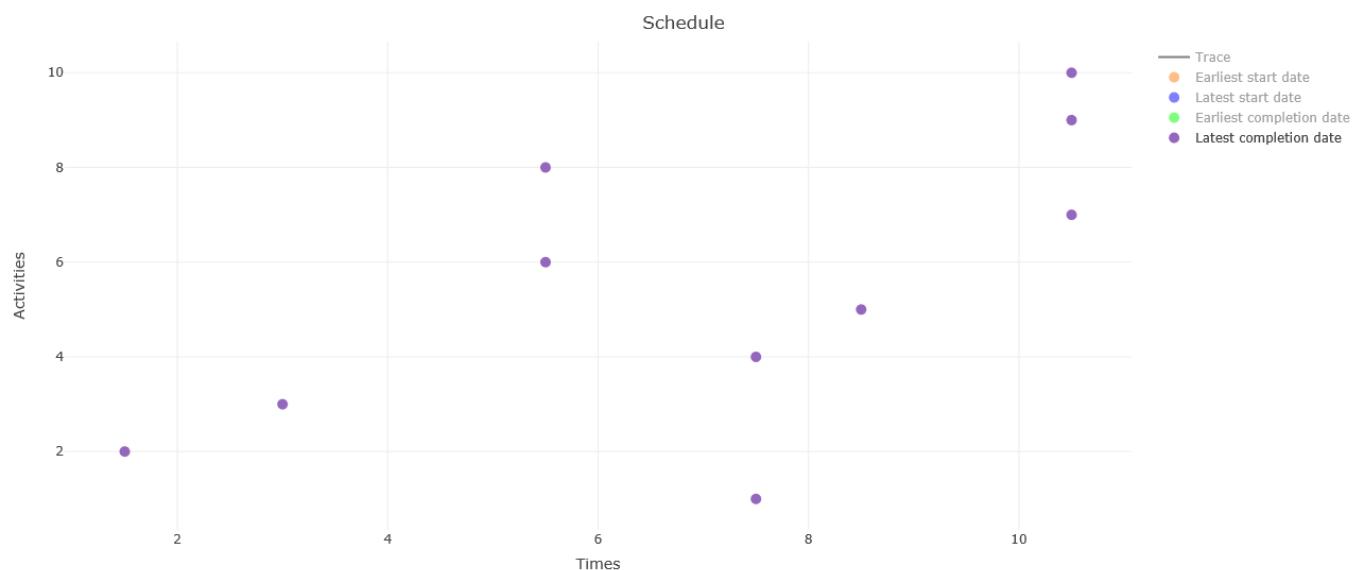


Figure 4: Latest completion times. This interactive graphic is the result of double clicking on the "Latest completion date" section of the legend.

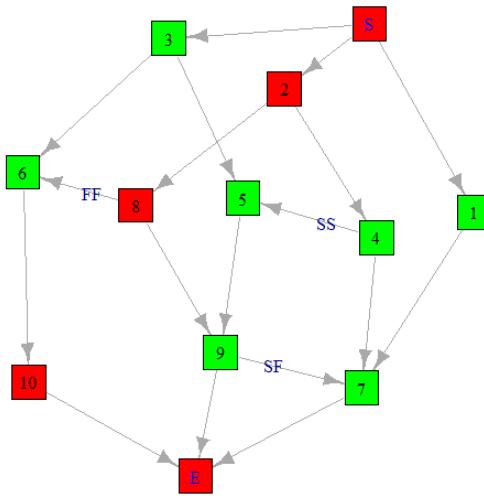


Figure 5: AON graph of the project. In an AON graph the activities are embodied in the nodes (squares) and the precedences of the various types, FS, SS, FF, SF, in the arcs (arrows). Nodes in red indicate critical activities.

Next, suppose we are interested in shortening the duration of the project. The `mce` function is used for this purpose. Let us use the function with the following input data: the minimum duration for each activity even if the resources are increased

$$\bar{x}^0 = (1, 1, 0.5, 1, 1, 2, 2, 3, 1, 3)$$

and the costs per unit time to shorten each activity

$$c = (1, 2, 1, 1, 3, 2, 1, 2, 3, 5).$$

```
> minimum.durations<-c(1,1,0.5,1,1,2,2,3,1,3)
> activities.costs<-c(1,2,1,1,3,2,1,2,3,5)
> mce(duration,minimum.durations,prec1and2,prec3and4,
activities.costs,duration.project=NULL)
necessary negative increase
1: 0.5
Read 1 item
Project duration =
[1] 10.0 9.5 9.0 8.5 8.0 7.5 7.0

Estimated durations =
Costs per solution =

2.0 2.0 2.0 2.0 2.0 2.0 2.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
1.5 1.5 1.0 1.0 1.0 1.0 1.0 0.0 0.0 1.0 1.0 1.0 1.0 1.0 1.0
1.0 1.0 1.0 1.0 1.0 1.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
1.5 1.5 1.5 1.5 1.5 1.5 1.5 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
2.0 2.0 2.0 2.0 2.0 2.0 2.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
2.5 2.5 2.5 2.5 2.5 2.5 2.5 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
3.0 3.0 3.0 3.0 3.0 3.0 3.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
3.5 3.0 3.0 3.0 3.0 3.0 3.0 1.0 2.0 2.0 2.0 2.0 2.0 2.0 2.0
2.0 2.0 2.0 2.0 2.0 2.0 2.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
5.0 5.0 5.0 4.5 4.0 3.5 3.0 0.0 0.0 0.0 2.5 5.0 7.5 10
```

The parameter `duration.project=NULL` means that we do not indicate a minimum duration of the project, so the function asks us for a decrease of the duration of the project to obtain all possible solutions. We have considered it convenient a decrease of 0.5 units of time. Therefore, we have obtained that the project can reduce its minimum duration to 10, 9.5, 9, 8.5, 8, 7.5 and 7. For each possible duration of the project, we have the durations of each activity (duration per column and activity per row), as well as the cost needed to reduce their times to these durations.

Suppose now that to complete the project each activity needs the amount of resources

$$(6, 6, 6, 3, 4, 2, 1, 2, 3, 1),$$

and we are interested in obtaining a new schedule with a uniform consumption of resources over time. To do this we use the function `levelling.resources` in such a way

```
> resources<-c(6,6,6,3,4,2,1,2,3,1)
> levelling.resources(duration,prec1and2,prec3and4,resources,int=0.5)
Earliest start times =
[1] 3.5 0.0 2.0 2.0 6.5 3.0 5.5 1.5 8.5 5.5
Resources by period=
[1] 6 6 6 2 11 11 7 10 10 10 10 2 2 6 6 6 4 4 4 4
```

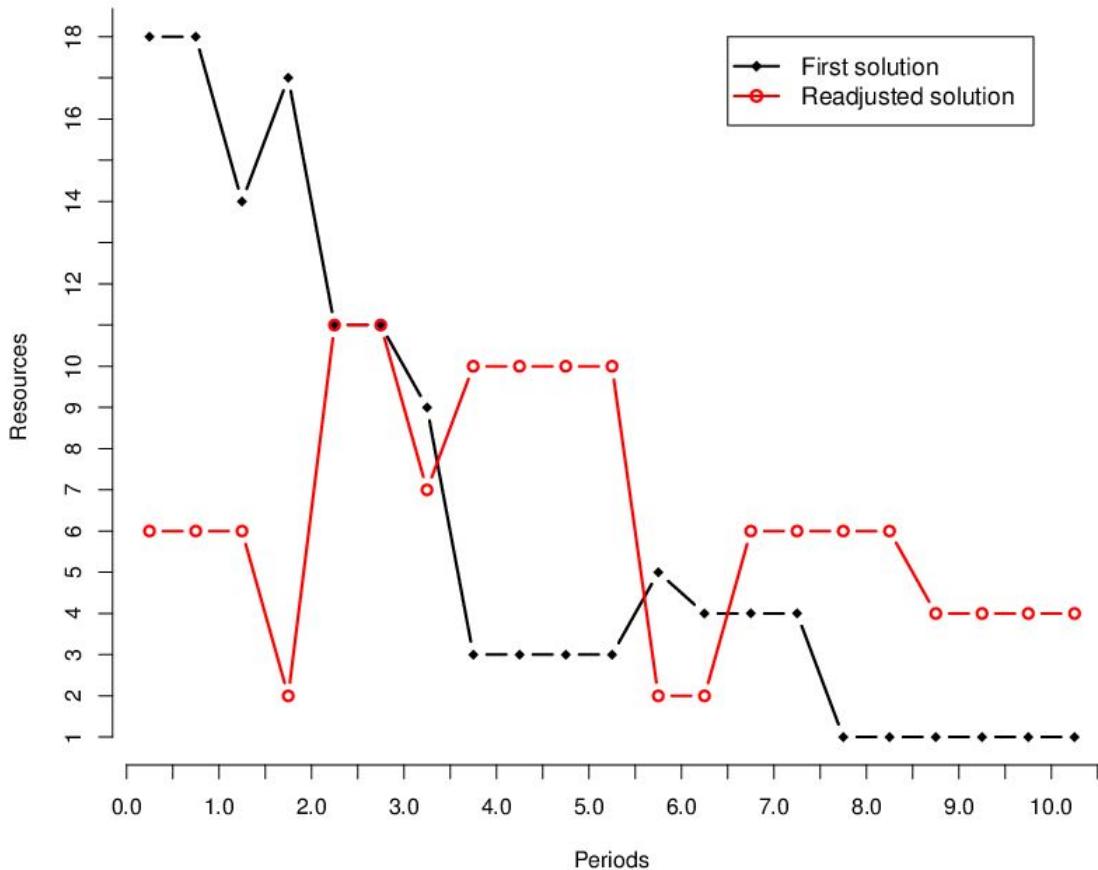


Figure 6: This graphic shows the resource consumption according to the initial scheduling (in black) and according to the scheduling after leveling (in red). The x-axis represents time and the y-axis represents resource consumption.

As we can see, the function returns the new earliest start times of the activities and the resources consumed in each period with the new schedule, where time periods start at 0 and end at 10.5 with an increase of 0.5 time units. Figure 6 represents the resources required in each period of time, before and after the readjustment.

To conclude with the analysis of resources, consider that the maximum amount of resources available in each period is 10. We use the `resource.allocation` function in this situation.

```
> max.resources<-10
> resource.allocation(duration,prec1and2,prec3and4,resources,
max.resources,int=0.5)
Project duration =
[1] 11
Earliest start times =
[1] 6.0 0.0 1.5 2.5 4.0 2.5 8.0 1.5 6.0 5.5
Resources by period =
[1] 6 6 6 8 8 7 7 8 8 6 5 10 10 10 2 2 2 2 1
```

With the new restriction, the minimum duration of the project becomes 11 instead of 10. The output includes the new earliest start times for each activity and the consumption of resources by period (note that the last period is now 11).

Continuing the example, we now analyse the allocation of delays. The function `delay.pert` shows if there has been a delay in the project and, in that case, allocates it among the activities. Let us see it using the delay cost function

$$C(D(G, y)) = \begin{cases} 0 & \text{if } D(G, y) \leq 10.5, \\ D(G, y) - 10.5 & \text{otherwise,} \end{cases}$$

and the (observed) actual durations

$$x = (2.5, 1.5, 2, 2, 2, 6, 4, 6, 3, 5.5).$$

```
> observed.duration<-c(2.5,1.5,2,2,2,6,4,6,3,5.5)
> cost.function<-function(x){return(max(x-10.5,0))}
> delay.pert(duration,prec1and2,prec3and4,observed.duration,
delta=NULL,cost.function)
There has been a delay of = 3
```

	1	2	3	4	5
The proportional payment	0.15000	0.00000	0.30000	0.15000	0.00000
The truncated proportional payment	0.15789	0.00000	0.31579	0.15789	0.00000
Shapley rule	0.00000	0.00000	0.33333	0.00000	0.00000
	6	7	8	9	10
The proportional payment	1.05000	0.30000	0.60000	0.30000	0.15000
The truncated proportional payment	0.94737	0.31579	0.63158	0.31579	0.15789
Shapley rule	1.08333	0.00000	1.08333	0.00000	0.50000

The output shows that there is a delay in the project of 3 units. As there is a delay, we proceed to make the allocation using three rules: proportional, truncated proportional and Shapley. We can see the differences between the three rules, especially in activities 1, 4, 7 and 9. While the proportional and truncated proportional rules assign a positive payment, the Shapley rule does not assign costs to these activities. This is due to the fact that, although they fall behind the planned duration, they do not affect the overall delay of the project. Note that if the project has more than ten activities, `delay.pert` does not calculate the Shapley rule; instead, it asks the user if he wants to calculate an estimate of its value.

Let us now assume that we are in a stochastic context, with additional information on planned durations being random variables. Using the function `stochastic.pert` with the following random variables to describe the duration of the activities

$$X^0 = (t(1, 2, 3), \exp(2/3), t(1/2, 5/4, 5/4), t(1/4, 7/4, 5/2), t(1, 2, 3), \\ t(1, 3/2, 5), t(1, 1, 7), t(3, 4, 5), t(1/2, 5/2, 3), t(1, 6, 8)),$$

where $t(a, b, c)$ denotes the triangular distribution with parameters a , b , and c , and $\exp(\alpha)$ denotes the exponential distribution with parameter α , we can obtain relevant information about the project. Note that with the argument `plot.activities.times=c(7,8)` we indicate the activities for which we want to estimate the densities of their earliest and latest start and completion times; in this example we have requested only such densities for activities 7 and 8.

```
> values<-matrix(c(1,3,2,2/3,0,0,1/2,5/4,5/4,1/4,5/2,7/4,1,3,2,1,5,3/2,
1,7,1,3,5,4,1/2,3,5/2,1,8,6),nrow=10,ncol=3,byrow=T)
> distribution<-c("TRIANGLE","EXPONENTIAL",rep("TRIANGLE",8))
> stochastic.pert(prec1and2,prec3and4,distribution,values,percentile=0.95,
plot.activities.times=c(7,8))
Average duration of the project = 10.64242
Percentile duration of the project = 14.21999
```

Criticality index by activity	1 0.6	2 88	3 11.4	4 2	5 0.1	6 11.3	7 2.6	8 86	9 4	10 93.4
-------------------------------	----------	---------	-----------	--------	----------	-----------	----------	---------	--------	------------

In the output we can see the average duration of the project and the percentile duration of the project. The percentile duration of the project shows the value of such that the probability that the duration of the project is smaller than d equals the variable percentile introduced by the user (see Table 2); in this case percentile=0.95. In addition, we obtain the criticality index by activity, that is, the proportion of times that an activity is critical. An activity is critical when it has zero slack. Figure 7 plots estimations of the density function of the project duration, the earliest start time and the latest completion time of activities 7 and 8.

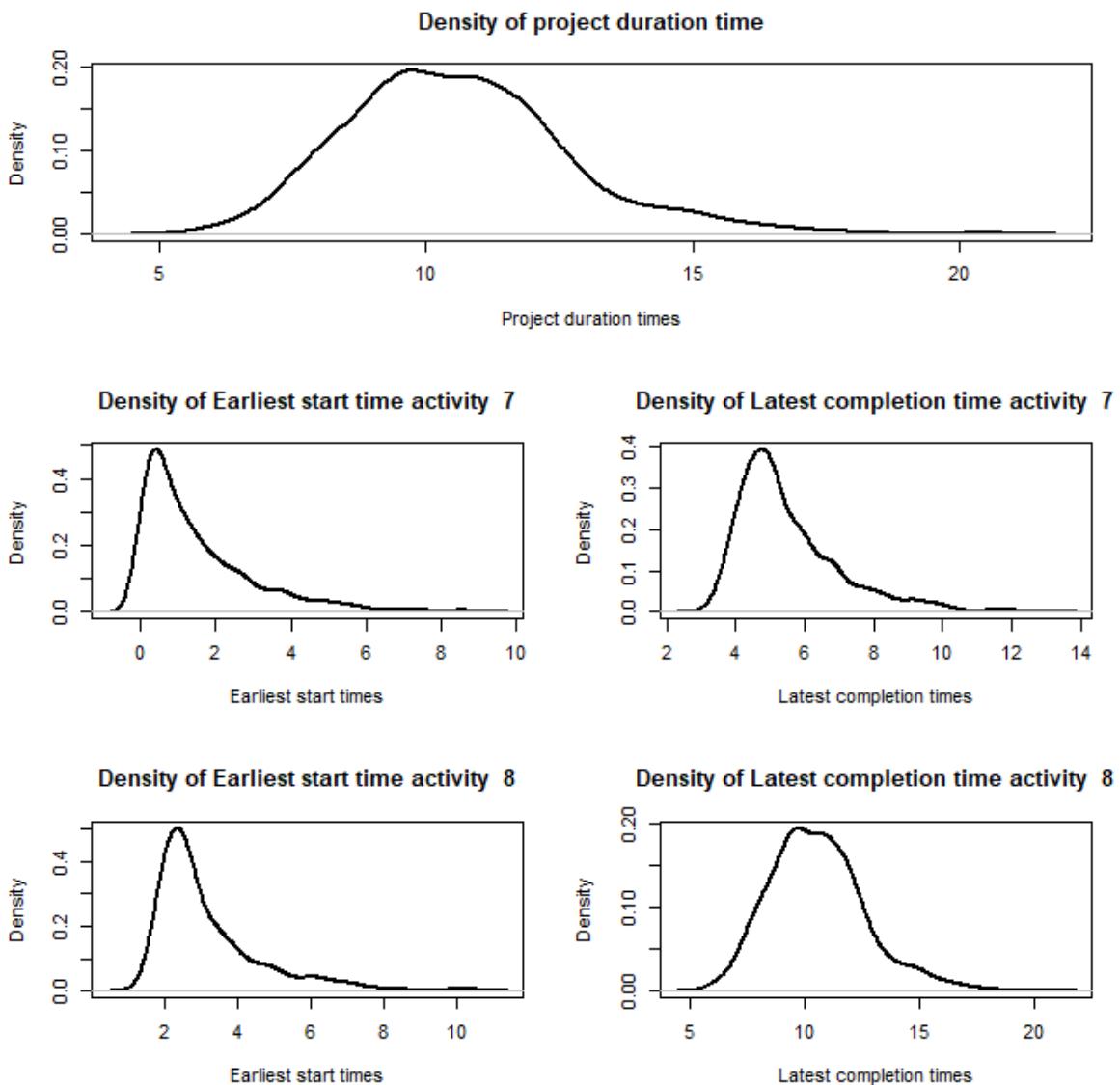


Figure 7: Density estimation of project duration time and earliest start and latest completion times for activities 7 and 8.

We proceed now to the allocation of the delay cost in the stochastic model using the function `delay.stochastic.pert`. To be able to compare the results, we will use the same delay cost function as in the deterministic case. As expected, there are noticeable differences in the allocations between the two models, as the stochastic model makes use of more complex information.

```
> delay.stochastic.pert(prec1and2,prec3and4,distribution,values,
observed.duration,percentile=NULL,delta=NULL,cost.function)
Total delay of the stochastic project = 3
```

	1	2	3	4	5
Stochastic Shapley rule	0.07238	-0.07569	0.56946	0.07440	0.07238
Stochastic Shapley rule 2	0.00693	0.03470	0.51967	0.01363	0.00731
The proportional payment	0.15000	0.00000	0.30000	0.15000	0.00000
The truncated proportional payment	0.15789	0.00000	0.31579	0.15789	0.00000
	6	7	8	9	10
Stochastic Shapley rule	1.66514	0.08271	0.11897	0.07256	0.34769
Stochastic Shapley rule 2	1.62101	0.02287	0.25006	0.01046	0.51336
The proportional payment	1.05000	0.30000	0.60000	0.30000	0.15000
The truncated proportional payment	0.94737	0.31579	0.63158	0.31579	0.15789

Finally, to illustrate the runtime of previously used functions, Table 8 shows the time (in seconds) needed to compute several problems. We have selected a variety of projects with 2, 4, 6, 8 and 10 activities, and we have run the different routines on a computer with Intel Core i5 – 7200U and 12 GB of RAM.

Activities	2	4	6	8	10
delay.pert	0.00	0.00	0.00	0.03	0.11
delay.stochastic.pert	0.06	0.44	1.68	6.23	30.58
early.time	0.00	0.00	0.00	0.00	0.00
last.time	0.00	0.00	0.00	0.00	0.00
levelling.resources	0.00	0.00	0.00	0.02	0.03
mce	0.00	0.00	0.00	0.00	0.02
organize	0.00	0.00	0.00	0.00	0.00
resources.allocation	0.00	0.00	0.00	0.01	0.02
schedule.pert	0.08	0.11	0.13	0.11	0.12
stochastic.pert	0.02	0.03	0.05	0.04	0.04

Table 8: Runtime in seconds of ProjectManagemet functions.

Acknowledgements

This work has been supported by the MINECO grant MTM2017-87197-C3-1-P and by the Xunta de Galicia through the ERDF (Grupos de Referencia Competitiva ED431C-2016-015 and Centro Singular de Investigación de Galicia ED431G/01). The comments of an anonymous reviewer have helped us to improve this paper significantly.

Bibliography

- G. Bergantiños, A. Valencia-Toledo, and J. Vidal-Puga. Hart and mas-colell consistency in pert problems. *Discrete Applied Mathematics*, 243:11–20, 2018. URL <https://doi.org/10.1016/j.dam.2017.08.012>. [p]
- R. Brânzei, G. Ferrari, V. Fragnelli, and S. Tijs. Two approaches to the problem of sharing delay costs in joint projects. *Annals of Operations Research*, 109(1):359–374, 2002. URL <https://doi.org/10.1023/A:1016372707256>. [p]
- S. Cano-Berlanga. *GameTheory: Cooperative Game Theory*, 2017. URL <https://CRAN.R-project.org/package=GameTheory>. R package version 2.7. [p]
- R. Carnell. *triangle: Provides the Standard Distribution Functions for the Triangle Distribution*, 2019. URL <https://CRAN.R-project.org/package=triangle>. R package version 0.12. [p]
- J. Castro, D. Gómez, and J. Tejada. A project game for pert networks. *Operations Research Letters*, 35(6):791–798, 2007. URL <https://doi.org/10.1016/j.orl.2007.01.003>. [p]

- J. Castro, D. Gómez, and J. Tejada. Polynomial calculation of the shapley value based on sampling. *Computers and Operations Research*, 36(5):1726–1730, 2009. URL <https://doi.org/10.1016/j.cor.2008.04.004>. [p]
- G. Csardi and T. Nepusz. The igraph software package for complex network research. *InterJournal, Complex Systems*:1695, 2006. URL <http://igraph.org>. [p]
- I. V. Evdokimov, R. Y. Tsarev, T. N. Yamskikh, and A. N. Pupkov. Using pert and gantt charts for planning software projects on the basis of distributed digital ecosystems. In *Journal of Physics: Conference Series*, volume 1074, page 012127. IOP Publishing, 2018. URL <https://doi.org/10.1088/1742-6596/1074/1/012127>. [p]
- J. C. Gonçalves-Dosantos, I. García-Jurado, and J. Costa. Sharing delay costs in stochastic scheduling problems with delays. *4OR - A Quarterly Journal of Operations Research*, 2020a. URL <https://doi.org/10.1007/s10288-019-00427-9>. [p]
- J. C. Gonçalves-Dosantos, I. García-Jurado, and J. Costa. *ProjectManagement: Management of Deterministic and Stochastic Projects*, 2020b. URL <https://CRAN.R-project.org/package=ProjectManagement>. R package version 1.3.3. [p]
- M. Grabisch, I. Kojadinovic, and P. Meyer. *kappalab: Non-Additive Measure and Integral Manipulation Functions*, 2015. URL <https://CRAN.R-project.org/package=kappalab>. R package version 0.4-7. [p]
- G. Gregoriou, K. Kirytopoulos, and C. Kiriklidis. Project management educational software (promes). *Computer Applications in Engineering Education*, 21(1):46–59, 2013. URL <https://doi.org/10.1002/cae.20450>. [p]
- N. G. Hall. Project management: Recent developments and research opportunities. *Journal of Systems Science and Systems Engineering*, 21(2):129–143, 2012. URL <https://doi.org/10.1007/s11518-012-5190-5>. [p]
- T. Hegazy. Optimization of resource allocation and leveling using genetic algorithms. *Journal of construction engineering and management*, 125(3):167–175, 1999. URL [https://doi.org/10.1061/\(ASCE\)0733-9364\(1999\)125:3\(167\)](https://doi.org/10.1061/(ASCE)0733-9364(1999)125:3(167)). [p]
- F. S. Hillier and G. J. Lieberman. *Introduction to Operations Research*. McGraw-Hill, 2001. ISBN 0072321695. [p]
- D. Kelley. *plan: Tools for Project Planning*, 2018. URL <https://CRAN.R-project.org/package=plan>. R package version 0.4-3. [p]
- J. E. Kelley. Critical-path planning and scheduling: Mathematical basis. *Operations Research*, 9(3):296–320, 1961. URL <https://doi.org/10.1287/opre.9.3.296>. [p]
- lp_solve, K. Konis, and F. Schwendinger. *lpSolveAPI: R Interface to lp_solve Version 5.5.2.0*, 2020. URL <https://CRAN.R-project.org/package=lpSolveAPI>. R package version 5.5.2.0-17.7. [p]
- S. Moretti and F. Patrone. Transversality of the shapley value. *Top*, 16(1):1–41, 2008. URL <https://doi.org/10.1007/s11750-008-0044-5>. [p]
- J. C. Muñoz. *PlotPrjNetworks: Useful Networking Tools for Project Management*, 2015. URL <https://CRAN.R-project.org/package=PlotPrjNetworks>. R package version 1.0.0. [p]
- L. Özdamar and G. Ulusoy. A survey on the resource-constrained project scheduling problem. *IIE transactions*, 27(5):574–586, 1995. URL <https://doi.org/10.1080/07408179508936773>. [p]
- L. Salas-Morera, A. Arauzo-Azofra, L. García-Hernández, J. M. Palomo-Romero, and H.-M. César. Ppcproject: An educational tool for software project management. *Computers and Education*, 69:181–188, 2013. URL <https://doi.org/10.1016/j.comedu.2013.07.018>. [p]
- K. Schmitz, R. Mahapatra, and S. Nerur. User engagement in the era of hybrid agile methodology. *IEEE Software*, 36(4):32–40, 2019. URL <https://doi.org/10.1109/MS.2018.290100623>. [p]
- L. S. Shapley. A value for n-person games. *Contributions to the Theory of Games*, 2(28):307–317, 1953. URL <https://doi.org/10.1016/j.comedu.2013.07.018>. [p]
- C. Sievert. *Interactive Web-Based Data Visualization with R, plotly, and shiny*. Chapman and Hall/CRC, 2020. ISBN 9781138331457. URL <https://plotly-r.com>. [p]

Juan Carlos Gonçalves-Dosantos

*Grupo MODES, CITIC and Departamento de Matemáticas, Universidade da Coruña
Campus de Elviña, 15071 A Coruña
Spain*

ORCID: 0000-0003-1578-8411

juan.carlos.goncalves@udc.es

Ignacio García-Jurado

*Grupo MODES, CITIC and Departamento de Matemáticas, Universidade da Coruña
Campus de Elviña, 15071 A Coruña
Spain*

ORCID: 0000-0002-6681-1629

ignacio.garcia.jurado@udc.es

Julian Costa

*Grupo MODES, Departamento de Matemáticas, Universidade da Coruña
Campus de Elviña, 15071 A Coruña
Spain*

ORCID: 0000-0001-9760-9581

julian.costa@udc.es

The Rockerverse: Packages and Applications for Containerisation with R

by Daniel Nüst, Dirk Eddelbuettel, Dom Bennett, Robrecht Cannoodt, Dav Clark, Gergely Daróczy, Mark Edmondson, Colin Fay, Ellis Hughes, Lars Kjeldgaard, Sean Lopp, Ben Marwick, Heather Nolis, Jacqueline Nolis, Hong Ooi, Karthik Ram, Noam Ross, Lori Shepherd, Péter Sólymos, Tyson Lee Swetnam, Nitesh Turaga, Charlotte Van Petegem, Jason Williams, Craig Willis, Nan Xiao

Abstract The Rocker Project provides widely used Docker images for R across different application scenarios. This article surveys downstream projects that build upon the Rocker Project images and presents the current state of R packages for managing Docker images and controlling containers. These use cases cover diverse topics such as package development, reproducible research, collaborative work, cloud-based data processing, and production deployment of services. The variety of applications demonstrates the power of the Rocker Project specifically and containerisation in general. Across the diverse ways to use containers, we identified common themes: reproducible environments, scalability and efficiency, and portability across clouds. We conclude that the current growth and diversification of use cases is likely to continue its positive impact, but see the need for consolidating the Rockerverse ecosystem of packages, developing common practices for applications, and exploring alternative containerisation software.

Introduction

The R community continues to grow. This can be seen in the number of new packages on CRAN, which is still on growing exponentially (Hornik et al., 2019), but also in the numbers of conferences, open educational resources, meetups, unconferences, and companies that are adopting R, as exemplified by the useR! conference series¹, the global growth of the R and R-Ladies user groups², or the foundation and impact of the R Consortium³. These trends cement the role of R as the *lingua franca* of statistics, data visualisation, and computational research. The last few years, coinciding with the rise of R, have also seen the rise of Docker as a general tool for distributing and deploying of server applications—in fact, Docker can be called the *lingua franca* of describing computing environments and packaging software. Combining both these topics, the *Rocker Project* (<https://www.rocker-project.org/>) provides Docker images with R (see the next section for more details). The considerable uptake and continued evolution of the Rocker Project has led to numerous projects that extend or build upon Rocker images, ranging from reproducible⁴ research to production deployments. As such, this article presents what we may call the *Rockerverse* of projects across all development stages: early demonstrations, working prototypes, and mature products. We also introduce related activities that connect the R language and environment with other containerisation solutions. Our main contribution is a coherent picture of the current status of using containers in, with, and for R.

The article continues with a brief introduction of containerisation basics and the Rocker Project, followed by use cases and applications, starting with the R packages specifically for interacting with Docker, next the second-level packages that use containers indirectly or only for specific features, and finally some complex use cases that leverage containers. We conclude by reflecting on the landscape of packages and applications and point out future directions of development.

Containerisation and Rocker

Docker, an application and service provided by the eponymous company, has, in just a few short years, risen to prominence for developing, testing, deploying and distributing computer software

¹<https://www.r-project.org/conferences/>

²<https://www.r-consortium.org/blog/2019/09/09/r-community-explorer-r-user-groups>, <https://www.r-consortium.org/blog/2019/08/12/r-community-explorer>

³<https://www.r-consortium.org/news/announcements>, <https://www.r-consortium.org/blog/2019/11/14/data-driven-tracking-and-discovery-of-r-consortium-activities>

⁴“Reproducible” in the sense of the *Claerbout/Donoho/Peng* terminology (Barba, 2018).

(cf. Datadog, 2018; Muñoz, 2019). While related approaches exist, such as LXC⁵ or Singularity (Kurtzer et al., 2017), Docker has become synonymous with “containerisation”—the method of taking software artefacts and bundling them in such a way that use becomes standardized and portable across operating systems. In doing so, Docker had recognised and validated the importance of one very important thread that had been emerging, namely virtualisation. By allowing (one or possibly) multiple applications or services to run concurrently on one host machine without any fear of interference between them, Docker provides an important scalability opportunity. Beyond this though, Docker has improved this compartmentalisation by accessing the host system—generally Linux—through a much thinner and smaller shim than a full operating system emulation or virtualisation. This containerisation, also called operating-system-level virtualisation (Wikipedia contributors, 2020b), makes efficient use of operating system resources (Felter et al., 2015) and allows another order of magnitude in terms of scalability of deployment (cf. Datadog, 2018), because virtualisation may emulate a whole operating system, a container typically runs only one process. The single process together with sharing the host’s kernel results in a reduced footprint and faster start times. While Docker makes use of Linux kernel features, it has become important enough that some required aspects of running Docker have been added to other operating systems so that those systems can more efficiently support Docker (Microsoft, 2019b). The success of Docker has even paved the way for industry collaboration and standardisation (OCI, 2019).

The key accomplishment of Docker as an “application” is to make a “bundled” aggregation of software, the so-called “image”, available to any system equipped to run Docker, without requiring much else from the host besides the actual Docker application installation. This is a rather attractive proposition, and Docker’s very easy to operate user interface has led to widespread adoption and use of Docker in a variety of domains, e.g., cloud computing infrastructure (e.g., Bernstein, 2014), data science (e.g., Boettiger, 2015), and edge computing (e.g., Alam et al., 2018). It has also proven to be a natural match for “cloud deployment” which runs, or at least appears to run, “seamlessly” without much explicit reference to the underlying machine, architecture or operating system: Containers are portable and can be deployed with very little dependencies on the host system—only the container runtime is required. These Docker images are normally built from plain text documents called **Dockerfiles**; a **Dockerfile** has a specific set of instructions to create and document a well-defined environment, i.e., install specific software and expose specific ports.

For statistical computing and analysis centred around R, the **Rocker Project** has provided a variety of Docker containers since it began in 2014 (Boettiger and Eddelbuettel, 2017). The Rocker Project provides several lines of containers spanning from building blocks with **R-release** or **R-devel**, via containers with **RStudio Server** and **Shiny Server**, to domain-specific containers such as **rocker/geospatial** (Boettiger et al., 2019). These containers form *image stacks*, building on top of each other for easier maintainability (i.e., smaller **Dockerfiles**), better composability, and to reduce build time. Also of note is a series of “versioned” containers which match the R release they contain with the *then-current* set of packages via the MRAN Snapshot views of CRAN (Microsoft, 2019a). The Rocker Project’s impact and importance was acknowledged by the Chan Zuckerberg Initiative’s *Essential Open Source Software for Science*, which provides funding for the project’s sustainable maintenance, community growth, and targeting new hardware platforms including GPUs (Chan Zuckerberg Initiative et al., 2019).

Docker is not the only containerisation software. **Singularity** stems from the domain of high-performance computing (Kurtzer et al., 2017) and can also run Docker images. Rocker images work out of the box if the main process is R, e.g., in **rocker/r-base**, but Singularity does not succeed in running images where there is an init script, e.g., in containers that by default run RStudio Server. In the latter case, a **Singularity** file, a recipe akin to a **Dockerfile**, needs to be used to make necessary adjustments. To date, no comparable image stack to the Rocker Project’s images exists on **Singularity Hub**. A further tool for running containers is **podman**, which can also build **Dockerfiles** and run Docker images. Proof of concepts exists for using podman to build and run Rocker containers⁶, but the prevalence of Docker, especially in the broader user community beyond experts or niche systems and the vast amount of blog posts and courses for Docker currently cap specific development efforts for both Singularity and podman in the R community. This might quickly change if the usability and spread of Singularity or podman increase, or if security features such as rootless/unprivileged containers, which both these tools support out of the box, become more sought after.

⁵<https://en.wikipedia.org/wiki/LXC>

⁶See <https://github.com/nuest/rodman> and <https://github.com/rocker-org/rocker-versioned/issues/187>

Functionality	stevedore	harbor	googleCloudRunner	dockyard	dockermachine	babelwhale	AzureContainers
Generate a Dockerfile						✓	
Build an image	✓			✓	✓		
Execute a container locally or remotely	✓	✓	✓	✓	✓	✓	✓
Deploy or manage instances in the cloud	✓		✓		✓	✓	✓
Interact with an instance (e.g., file transfer)		✓	✓				✓
Manage storage of images						✓	✓
Supports Docker and Singularity			✓				
Direct access to Docker API instead of using the CLI							✓
Installing Docker software					✓		

Table 1: R packages with Docker client functionality.

Interfaces for Docker in R

Users interact with the Docker daemon typically through the [Docker Command Line Interface](#) (Docker CLI). However, moving back and forth between an R console and the command line can create friction in workflows and reduce reproducibility because of manual steps. A number of first-order R packages provide an interface to the Docker CLI, allowing for the interaction with the Docker CLI from an R console. Table 1 gives an overview of packages with client functionality, each of which provides functions for interacting with the Docker daemon. The packages focus on different aspects and support different stages of a container’s life cycle. As such, the choice of which package is most useful depends on the use case at hand as well as on the user’s level of expertise.

harbor (<https://github.com/wch/harbor>) is no longer actively maintained, but it should be honourably mentioned as the first R package for managing Docker images and containers. It uses the `sys` package (Ooms, 2019) to run system commands against the Docker CLI, both locally and through an SSH connection, and it has convenience functions, e.g., for listing and removing containers/images and for accessing logs. The outputs of container executions are converted to appropriate R types. The Docker CLI’s basic functionality, although it evolves quickly and with little concern for avoiding breaking changes, has remained unchanged in core functions, meaning that a core function such as `harbor::docker_run(image = "hello-world")` still works despite its stopped development.

stevedore is currently the most powerful Docker client in R (FitzJohn, 2020). It interfaces with the Docker daemon over the Docker HTTP API⁷ via a Unix socket on Linux or MacOS, over a named pipe on Windows, or over an HTTP/TCP connection. The package is the only one not using system calls to the `docker` CLI tool for managing images and containers. The package thereby enables connections to remote Docker instances without direct configuration of the local Docker daemon. Furthermore using the API gives access to information in a structured way, is system independent, and is likely more reliable than parsing command line output. **stevedore**’s own interface is automatically generated based on the OpenAPI specification of the Docker daemon, but it is still similar to the Docker CLI. The interface is similar to R6 objects, in that an object of class “`stevedore_object`” has a number of functions attached to it that can be called, and multiple specific versions of the Docker API can be supported thanks to the automatic generation⁸.

AzureContainers is an interface to a number of container-related services in Microsoft’s [Azure Cloud](#) (Ooi, 2019). While it is mainly intended for working with Azure, as a convenience feature it includes lightweight, cross-platform shells to Docker and Kubernetes (tools `kubectl` and `helm`).

⁷<https://docs.docker.com/engine/api/latest/>

⁸See <https://github.com/richfitz/stevedore/blob/master/development.md>.

These can be used to create and manage arbitrary Docker images and containers, as well as Kubernetes clusters on any platform or cloud service.

`googleCloudRunner` is an interface with [Google Cloud Platform](#) container-related services, with tools to make it easier for R users to interact with them for common use cases ([Edmondson, 2020](#)). It includes deployment functions for creating R APIs using the Docker-based [Cloud Run](#) service. Users can create long running batch jobs calling any Docker image including Rocker via [Cloud Build](#) and schedule services using [Cloud Scheduler](#).

`babelwhale` provides a unified interface to interact with Docker and Singularity containers ([Cannoodt and Saelens, 2019](#)). Users can, for example, execute a command inside a container, mount a volume, or copy a file with the same R commands for both container runtimes.

`dockyard` (<https://github.com/thebioengineer/dockyard>) has the goal of lowering the barrier to creating [Dockerfiles](#), building Docker images, and deploying Docker containers. The package follows the increasingly used piping paradigm of the Tidyverse-style ([Wickham et al., 2019](#)) of programming for chaining R functions representing the instructions in a [Dockerfile](#). An existing [Dockerfile](#) can be used as a template. `dockyard` also includes wrappers for common steps, such as installing an R package or copying files, and provides built-in functions for building an image and running a container, which make Docker more approachable within a single R-based user interface.

`dockermachine` (<https://github.com/cboettig/dockermachine>) is an R package to provide a convenient interface to [Docker Machine](#) from R. The CLI tool `docker-machine` allows users to create and manage a virtual host on local computers, local data centres, or at cloud providers. A local Docker installation can be configured to transparently forward all commands issued on the local Docker CLI to a selected (remote) virtual host. Docker Machine was especially crucial for local use in the early days of Docker, when no native support was available for Mac or Windows computers, but it remains relevant for provisioning on remote systems. The package has not received any updates for two years, but it is functional with a current version of `docker-machine` (0.16.2). It potentially lowers the barriers for R users to run containers on various hosts if they perceive that using the Docker Machine CLI directly as a barrier and it enables scripted workflows with remote processing.

Use cases and applications

Image stacks for communities of practice

Bioconductor (<https://bioconductor.org/>) is an open-source, open development project for the analysis and comprehension of genomic data ([Gentleman et al., 2004](#)). As of October 30th 2019, the project consists of 1823 R software packages, as well as packages containing annotation or experiment data. *Bioconductor* has a semi-annual release cycle, where each release is associated with a particular version of R, and Docker images are provided for current and past versions of *Bioconductor* for convenience and reproducibility. All images, which are described on the *Bioconductor* web site (see <https://bioconductor.org/help/docker/>), are created with [Dockerfiles](#) maintained on GitHub and distributed through Docker Hub⁹. *Bioconductor*'s “base” Docker images are built on top of the `rocker/rstudio` image. *Bioconductor* installs packages based on the R version in combination with the Bioconductor version and, therefore, uses Bioconductor version tagging `devel` and `RELEASE_X_Y`, e.g., `RELEASE_3_10`. Past and current combinations of R and *Bioconductor* will therefore be accessible via specific image tags.

The *Bioconductor Dockerfile* selects the desired R version from Rocker images, adds required system dependencies, and uses the `BiocManager` package for installing appropriate versions of *Bioconductor* packages ([Morgan, 2019](#)). A strength of this approach is that the responsibility for complex software configuration and customization is shifted from the user to the experienced *Bioconductor* core team. However, a recent audit of the *Bioconductor* image stack `Dockerfile` led to the deprecation of several community-maintained images, because the numerous specific images became too hard to understand, complex to maintain, and cumbersome to customise. As part of the simplification, a recent innovation is the `bioconductor_docker:devel` image, which emulates the *Bioconductor* environment for nightly builds as closely as possible. This image contains the environment variables and the system dependencies needed to install and check almost all *Bioconductor* software packages (1813 out of 1823). It saves users and package developers from creating this environment themselves. Furthermore, the image is configured so that `.libPaths()` has ‘`/usr/local/lib/R/host-site-library`’ as the first location. Users mounting a location on the host file system to this location can persistently manage installed packages across Docker contain-

⁹See https://github.com/Bioconductor/bioconductor_docker and <https://hub.docker.com/u/bioconductor> respectively.

ers or image updates. Many R users pursue flexible workflows tailored to particular analysis needs rather than standardized workflows. The new `bioconductor_docker` image is well suited for this preference, while `bioconductor_docker:devel` provides developers with a test environment close to *Bioconductor*'s build system.

Data science is a widely discussed topic in all academic disciplines (e.g., [Donoho, 2017](#)). These discussions have shed light on the tools and craftsmanship behind the analysis of data with computational methods. The practice of data science often involves combining tools and software stacks and requires a cross-cutting skillset. This complexity and an inherent concern for openness and reproducibility in the data science community has led to Docker being used widely. The remainder of this section presents example Docker images and image stacks featuring R intended for data science.

- The [Jupyter Docker Stacks](#) project is a set of ready-to-run Docker images containing Jupyter applications and interactive computing tools ([Jupyter, 2018](#)). The `jupyter/r-notebook` image includes R and “popular packages”, and naturally also the IRKernel (<https://irkernel.github.io/>), an R kernel for Jupyter, so that Jupyter Notebooks can contain R code cells. R is also included in the catchall `jupyter/datascience-notebook` image¹⁰. For example, these images allow users to quickly start a Jupyter Notebook server locally or build their own specialised images on top of stable toolsets. R is installed using the Conda package manager¹¹, which can manage environments for various programming languages, pinning both the R version and the versions of R packages¹².
- *Kaggle* provides the `gcr.io/kaggle-images/rstats` image (previously `kaggle/rstats`) and `corresponding Dockerfile` for usage in their Machine Learning competitions and easy access to the associated datasets. It includes machine learning libraries such as Tensorflow and Keras (see also image `rocker/ml` in Section [Common or public work environments](#)), and it also configures the `reticulate` package ([Ushey et al., 2019](#)). The image uses a base image with *all packages from CRAN*, `gcr.io/kaggle-images/rcran`, which requires a Google Cloud Build because Docker Hub would time out¹³. The final extracted image size is over 25 GB, which calls into question whether having everything available is actually convenient.
- The [Radiant](#) project provides several images, e.g., `vnijs/rsm-msba-spark`, for their browser-based business analytics interface based on [Shiny](#) ([Chang et al., 2019](#)), and for use in education as part of an MSc course¹⁴. As data science often applies a multitude of tools, this image favours inclusion over selection and features Python, Postgres, JupyterLab and Visual Studio Code besides R and RStudio, bringing the image size up to 9 GB.
- *Gigantum* (<http://gigantum.com/>) is a platform for open and decentralized data science with a focus on using automation and user-friendly tools for easy sharing of reproducible computational workflows. Gigantum builds on the *Gigantum Client* (running either locally or on a remote server) for development and execution of data-focused *Projects*, which can be stored and shared via the *Gigantum Hub* or via a zipfile export. The Client is a user-friendly interface to a backend using Docker containers to package, build, and run Gigantum projects. It is configured to use a default set of Docker base images (<https://github.com/gigantum/base-images>), and users are able to define and configure their own custom images. The available images include two with R based on Ubuntu Linux and these have the `c2d4u CRAN PPA` pre-configured for installation of binary R packages¹⁵. The R images vary in the included authoring environment, i.e., Jupyter in `r-tidyverse` or both Jupyter & RStudio in `rstudio-server`. The independent image stack can be traced back to the Gigantum environment and its features. The R images are based on Gigantum's `python3-minimal` image, originally to keep the existing front-end configuration, but also to provide consistent Python-to-R interoperability. The `Dockerfiles` also use build args to specify bases, for example for different versions of NVIDIA CUDA for GPU processing¹⁶, so that appropriate GPU drivers can be enabled automatically when supported. Furthermore, Gigantum's focus lies on environment management via GUI and ensuring a smooth user interaction, e.g., with reliable and easy conflict detection and resolution. For this reason, project repositories store authoritative package information in a separate file per package, allowing Git to directly detect conflicts

¹⁰<https://jupyter-docker-stacks.readthedocs.io/en/latest/using/selecting.html>

¹¹<https://conda.io/>

¹²See `jupyter/datascience-notebook`'s `Dockerfile` at <https://github.com/jupyter/docker-stacks/blob/master/datascience-notebook/Dockerfile#L47>.

¹³Originally, a stacked collection of over 20 images with automated builds on Docker Hub was used, see <https://web.archive.org/web/20190606043353/http://blog.kaggle.com/2016/02/05/how-to-get-started-with-data-science-in-containers/> and <https://hub.docker.com/r/kaggle/rcran/dockerfile>

¹⁴'Dockerfile' available on GitHub: <https://github.com/radiant-rstats/docker>.

¹⁵<https://docs.gigantum.com/docs/using-r>

¹⁶See https://github.com/gigantum/base-images/blob/master/_templates/python3-minimal-template/Dockerfile for the `Dockerfile` of `python3-minimal`.

and changes. A Dockerfile is generated from this description that inherits from the specified base image, and additional custom Docker instructions may be appended by users, though Gigantium's default base images do not currently include the `littler` tool, which is used by Rocker to install packages within `Dockerfiles`. Because of these specifics, instructions from `rocker/r-ubuntu` could *not* be readily re-used in this image stack (see Section [Conclusions](#)). Both approaches enable the `apt` package manager ([Wikipedia contributors, 2020a](#)) as an installation method, and this is exposed via the GUI-based environment management¹⁷. The image build and publication process is scripted with Python and JSON template configuration files, unlike Rocker images which rely on plain `Dockerfiles`. A further reason in the creation of an independent image stack were project constraints requiring a Rocker-incompatible licensing of the `Dockerfiles`, i.e., the MIT License.

Capture and create environments

Community-maintained images provide a solid basis so users can meet their own individual requirements. Several second-order R packages attempt to streamline the process of creating Docker images and using containers for specific tasks, such as running tests or rendering reproducible reports. While authoring and managing an environment with Docker by hand is possible and feasible for experts¹⁸, the following examples show that when environments become too cumbersome to create manually, automation is a powerful tool. In particular, the practice of *version pinning*, with system package managers for different operating systems and with packages `remotes` and `versions` or by using MRAN for R, can greatly increase the reproducibility of built images and are common approaches.

`dockefiler` is an R package designed for building `Dockerfiles` straight from R ([Fay, 2019](#)). A scripted creation of a `Dockerfile` enables iteration and automation, for example for packaging applications for deployment (see [Deployment and continuous delivery](#)). Developers can retrieve system requirements and package dependencies to write a `Dockerfile`, for example, by leveraging the tools available in R to parse a `DESCRIPTION` file.

`containerit` (<https://github.com/o2r-project/containerit/>) attempts to take this one step further and includes these tools to automatically create a `Dockerfile` that can execute a given workflow ([Nüst and Hinz, 2019](#)). `containerit` accepts an R object of classes "`sessionInfo`" or "`session_info`" as input and provides helper functions to derive these from workflows, e.g., an R script or R Markdown document, by analysing the session state at the end of the workflow. It relies on the `sysreqs` (<https://github.com/r-hub/sysreqs/>) package and its mapping of package system dependencies to platform-specific installation package names¹⁹. `containerit` uses `stevedore` to streamline the user interaction and improve the created `Dockerfiles`, e.g., by running a container for the desired base image to extract the already available R packages.

`dockr` is a similar package focusing on the generation of Docker images for R packages, in which the package itself and all of the R dependencies, including local non-CRAN packages, are available ([Kjeldgaard, 2019a,b](#)). `dockr` facilitates the organisation of code in the R package structure and the resulting Docker image mirrors the package versions of the current R session. Users can manually add statements for non-R dependencies to the `Dockerfile`.

`liftr` ([Xiao, 2019](#)) aims to solve the problem of persistent reproducible reporting in statistical computing based on the R Markdown format ([Xie et al., 2018](#)). The irreproducibility of authoring environments can become an issue for collaborative documents and large-scale platforms for processing documents. `liftr` makes the dynamic R Markdown document the main and sole workflow control file and the only file that needs to be shared between collaborators for consistent environments, e.g. demonstrated in the DockFlow project (<https://dockflow.org>). It introduces new fields to the document header, allowing users to manually declare the versioned dependencies required for rendering the document. The package then generates a `Dockerfile` from this metadata and provides a utility function to render the document inside a Docker container, i.e., `render_docker("foo.Rmd")`. An RStudio addin even allows compilation of documents with the single push of a button.

System dependencies are the domain of Docker, but for a full description of the computing environment, one must also manage the R version and the R packages. R versions are available via the versioned Rocker image stack ([Boettiger and Eddelbuettel, 2017](#)). `r-online` leverages these images and provides an app for helping users to detect breaking changes between different R versions and for historic exploration of R. With a standalone NodeJS app or `r-online`, the user can compare

¹⁷See <https://docs.gigantium.com/docs/environment-management>

¹⁸See, e.g., this tutorial by RStudio on how to manage environments and package versions and to ensure deterministic image builds with Docker: <https://environments.rstudio.com/docker>.

¹⁹See <https://sysreqs.r-hub.io/>.

a piece of code run in two separate versions of R. Internally, `r-online` opens one or two Docker instances with the given version of R based on Rocker images, executes a given piece of code, and returns the result to the user. Regarding R package management, this can be achieved with MRAN, or with packages such as `checkpoint` (Ooi et al., 2020) and `renv` (Ushey, 2020), which can naturally be applied within images and containers. For example, `renv` helps users to manage the state of the R library in a reproducible way, further providing isolation and portability. While `renv` does not cover system dependencies, the `renv`-based environment can be transferred into a container either by restoring the environment based on the main configuration file `renv.lock` or by storing the `renv`-cache on the host and not in the container (Ushey, 2019). With both the system dependencies and R packages consciously managed in a Docker image, users can start using containers as the *only* environment for their workflows, which allows them to work independently of physical computers²⁰ and to assert a specific degree of confidence in the stability of a developed software (cf. `README.Rmd` in Marwick, 2017).

Development, debugging, and testing

Containers can also serve as playgrounds and provide specific or ad hoc environments for the purposes of developing R packages. These environments may have specific versions of R, of R extension packages, and of system libraries used by R extension packages, and all of the above in a specific combination.

First, such containers can greatly facilitate **fixing bugs and code evaluation**, because developers and users can readily start a container to investigate a bug report or try out a piece of software (cf. Ooms, 2017). The container can later be discarded and does not affect their regular system. Using the Rocker images with RStudio, these disposable environments lack no development comfort (cf. Section [Packaging research reproducibly](#)). Ooms (2017) describes how `docker exec` can be used to get a root shell in a container for customisation during software evaluation without writing a `Dockerfile`. Eddelbuettel and Koenker (2019) describes an example of how a Docker container was used to debug an issue with a package only occurring with a particular version of Fortran, and using tools which are not readily available on all platforms (e.g., not on macOS).

Second, the strong integration of **system libraries in core packages** in the [R-spatial community](#) makes containers essential for stable and proactive development of common classes for geospatial data modelling and analysis. For example, GDAL (GDAL/OGR contributors, 2019) is a crucial library in the geospatial domain. GDAL is a system dependency allowing R packages such as `sf`, which provides the core data model for geospatial vector data, or `rgdal`, to accommodate users to be able to read and write hundreds of different spatial raster and vector formats (Pebesma, 2018; Bivand et al., 2019). `sf` and `rgdal` have hundreds of indirect reverse imports and dependencies and, therefore, the maintainers spend a lot of effort trying not to break them. Purpose-built Docker images are used to prepare for upcoming releases of system libraries, individual bug reports, and for the lowest supported versions of system libraries²¹.

Third, special-purpose images exist for identifying problems beyond the mere R code, such as **debugging R memory problems**. These images significantly reduce the barriers to following complex steps for fixing memory allocation bugs (cf. Section 4.3 in R Core Team, 1999). These problems are hard to debug and critical, because when they do occur they lead to fatal crashes. `rocker/r-devel-san` and `rocker/r-devel-ubsan-clang` are Docker images that have a particularly configured version of R to trace such problems with gcc and clang compilers, respectively (cf. `sanitizers` for examples, Eddelbuettel, 2014). `wch/r-debug` is a purpose-built Docker image with *multiple* instrumented builds of R, each with a different diagnostic utility activated.

Fourth, containers are useful for **testing** R code during development. To submit a package to CRAN, an R package must work with the development version of R, which must be compiled locally; this can be a challenge for some users. The **R-hub** project provides “*a collection of services to help R package development*”, with the package builder as the most prominent one (R-hub project, 2019). R-hub makes it easy to ensure that no errors occur, but fixing errors still often warrants a local setup, e.g., using the image `rocker/r-devel`, as is testing packages with native code, which can make the process more complex (cf. Eckert, 2018). The R-hub Docker images can also be used to debug problems locally using various combinations of Linux platforms, R versions, and compilers²². The images go beyond the configurations, or *flavours*, used by CRAN for checking packages²³,

²⁰Allowing them to be digital “nomads”, cf. J. Bryan’s <https://github.com/jennybc/docker-why>.

²¹Cf. <https://github.com/r-spatial/sf/tree/master/inst/docker>, https://github.com/Nowosad/rspatial_proj6, and <https://github.com/r-spatial/sf/issues/1231>

²²See <https://r-hub.github.io/rhub/articles/local-debugging.html> and <https://blog.r-hub.io/2019/04/25/r-devel-linux-x86-64-debian-clang/>

²³https://cran.r-project.org/web/checks/check_flavors.html

e.g., with CentOS-based images, but they lack a container for checking on Windows or OS X. The images greatly support package developers to provide support on operating systems with which they are not familiar. The package **dockertest** (<https://github.com/traitecoevo/dockertest/>) is a proof of concept for automatically generating **Dockerfiles** and building images specifically to run tests²⁴. These images are accompanied by a special launch script so the tested source code is not stored in the image; instead, the currently checked in version from a local Git repository is cloned into the container at runtime. This approach separates the test environment, test code, and current working copy of the code. Another use case where a container can help to standardise tests across operating systems is detailed the vignettes of the package **RSelenium** (Harrison, 2019). The package recommends Docker for running the **Selenium** Server application needed to execute test suites on browser-based user interfaces and webpages, but it requires users to manually manage the containers.

Fifth, Docker images can be used on continuous integration (CI) platforms to streamline the testing of packages. Ye (2019) describes how they speed up the process of testing by running tasks on **Travis CI** within a container using **docker exec**, e.g., the package check or rendering of documentation. Cardozo (2018) also saved time with Travis CI by re-using the testing image as the basis for an image intended for publication on Docker Hub. **r-ci** is, in turn, used with **GitLab CI**, which itself is built on top of Docker images: the user specifies a base Docker image and control code, and the whole set of tests is run inside a container. The **r-ci** image stack combines **rocker** versioning and a series of tools specifically designed for testing in a fixed environment with a customized list of preinstalled packages. Especially for long-running tests or complex system dependencies, these approaches to separate installation of build dependencies with code testing streamline the development process. Containers can also simplify the integration of R software into larger, multi-language CI pipelines. Furthermore, with each change, even this manuscript is rendered into a PDF and deployed to a GitHub-hosted website (see **.travis.yml** and **Dockerfile** in the manuscript repository), not because of concern about time, but to control the environment used on a CI server. This gives, on the one hand, easy access after every update of the R Markdown source code and, on the other hand, a second controlled environment to make sure that the article renders successfully and correctly.

Processing

The portability of containerised environments becomes particularly useful for improving expensive processing of data or shipping complex processing pipelines. First, it is possible to offload complex processing to a server or clouds and also to execute processes in parallel to speed up or to serve many users. **batchtools** provides a parallel implementation of the **Map** function for various schedulers (Lang et al., 2017). For example, the package can schedule jobs with Docker Swarm. **googleComputeEngineR** has the function **gce_vm_cluster()** to create clusters of 2 or more virtual machines, running multi-CPU architectures (Edmondson, 2019). Instead of running a local R script with the local CPU and RAM restrictions, the same code can be processed on all CPU threads of the cluster of machines in the cloud, all running a Docker container with the same R environments. **googleComputeEngineR** integrates with the R parallelisation package **future** (Bengtsson, 2020a) to enable this with only a few lines of R code²⁵. Google Cloud Run is a CaaS (Containers as a Service) platform. Users can launch containers using any Docker image without worrying about underlying infrastructure in a so-called serverless configuration. The service takes care of network ingress, scaling machines up and down, authentication, and authorisation—all features which are non-trivial for a developer to build and maintain on their own. This can be used to scale up R code to millions of instances if need be with little or no changes to existing code, as demonstrated by the proof of concept **cloudRunR**²⁶, which uses Cloud Run to create a scalable R-based API using **plumber** (Trestle Technology, LLC, 2018). Google Cloud Build and the Google Container Registry are a continuous integration service and an image registry, respectively, that offload building of images to the cloud, while serving the needs of commercial environments such as private Docker images or image stacks. As Google Cloud Build itself can run any container, the package **googleCloudRunner** demonstrates how R can be used as the control language for one-time or batch processing jobs and scheduling of jobs²⁷. **drake** is a workflow manager for data science projects (Landau, 2018). It features implicit parallel computing and automated detection of the parts of the work that actually needs to be re-executed. **drake** has been demonstrated to run inside containers for high

²⁴**dockertest** is not actively maintained, but mentioned still because of its interesting approach.

²⁵<https://cloudry.github.io/googleComputeEngineR/articles/massive-parallel.html>

²⁶<https://github.com/MarkEdmondson1234/cloudRunR>

²⁷<https://code.markedmondson.me/googleCloudRunner/articles/cloudbuild.html>

reproducibility²⁸. Furthermore, **drake** workflows have been shown to use **future** package's function `makeClusterPSOCK()` for sending parts of the workflow to a Docker image for execution²⁹ (see package's function documentation; Bengtsson, 2020b). In the latter case, the container control code must be written by the user, and the **future** package ensures that the host and worker can connect for communicating over socket connections. **RStudio Server Pro** includes a functionality called **Launcher** (since version 1.2, released in 2019). It gives users the ability to spawn R sessions and background/batch jobs in a scalable way on external clusters, e.g., **Kubernetes based on Docker images** or **Slurm** clusters, and optionally, with Singularity containers. A benefit of the proprietary Launcher software is the ability for R and Python users to leverage containerisation's advantages in RStudio without writing specific deployment scripts or learning about Docker or managing clusters at all.

Second, containers are perfectly suited for **packaging and executing software pipelines** and required data. Containers allow for building complex processing pipelines that are independent of the host programming language. Due to its original use case (see [Introduction](#)), Docker has no standard mechanisms for chaining containers together; it lacks definitions and protocols for how to use environment variables, volume mounts, and/or ports that could enable the transfer of input (parameters and data) and output (results) to and from containers. Some packages, e.g., **containerit**, provide Docker images that can be used very similar to a CLI, but this usage is cumbersome³⁰. **outsider** (<https://docs.ropensci.org/outsider/>) tackles the problem of integrating external programs into an R workflow without the need for users to directly interact with containers (Bennett et al., 2020). Installation and usage of external programs can be difficult, convoluted and even impossible if the platform is incompatible. Therefore, **outsider** uses the platform-independent Docker images to encapsulate processes in *outsider modules*. Each outsider module has a `Dockerfile` and an R package with functions for interacting with the encapsulated tool. Using only R functions, an end-user can install a module with the **outsider** package and then call module code to seamlessly integrate a tool into their own R-based workflow. The **outsider** package and module manage the containers and handle the transmission of arguments and the transfer of files to and from a container. These functionalities also allow a user to launch module code on a remote machine via SSH, expanding the potential computational scale. Outsider modules can be hosted code-sharing services, e.g., on GitHub, and **outsider** contains discovery functions for them.

Deployment and continuous delivery

The cloud is the natural environment for containers, and, therefore, containers are the go-to mechanism for deploying R server applications. More and more continuous integration (CI) and continuous delivery (CD) services also use containers, opening up new options for use. The controlled nature of containers, i.e., the possibility to abstract internal software environment from a minimal dependency outside of the container is crucial, for example to match test or build environments with production environments or transfer runnable entities to as-a-service infrastructures.

First, different packages use containers for the **deployment of R and Shiny apps**. **Shiny** is a popular package for creating interactive online dashboards with R, and it enables users with very diverse backgrounds to create stable and user-friendly web applications (Chang et al., 2019). **ShinyProxy** (<https://www.shinyproxy.io/>) is an open-source tool to deploy Shiny apps in an enterprise context, where it features single sign-on, but it can also be used in scientific use cases (e.g., Savini et al., 2019; Glouzon et al., 2017). ShinyProxy uses Docker containers to isolate user sessions and to achieve scalability for multi-user scenarios with multiple apps. ShinyProxy itself is written in Java to accommodate corporate requirements and may itself run in a container for stability and availability. The tool is built on ContainerProxy (<https://www.containerproxy.io/>), which provides similar features for executing long-running R jobs or interactive R sessions. The started containers can run on a regular Docker host but also in clusters. Continuous integration and deployment (CI/CD) for Shiny applications using Shinyproxy can be achieved, e.g., via GitLab pipelines or with a combination of GitHub and Docker Hub. A pipeline can include building and checking R packages and Shiny apps. After the code has passed the checks, Docker images are built and pushed to the container registry. The pipeline finishes with triggering a `webhook` on the server, where the deployment script is executed. The script can update configurations or pull the new Docker images. There is a **ShinyProxy 1-Click App** in the DigitalOcean marketplace that is set up with these webhooks. The [documentation](#) explains how to set up HTTPS with ShinyProxy and webhooks.

²⁸See for example <https://github.com/joelnitta/pleurosoriopsis> or <https://gitlab.com/ecohealthalliance/drake-gitlab-docker-example>, the latter even running in a continuous integration platform (cf. [Development, debugging, and testing](#)).

²⁹<https://docs.ropensci.org/drake/index.html?q=docker#with-docker>

³⁰<https://o2r.info/containerit/articles/container.html>

Another example is the package `golem`, which makes heavy use of `dockerfiler` when it comes to creating the `Dockerfile` for building and deploying production-grade Shiny applications (Guyader et al., 2019). `googleComputeEngineR` enables quick deployments of key R services, such as RStudio and Shiny, onto cloud virtual machines (VMs) with Google Cloud Compute Engine (Edmondson, 2019). The package utilises `Dockerfiles` to move the labour of setting up those services from the user to a premade Docker image, which is configured and run in the cloud VM. For example, by specifying the template `template="rstudio"` in functions `gce_vm_template()` and `gce_vm()` an up-to-date RStudio Server image is launched for development work, whereas specifying `template="rstudio-gpu"` will launch an RStudio Server image with a GPU attached, etc.

Second, containers can be used to create **platform installation packages** in a DevOps setting. The `OpenCPU` system provides an HTTP API for data analysis based on R. Ooms (2017) describes how various platform-specific installation files for OpenCPU are created using Docker Hub. The automated builds install the software stack from the source code on different operating systems; afterwards a script file downloads the images and extracts the OpenCPU binaries.

Third, containers can greatly facilitate the **deployment to existing infrastructures**. `Kubernetes` (<https://kubernetes.io/>) is a container-orchestration system for managing container-based application deployment and scaling. A *cluster* of containers, orchestrated as a single deployment, e.g., with Kubernetes, can mitigate limitations on request volumes or a container occupied with a computationally intensive task. A cluster features load-balancing, autoscaling of containers across numerous servers (in the cloud or on premise), and restarting failed ones. Many organisations already use a Kubernetes cluster for other applications, or a managed cluster can be acquired from service providers. Docker containers are used within Kubernetes clusters to hold native code, for which Kubernetes creates a framework around network connections and scaling of resources up and down. Kubernetes can thereby host R applications, big parallel tasks, or scheduled batch jobs in a scalable way, and the deployment can even be triggered by changes to code repositories (i.e., CD, see Edmondson, 2018). The package `googleKubernetesR` (<https://github.com/RhysJackson/googleKubernetesR>) is a proof of concept for wrapping the Google Kubernetes Engine API, Google's hosted Kubernetes solution, in an easy-to-use R package. The package `analogsea` provides a way to programmatically create and destroy cloud VMs on the Digital Ocean platform (Chamberlain et al., 2019). It also includes R wrapper functions to install Docker in such a VM, manage images, and control containers straight from R functions. These functions are translated to Docker CLI commands and transferred transparently to the respective remote machine using SSH. `AzureContainers` is an umbrella package that provides interfaces for three commercial services of Microsoft's Azure Cloud, namely `Container Instances` for running individual containers, `Container Registry` for private image distribution, and `Kubernetes Service` for orchestrated deployments. While a package like `plumber` provides the infrastructure for turning an R workflow into a web service, for production purposes it is usually necessary to take into account scalability, reliability and ease of management. `AzureContainers` provides an R-based interface to these features and, thereby, simplifies complex infrastructure management to a number of R function calls, given an Azure account with sufficient credit³¹. `Heroku` is another cloud platform as a service provider, and it supports container-based applications. `heroku-docker-r` (<https://github.com/virtualstaticvoid/heroku-docker-r>) is an independent project providing a template for deploying R applications based on Heroku's image stack, including multiple examples for interfacing R with other programming languages. Yet the approach requires manual management of the computing environment.

Independent integrations of R for different cloud providers lead to repeated efforts and code fragmentation. To mitigate these problems and to avoid vendor lock-in motivated the `OpenFaaS` project. OpenFaaS facilitates the deployment of functions and microservices to Kubernetes or Docker Swarm. It is language-agnostic and provides auto-scaling, metrics, and an API gateway. Reduced boilerplate code is achieved via templates. Templates for R³² are provided based on Rocker's Debian and R-hub's `r-minimal` Alpine images. The templates use multi-stage Docker builds to combine R base images with the OpenFaaS 'watchdog', a tiny Golang web server. The watchdog marshals an HTTP request and invokes the actual application. The R session uses `plumber` or similar packages for the API endpoint with packages and data preloaded, thus minimizing response times.

The prevalence of Docker in industry naturally leads to the use of R in containers, as companies already manage platforms in Docker containers. These products often entail a large amount of open-source software in combination with proprietary layers adding the relevant commercialisation features. One such example is RStudio's data science platform `RStudio Team`. It allows teams of data scientists and their respective IT/DevOps groups to develop and deploy code in R and Python

³¹See "Deploying a prediction service with Plumber" vignette for details: https://cran.r-project.org/web/packages/AzureContainers/vignettes/vig01_plumber_deploy.html.

³²See OpenFaaS R templates at <https://github.com/analythium/openfaas-rstats-templates>.

around the RStudio Open-Source Server inside of Docker images, without requiring users to learn new tools or directly interact with containers. The best practices for running RStudio with Docker containers as well as Docker images for RStudio's commercial products are publicly available.

Using R to power enterprise software in production environments

R has been historically viewed as a tool for analysis and scientific research, but not for creating software that corporations can rely on for production services. However, thanks to advancements in R running as a web service, along with the ability to deploy R in Docker containers, modern enterprises are now capable of having real-time machine learning powered by R. A number of packages and projects have enabled R to respond to client requests over TCP/IP and local socket servers, such as **Rserve** (Urbanek, 2019), **svSocket** (Grosjean, 2019), **rApache** and more recently **plumber** (<https://www.rplumber.io/>) and **RestRserve** (<http://restrserve.org>), which even processes incoming requests in parallel with forked processes using **Rserve**. The latter two also provide documentation for deployment with Docker or ready-to-use images with automated builds³³. These software allow other (remote) processes and programming languages to interact with R and to expose R-based function in a service architecture with HTTP APIs. APIs based on these packages can be deployed with scalability and high availability using containers. This pattern of deploying code matches those used by software engineering services created in more established languages in the enterprise domain, such as Java or Python, and R can be used alongside those languages as a first-class member of a software engineering technical stack.

CARD.com implemented a web application for the optimisation of the acquisition flow and the real-time analysis of debit card transactions. The software used **Rserve** and **rApache** and was deployed in Docker containers. The R session behind **Rserve** acted as a read-only in-memory database, which was extremely fast and scalable, for the many concurrent **rApache** processes responding to the live-scoring requests of various divisions of the company. Similarly deodorised R scripts were responsible for the ETL processes and even the client-facing email, text message and push notification alerts sent in real-time based on card transactions. The related Docker images were made available at <https://github.com/cardcorp/card-rocker>. The images extended **rocker/r-base** and additionally entailed an SSH client and a workaround for being able to mount SSH keys from the host, Pandoc, the Amazon Web Services (AWS) SDK, and Java, which is required by the AWS SDK. The AWS SDK allowed for running R consumers reading from real-time data processing streams of **AWS Kinesis**³⁴. The applications were deployed on Amazon Elastic Container Service (**ECS**). The main takeaways from using R in Docker were not only that pinning the R package versions via MRAN is important, but also that moving away from Debian testing to a distribution with long-term support can be necessary. For the use case at hand, this switch allowed for more control over upstream updates and for minimising the risk of breaking the automated builds of the Docker images and production jobs.

The AI @ T-Mobile team created a set of machine learning models for natural language processing to help customer care agents manage text-based messages from customers (T-Mobile et al., 2018). For example, one model identifies whether a message is from a customer (see **Shiny**-based demo further described by Nolis and Werdell, 2019), and others tell which customers are likely to make a repeat purchase. If a data scientist creates a such a model and exposes it through a **plumber** API, then someone else on the marketing team can write software that sends different emails depending on that real-time prediction. The models are convolutional neural networks that use the **keras** package (Allaire and Chollet, 2019) and run in a Rocker container. The corresponding **Dockerfiles** are published on GitHub. Since the models power tools for agents and customers, they need to have extremely high uptime and reliability. The AI @ T-Mobile team found that the models performed well, and today these models power real-time services that are called over a million times a day.

Common or public work environments

The fact that Docker images are portable and well defined make them useful when more than one person needs access to the same computing environment. This is even more useful when some of the users do not have the expertise to create such an environment themselves, and when these environments can be run in public or using shared infrastructure. For example, **RCloud** (<https://rcloud.social>) is a cloud-based platform for data analysis, visualisation and collaboration using

³³See <https://www.rplumber.io/docs/hosting.html#docker>, <https://hub.docker.com/r/trestletech/plumber/> and <https://hub.docker.com/r/rexyai/restrserve/>.

³⁴See useR!2017 talk "Stream processing with R in AWS".

R. It provides a `rocker/dr` base image for easy evaluation of the platform³⁵.

The **Binder** project, maintained by the team behind Jupyter, makes it possible for users to **create and share computing environments** with others (Jupyter et al., 2018). A *BinderHub* allows anyone with access to a web browser and an internet connection to launch a temporary instance of these custom environments and execute any workflows contained within. From a reproducibility standpoint, Binder makes it exceedingly easy to compile a paper, visualize data, and run small examples from papers or tutorials without the need for any local installation. To set up Binder for a project, a user typically starts at an instance of a BinderHub and passes the location of a repository with a workspace, e.g., a hosted Git repository, or a data repository like Zenodo. Binder's core internal tool is `repo2docker`. It deterministically builds a Docker image by parsing the contents of a repository, e.g., project dependency configurations or simple configuration files³⁶. In the most powerful case, `repo2docker` builds a given *Dockerfile*. While this approach works well for most run-of-the-mill Python projects, it is not so seamless for R projects. This is partly because `repo2docker` does not support arbitrary base images due to the complex auto-generation of the *Dockerfile* instructions.

Two approaches make using Binder easier for R users. First, **holepunch** (<https://github.com/karthik/holepunch>) is an R package that was designed to make sharing work environments accessible to novice R users based on Binder. For any R projects that use the Tidyverse suite (Wickham et al., 2019), the time and resources required to build all dependencies from source can often time out before completion, making it frustrating for the average R user. **holepunch** removes some of these limitations by leveraging Rocker images that contain the Tidyverse along with special Jupyter dependencies, and only installs additional packages from CRAN and Bioconductor that are not already part of these images. It short circuits the configuration file parsing in `repo2docker` and starts with the Binder/Tidyverse base images, which eliminates a large part of the build time and, in most cases, results in a Binder instance launching within a minute. **holepunch** also creates a `DESCRIPTION` file for essential metadata and dependency specification, and thereby turns any project into a research compendium (see [Packaging research reproducibly](#)). The *Dockerfile* included with the project can also be used to launch an RStudio Server instance locally, i.e., independent of Binder, which is especially useful when more or special computational resources can be provided there. The local image usage reduces the number of separately managed environments and, thereby, reduces work and increases portability and reproducibility.

Second, the **Whole Tale** project (<https://whole-tale.org>) combines the strengths of the Rocker Project's curated Docker images with `repo2docker`. Whole Tale is a National Science Foundation (NSF) funded project developing a scalable, open-source, multi-user platform for reproducible research (Brinckman et al., 2019; Chard et al., 2019b). A central goal of the platform is to enable researchers to easily create and publish executable research objects³⁷ associated with published research (Chard et al., 2019a). Using Whole Tale, researchers can create and publish Rocker-based reproducible research objects to a growing number of repositories including DataONE member nodes, Zenodo and soon Dataverse. Additionally, Whole Tale supports automatic data citation and is working on capabilities for image preservation and provenance capture to improve the transparency of published computational research artefacts (Mecum et al., 2018; McPhillips et al., 2019). For R users, Whole Tale extends the Jupyter Project's `repo2docker` tool to simplify the customisation of R-based environments for researchers with limited experience with either Docker or Git. Multiple options have been discussed to allow users to change the Ubuntu LTS (long-term support, currently *Bionic Beaver*) base image, `buildpack-deps:bionic`, used in `repo2docker`. Whole Tale implemented a custom `RockerBuildPack`³⁸. The build pack combines a `rocker/geospatial` image with `repo2docker`'s composability³⁹. This works because both Rocker images and the `repo2docker` base image use distributions with APT ([Wikipedia contributors](#), 2020a) so that the instructions created by the latter work because of the compatible shell and package manager.

In **high-performance computing**, one use for containers is to run workflows on shared local hardware where teams manage their own high-performance servers. This can follow one of several design patterns: Users may deploy containers to hardware as a work environment for a specific project, containers may provide per-user persistent environments, or a single container can act as a common multi-user environment for a server. In all cases, though, the containerised approach provides several advantages: First, users may use the same image and thus work environment on

³⁵<https://github.com/att/rcloud/tree/master/docker>

³⁶See supported file types at https://repo2docker.readthedocs.io/en/latest/config_files.html. For R, the

³⁷In Whole Tale a *tale* is a research object that contains metadata, data (by copy or reference), code, narrative, documentation, provenance, and information about the computational environment to support computational reproducibility.

³⁸See https://github.com/whole-tale/repo2docker_wholetale.

³⁹Composability refers to the ability to combine multiple package managers and their configuration files, such as R, 'pip', and 'conda'; see Section [Common or public work environments](#) for details.

desktop and laptop computers. The first two patterns provide modularity, while the last approach is most similar to a simple shared server. Second, software updates can be achieved by updating and redeploying the container rather than by tracking local installs on each server. Third, the containerised environment can be quickly deployed to other hardware, cloud or local, if more resources are necessary or in case of server destruction or failure. In any of these cases, users need a method to interact with the containers, be it an IDE exposed over an HTTP port or command-line access via tools such as SSH. A suitable method must be added to the container recipes. The Rocker Project provides containers pre-installed with the RStudio IDE. In cases where users store nontrivial amounts of data for their projects, the data needs to persist beyond the life of the container. This may be in shared disks, attached network volumes, or in separate storage where it is uploaded between sessions. In the case of shared disks or network-attached volumes, care must be taken to match user permissions, and of course backups are still necessary.

CyVerse is an open-source, NSF-funded cyberinfrastructure platform for the life sciences providing easy access to computing and storage resources (Merchant et al., 2016). CyVerse has a browser-based ‘data science workbench’ called the [Discovery Environment](#) (DE). The DE uses a combination of [HTCondor](#) and Kubernetes for orchestrating container-based analysis and integrates with external HPC, i.e., [NSF-XSEDE](#), through [TAPIS](#) (TACC-API’s). CyVerse hosts a multi-petabyte Data Store based on [iRODS](#) with shared access by its users. The DE runs Docker containers on demand, with users able to integrate bespoke containers from DockerHub or other registries (Devisetty et al., 2016). Rocker image integration in the DE is designed to provide researchers with scalable, compute-intensive, R analysis capabilities for large and complex datasets (e.g., genomics/multi-omics, GWAS, phenotypic data, geospatial data, etc.). These capabilities give users flexibility similar to Binder, but allow containers to be run on larger computational resources (RAM, CPU, Disk, GPU), and for longer periods of time (days to weeks). The Rocker Project’s RStudio and Shiny are integrated into the DE by deriving new images from Rocker images⁴⁰. These new images include a reverse proxy using [nginx](#) to handle communication with CyVerse’s authentication system ([RStudio Support](#), 2020); CyVerse also allows owners to invite other registered users to securely access the same instance. The CyVerse Rocker images further include tools for connecting to its Data Store, such as the CLI utility [icommands](#) for iRODS. CyVerse accounts are free (with some limitations for non-US users), and the [CyVerse Learning Center](#) provides community members with information about the platform, including training and education opportunities.

Using **GPUs** (graphical processing units) as specialised hardware from containerised common work environments is also possible and useful (Haydel et al., 2015). GPUs are increasingly popular for compute-intensive machine learning (ML) tasks, e.g., deep artificial neural networks (Schmidhuber, 2015). Although in this case containers are not completely portable between hardware environments, but the software stack for ML with GPUs is so complex to set up that a ready-to-use container is helpful. Containers running GPU software require drivers and libraries specific to GPU models and versions, and containers require a specialized runtime to connect to the underlying GPU hardware. For NVIDIA GPUs, the [NVIDIA Container Toolkit](#) includes a specialized runtime plugin for Docker and a set of base images with appropriate drivers and libraries. The Rocker Project [has a repository](#) with (beta) images based on these that include GPU-enabled versions of machine-learning R packages, e.g., [rocker/ml](#) and [rocker/tensorflow-gpu](#).

Teaching

Two use cases demonstrate the practical usefulness and advantages of containerisation in the context of teaching. On the one hand a special case of shared computing environments (see Section 4.7), and on the other hand leveraging sandboxing and controlled environments for auto-grading.

Prepared environments for teaching are especially helpful for (a) introductory courses, where students often struggle with the first step of installation and configuration ([Çetinkaya Rundel and Rundel, 2018](#)), and (b) courses that require access to a relatively complex setup of software tools, e.g., database systems. Çetinkaya Rundel and Rundel (2018) describe how a Docker-based deployment of RStudio (i) avoided problems with troubleshooting individual students’ computers and greatly increased engagement through very quickly showing tangible outcomes, e.g., a visualisation, and (ii) reduced demand on teaching and IT staff. Each student received access to a personal RStudio instance running in a container after authentication with the university login, which gives the benefits of sandboxing and the possibility of limiting resources. Çetinkaya Rundel and Rundel (2018) found that for the courses at hand, actual usage of the UI is intermittent so a single cloud-based VM with four cores and 28 GB RAM sufficed for over 100 containers. An example for mitigating *complex setups* is teaching databases. R is very useful tool for interfacing

⁴⁰See <https://github.com/cyverse-vice/> for Dockerfiles and configuration scripts; images are auto-built on DockerHub at <https://hub.docker.com/u/cyversevice>.

ing with databases, because almost every open-source and proprietary database system has an R package that allows users to connect and interact with it. This flexibility is even broadened by **DBI** (R Special Interest Group on Databases (R-SIG-DB) et al., 2019), which allows for creating a common API for interfacing these databases, or the **dbplyr** package (Wickham and Ruiz, 2019), which runs **dplyr** (Wickham et al., 2020) code straight against the database as queries. But learning and teaching these tools comes with the cost of deploying or having access to an environment with the software and drivers installed. For people teaching R, it can become a barrier if they need to install local versions of database drivers or connect to remote instances which might or might not be made available by IT services. Giving access to a sandbox for the most common environments for teaching databases is the idea behind **r-db**, a Docker image that contains everything needed to connect to a database from R. Notably, with **r-db**, users do not have to install complex drivers or configure their machine in a specific way. The **rocker/tidyverse** base image ensures that users can also readily use packages for analysis, display, and reporting.

The idea of a common environment and partitioning allows for using **containers in teaching for secure execution and automated testing** of submissions by students. First, **Dodona** is a web platform developed at Ghent University that is used to teach students basic programming skills, and it uses Docker containers to test submissions by students. This means that both the code testing the students' submissions and the submission itself are executed in a predictable environment, avoiding compatibility issues between the wide variety of configurations used by students. The containerisation is also used to shield the Dodona servers from bad or even malicious code: memory, time and I/O limits are used to make sure students cannot overload the system. The web application managing the containers communicates with them by sending configuration information as a JSON document over standard input. Every Dodona Docker image shares a **main.sh** file that passes through this information to the actual testing framework, while setting up some error handling. The testing process in the Docker containers sends back the test results by writing a JSON document to its standard output channel. In June 2019, R support was added to Dodona using an image derived from the **rocker/r-base** image that sets up the **runner** user and **main.sh** file expected by Dodona⁴¹. It also installs the packages required for the testing framework and the exercises so that this does not have to happen every time a student's submission is evaluated. The actual testing of R exercises is done using a custom framework loosely based on **testthat** (Wickham, 2011). During the development of the testing framework, it was found that the **testthat** framework did not provide enough information to its reporter system to send back all the fields required by Dodona to render its feedback. Right now, multiple statistics courses are developing exercises to automate the feedback for their lab classes.

Second, **PrairieLearn** is another example of a Docker-based teaching and testing platform. PrairieLearn is being developed at the University of Illinois at Urbana-Champaign (Zilles et al., 2018) and has been in extensive use across several faculties along with initial use on some other campuses. It uses Docker containers as key components, both internally for its operations (programmed mainly in Python as well as in Javascript), as well as for two reference containers providing, respectively, Python and R auto-graders. A key design decision made by PrairieLearn permits *external* grading containers to be supplied and accessed via a well-defined interface of invoking, essentially, a single script, **run.sh**. This script relies on a well-defined file layout containing JSON-based configurations, support files, exam questions, supplementary data, and student submissions. It returns per-question evaluations as JSON result files, which PrairieLearn evaluates, aggregates and records in a database. The **Data Science Programming Methods** course (Eddelbuettel, 2019) uses this via the custom **rocker-pl** container (Barbehenn and Eddelbuettel, 2019).⁴² The **rocker-pl** image extends **rocker/r-base** with the **plr** R package (Eddelbuettel and Barbehenn, 2019b) for integration into PrairieLearn testing and question evaluation, along with the actual R packages used in instruction and testing for the course in question. As automated grading of submitted student answers is close to the well-understood problem of unit testing, the **tinytest** package (van der Loo, 2019) is used for both its core features for testing as well as clean extensibility. The package **ttdo** (Eddelbuettel and Barbehenn, 2019a) utilizes the extensibility of **tinytest** to display context-sensitive colourized differences between incorrect answers and reference answers using the **diffobj** package (Gaslam, 2019). Additionally, **ttdo** addresses the issue of insufficient information collection that Dodona faced by allowing for the collection of arbitrary, test specific attributes for additional logging and feedback. The setup, described in more detail by Eddelbuettel and Barbehenn (2020), is an excellent illustration of both the versatility and flexibility offered by Docker-based approaches in teaching and testing.

⁴¹<https://github.com/dodona-edu/docker-images/blob/master/dodona-r.dockerfile>

⁴²The reference R container was unavailable at the time, and also relies on a heavier CentOS-based build so that a lighter alternative was established.

Packaging research reproducibly

Containers provide a high degree of isolation that is often desirable when attempting to capture a specific computational environment so that others can reproduce and extend a research result. Many computationally intensive research projects depend on specific versions of original and third-party software packages in diverse languages, joined together to form a pipeline through which data flows. New releases of even just a single piece of software in this pipeline can break the entire workflow, making it difficult to find the error and difficult for others to reuse existing pipelines. These breakages can make the original results irreproducible and, and the chance of a substantial disruption like this is high in a multi-year research project where key pieces of third-party software may have several major updates over the duration of the project. The classical “paper” article is insufficient to adequately communicate the knowledge behind such research projects (cf. Donoho, 2010; Marwick, 2015).

Gentleman and Lang (2007) coined the term **Research Compendium** for a dynamic document together with supporting data and code. They used the R package system (R Core Team, 1999) for the functional prototype all the way to structuring, validating, and distributing research compendia. This concept has been taken up and extended⁴³, not in the least by applying containerisation and other methods for managing computing environments—see Section [Capture and create environments](#). Containers give the researcher an isolated environment to assemble these research pipelines with specific versions of software to minimize problems with breaking changes and make workflows easier to share (cf. Boettiger, 2015; Marwick et al., 2018). Research workflows in containers are safe from contamination from other activities that occur on the researcher’s computer, for example the installation of the newest version of packages for teaching demonstrations or specific versions for evaluation of others’ works. Given the users in this scenario, i.e., often academics with limited formal software development training, templates and assistance with containers around research compendia is essential. In many fields, we see that a typical unit of research for a container is a research report or journal article, where the container holds the compendium, or self-contained set of data (or connections to data elsewhere) and code files needed to fully reproduce the article (Marwick et al., 2018). The package **rrtools** (<https://github.com/benmarwick/rrtools>) provides a template and convenience functions to apply good practices for research compendia, including a starter **Dockerfile**. Images of compendium containers can be hosted on services such as Docker Hub for convenient sharing among collaborators and others. Similarly, packages such as **containerit** and **dockerfiler** can be used to manage the **Dockerfile** to be archived with a compendium on a data repository (e.g. Zenodo, Dataverse, Figshare, OSF). A typical compendium’s **Dockerfile** will pull a rocker image fixed to a specific version of R, and install R packages from the MRAN repository to ensure the package versions are tied to a specific date, rather than the most recent version. A more extreme case is the **dynverse** project (Saelens et al.), which packages over 50 computational methods with different environments (R, Python, C++, etc.) in Docker images, which can be executed from R. **dynverse** uses a CI platform (see [Development, debugging, and testing](#)) to build Rocker-derived images, test them, and, if the tests succeed, publish them on Docker Hub.

Future researchers can download the compendium from the repository and run the included **Dockerfile** to build a new image that recreates the computational environment used to produce the original research results. If building the image fails, the human-readable instructions in a **Dockerfile** are the starting point for rebuilding the environment. When combined with CI (see [Development, debugging, and testing](#)), a research compendium set-up can enable *continuous analysis* with easier verification of reproducibility and audits trails (Beaulieu-Jones and Greene, 2017).

Further safeguarding practices are currently under development or not part of common practice yet, such as preserving images (Emsley and De Roure, 2018), storing both images and **Dockerfiles** (cf. Nüst et al., 2017), or pinning system libraries beyond the tagged base images, which may be seen as stable or dynamic depending on the applied time scale (see discussion on **debian:testing** base image in Boettiger and Eddelbuettel, 2017). A recommendation of the recent National Academies’ report on *Reproducibility and Replicability in Science* is that journals “consider ways to ensure computational reproducibility for publications that make claims based on computations” (Committee on Reproducibility and Replicability in Science, 2019). In fields such as political science and economics, journals are increasingly adopting policies that require authors to publish the code and data required to reproduce computational findings reported in published manuscripts, subject to independent verification (Jacoby et al., 2017; Vilhuber, 2019; Alvarez et al., 2018; Christian et al., 2018; Eubank, 2016; King, 1995). Problems with the computational environment, installation and availability of software dependencies are common. R is gaining popularity in these communities, such as for creating a research compendium. In a sample of 105 replication packages published by the *American Journal of Political Science* (AJPS), over 65% use R. The NSF-funded Whole Tale project, which was mentioned above, uses the Rocker Project community images with the goal of

⁴³See full literature list at <https://research-compendium.science/>.

improving the reproducibility of published research artefacts and simplifying the publication and verification process for both authors and reviewers by reducing errors and time spent specifying the environment.

Conclusions

This article is a snapshot of the R corner in a universe of applications built with a many-faced piece of software, Docker. `Dockerfiles` and Docker images are the go-to methods for collaboration between roles in an organisation, such as developers and IT operators, and between participants in the communication of knowledge, such as researchers or students. Docker has become synonymous with applying the concept of containerisation to solve challenges of reproducible environments, e.g., in research and in development & production, and of scalable deployments because it can easily move processing between machines, e.g., locally, a cloud provider's VM, another cloud provider's Container-as-a-Service. Reproducible environments, scalability & efficiency, and portability across infrastructures are the common themes behind R packages, use cases, and applications in this work.

The projects presented above show the growing number of users, developers, and real-world applications in the community and the resulting innovations. But the applications also point to the challenges of keeping up with a continuously evolving landscape. Some use cases have considerable overlap, which can be expected as a common language and understanding of good practices is still taking shape. Also, the ease with which one can create complex software systems with Docker to serve one's specific needs, such as an independent Docker image stack, leads to parallel developments. This ease-of-DIY in combination with the difficulty of reusing parts from or composing multiple `Dockerfiles` is a further reason for **fragmentation**. Instructions can be outsourced into distributable scripts and then copied into the image during build, but that makes `Dockerfiles` harder to read. Scripts added to a `Dockerfile` also add a layer of complexity and increase the risk of incomplete recipes. Despite the different image stacks presented here, the pervasiveness of Rocker images can be traced back to its maintainers and the user community valuing collaboration and shared starting points over impulses to create individual solutions. Aside from that, fragmentation may not be a bad sign but may instead be a reflection of a growing market that is able to sustain multiple related efforts. With the maturing of core building blocks, such as the Rocker suite of images, more working systems will be built, but they may simply work behind the curtains. Docker alone, as a flexible core technology, is not a feasible level of collaboration and abstraction. Instead, the use cases and applications observed in this work provide a more useful division.

Nonetheless, at least on the level of R packages some **consolidation** seems in order, e.g., to reduce the number of packages creating `Dockerfiles` from R code or controlling the Docker daemon with R code. It remains to be seen which approach to control Docker, via the Docker API as `stevedore` or via system calls as `dockyard/docker/dockr`, is more sustainable, or whether the question will be answered by the endurance of maintainers and sufficient funding. Similarly, capturing environments and their serialisation in form of a `Dockerfile` currently is happening at different levels of abstraction, and re-use of functionality seems reasonable, e.g., `liftr` could generate the environment with `containerit`, which in turn may use `dockerfiler` for low-level R objects representing a `Dockerfile` and its instructions. In this consolidation of R packages, the Rocker Project could play the role of a coordinating entity. Nonetheless, for the moment, it seems that the Rocker Project will focus on maintaining and extending its image stacks, e.g., images for GPU-based computing and artificial intelligence. Even with coding being more and more accepted as a required and achievable skill, an easier access, for example by exposing containerisation benefits via simple user interfaces in the users' IDE, could be an important next step, since currently containerisation happens more in the background for UI-based development (e.g., a `rocker/rstudio` image in the cloud). Furthermore, the maturing of the Rockerverse packages for managing containers may lead to them being adopted in situations where manual coding is currently required, e.g. in the case of `RSelenium` or `drake` (see Sections [Development](#), [debugging](#), and [testing](#) and [Processing](#) respectively). In some cases, e.g., for `analogsea`, the interaction with the Docker daemon may remain too specific to re-use first-order packages to control Docker.

New features which make complex workflows accessible and reproducible and the variety in packages connected with containerisation, even when they have overlapping features, are a signal and support for a growing user base. This growth is possibly the most important goal for the foreseeable future in the *Rockerverse*, and, just like the Rocker images have matured over years of use and millions of runs, the new ideas and prototypes will have to prove themselves. It should be noted that the dominant position is that Docker is a blessing and a curse for these goals. It might be wise to start experimenting with non-Docker containerisation tools now, e.g., R packages interfacing with other container engines, such as `podman/buildah`, or an R package for creating `Singularity` files. Such efforts might help to avoid lock-in and to design sustainable workflows based on concepts

of *containerisation*, not on their implementation in Docker. If adoption of containerisation and R continue to grow, the missing pieces for a success predominantly lie in (a) coordination and documentation of activities to reduce repeated work in favour of open collaboration, (b) the sharing of lessons learned from use cases to build common knowledge and language, and (c) a sustainable continuation and funding for development, community support, and education. A first concrete effort to work towards these missing pieces should be sustaining the structure and captured status quo from this work in the form of a *CRAN Task View on containerisation*.

Author contributions

The ordering of authors following DN and DE is alphabetical. DN conceived the article idea, initialised the formation of the writing team, wrote sections not mentioned below, and revised all sections. DE wrote the introduction and the section about containerisation and the Rocker Project, and reviewed all sections. DB wrote the section on **outsider**. GD contributed the CARD.com use case. RC contributed to the section on interfaces for Docker in R (*dynverse* and *dynwrap*). DC contributed content on Gigantum. ME contributed to the section on processing and deployment to cloud services. CF wrote paragraphs about **r-online**, **dockerflier**, **r-ci** and **r-db**. EH contributed content on **dockyard**. LK contributed content on **dockr**. SL contributed content on RStudio's usage of Docker. BM wrote the section on research compendia and made the project Binder-ready. HN & JN co-wrote the section on the T-Mobile use case. KR wrote the section about **holepunch**. NR wrote paragraphs about shared work environments and GPUs. LS & NT wrote the section on Bioconductor. PS wrote the paragraphs about CI/CD pipelines with Shinyproxy 1-Click app and OpenFaaS templates. TS & JW wrote the section on CyVerse. CvP wrote the section on the usage of Docker containers in Dodona. CW wrote the sections on Whole Tale and contributed content about publication reproducibility audits. NX contributed content on **liftr**. All authors approved the final version. This articles was collaboratively written at <https://github.com/nuest/rockerverse-paper/>. The [contributors page](#) and [discussion issues](#) provide details on the respective contributions.

Acknowledgements

DN is supported by the project Opening Reproducible Research ([o2r](#)) funded by the German Research Foundation (DFG) under project number [PE 1632/17-1](#). The funders had no role in data collection and analysis, decision to publish, or preparation of the manuscript. KR was supported in part by a grant from The Leona M. and Harry B. Helmsley Charitable Trust, award number 2016PG-BRI004. LS and NT are supported by US NIH / NHGRI awards U41HG00405 and U24HG010263. CW is supported by the Whole Tale project (<https://wholtale.org>) funded by the US National Science Foundation (NSF) under award [OAC-1541450](#). NR is supported in part by the Chan-Zuckerberg Initiative Essential Open Source Software for Science program. We would like to thank Celeste R. Brennecka from the Scientific Editing Service of the University of Münster for her editorial support.

Bibliography

- M. Alam, J. Rufino, J. Ferreira, S. H. Ahmed, N. Shah, and Y. Chen. Orchestration of Microservices for IoT Using Docker and Edge Computing. *IEEE Communications Magazine*, 56(9):118–123, Sept. 2018. ISSN 0163-6804, 1558-1896. doi: 10.1109/MCOM.2018.1701233. [p]
- J. Allaire and F. Chollet. *keras: R Interface to 'Keras'*, 2019. URL <https://CRAN.R-project.org/package=keras>. R package version 2.2.5.0. [p]
- R. M. Alvarez, E. M. Key, and L. Núñez. Research replication: Practical considerations. *PS: Political Science & Politics*, 51(2):422–426, Apr 2018. ISSN 1049-0965, 1537-5935. doi: 10.1017/S1049096517002566. [p]
- L. A. Barba. Terminologies for Reproducible Research. *arXiv:1802.03311 [cs]*, Feb. 2018. URL <http://arxiv.org/abs/1802.03311>. arXiv: 1802.03311. [p]
- A. Barbehenn and D. Eddelbuettel. *rocker-pl: Docker image for grading R in PrairieLearn*, 2019. URL <https://github.com/stat430dspm/rocker-pl>. Docker container to support STAT 430 'Data Science Programming Methods', Department of Statistics, University of Illinois at Urbana-Champaign. [p]

- B. K. Beaulieu-Jones and C. S. Greene. Reproducibility of computational workflows is automated using continuous analysis. *Nature Biotechnology*, advance online publication, Mar. 2017. ISSN 1087-0156. doi: 10.1038/nbt.3780. [p]
- H. Bengtsson. *future: Unified Parallel and Distributed Processing in R for Everyone*, 2020a. URL <https://CRAN.R-project.org/package=future>. R package version 1.16.0. [p]
- H. Bengtsson. *future: Unified Parallel and Distributed Processing in R for Everyone*, 2020b. URL <https://CRAN.R-project.org/package=future>. R package version 1.16.0. [p]
- D. Bennett, H. Hettling, D. Silvestro, R. Vos, and A. Antonelli. outsider: Install and run programs, outside of r, inside of r (under review). *Journal of Open Source Software*, 5(45):2038, 2020. doi: 10.21105/joss.02038. [p]
- D. Bernstein. Containers and cloud: From LXC to docker to kubernetes. *IEEE Cloud Computing*, 1(3):81–84, Sept. 2014. doi: 10.1109/mcc.2014.51. [p]
- R. Bivand, T. Keitt, and B. Rowlingson. *rgdal: Bindings for the 'Geospatial' Data Abstraction Library*, 2019. URL <https://CRAN.R-project.org/package=rgdal>. R package version 1.4-8. [p]
- C. Boettiger. An introduction to Docker for reproducible research, with examples from the R environment. *ACM SIGOPS Operating Systems Review*, 49(1):71–79, Jan. 2015. ISSN 01635980. doi: 10.1145/2723872.2723882. [p]
- C. Boettiger and D. Eddelbuettel. An Introduction to Rocker: Docker Containers for R. *The R Journal*, 9(2):527–536, 2017. doi: 10.32614/RJ-2017-065. [p]
- C. Boettiger, R. Lovelace, M. Howe, and J. Lamb. rocker-org/geospatial, Dec. 2019. [p]
- A. Brinckman, K. Chard, N. Gaffney, M. Hategan, M. B. Jones, K. Kowalik, S. Kulasekaran, B. Ludäscher, B. D. Mecum, J. Nabrzyski, et al. Computing environments for reproducibility: Capturing the “Whole Tale”. *Future Generation Computer Systems*, 94:854–867, 2019. doi: 10.1016/j.future.2017.12.029. [p]
- R. Cannoodt and W. Saelens. *babelwhale: Talking to 'Docker' and 'Singularity' Containers*, 2019. URL <https://CRAN.R-project.org/package=babelwhale>. R package version 1.0.1. [p]
- L. Cardozo. Faster Docker builds in Travis CI for R packages, 2018. URL <https://lecardozo.github.io/2018/03/01/automated-docker-build.html>. [p]
- S. Chamberlain, H. Wickham, and W. Chang. *analogsea: Interface to 'Digital Ocean'*, 2019. URL <https://CRAN.R-project.org/package=analogsea>. R package version 0.7.2. [p]
- Chan Zuckerberg Initiative, C. Boettiger, N. Ross, and D. Eddelbuettel. Maintaining Rocker: Sustainability for Containerized Reproducible Analyses, 2019. URL <https://chanzuckerberg.com/eoss/proposals/maintaining-rocker-sustainability-for-containerized-reproducible-analyses/>. [p]
- W. Chang, J. Cheng, J. Allaire, Y. Xie, and J. McPherson. *shiny: Web Application Framework for R*, 2019. URL <https://CRAN.R-project.org/package=shiny>. R package version 1.4.0. [p]
- K. Chard, N. Gaffney, M. B. Jones, K. Kowalik, B. Ludäscher, T. McPhillips, J. Nabrzyski, V. Stodden, I. Taylor, T. Thelen, M. J. Turk, and C. Willis. Application of BagIt-Serialized Research Object Bundles for Packaging and Re-execution of Computational Analyses. 2019a. doi: 10.5281/zenodo.3381754. To appear in 2019 IEEE 15th International Conference on e-Science (e-Science). [p]
- K. Chard, N. Gaffney, M. B. Jones, K. Kowalik, B. Ludäscher, J. Nabrzyski, V. Stodden, I. Taylor, M. J. Turk, and C. Willis. Implementing computational reproducibility in the whole tale environment. In *Proceedings of the 2nd International Workshop on Practical Reproducible Evaluation of Computer Systems*, P-RECS '19, pages 17–22, 2019b. doi: 10.1145/3322790.3330594. [p]
- T.-M. Christian, W. G. Jacoby, S. Lafferty-Hess, and T. Carsey. Operationalizing the replication standard. *International Journal of Digital Curation*, 13(1), 2018. doi: 10.2218/ijdc.v13i1.555. [p]
- Committee on Reproducibility and Replicability in Science. *Reproducibility and Replicability in Science*. National Academies Press, 2019. ISBN 978-0-309-48616-3. doi: 10.17226/25303. [p]
- Datadog. 8 surprising facts about real Docker adoption, June 2018. URL <https://www.datadoghq.com/docker-adoption/>. [p]

- U. K. Devisetty, K. Kennedy, P. Sarando, N. Merchant, and E. Lyons. Bringing your tools to CyVerse Discovery Environment using Docker. *F1000Research*, 5:1442, Dec. 2016. ISSN 2046-1402. doi: 10.12688/f1000research.8935.3. [p]
- D. Donoho. 50 Years of Data Science. *Journal of Computational and Graphical Statistics*, 26(4): 745–766, Oct. 2017. ISSN 1061-8600. doi: 10.1080/10618600.2017.1384734. [p]
- D. L. Donoho. An invitation to reproducible computational research. *Biostatistics*, 11(3):385–388, July 2010. ISSN 1465-4644. doi: 10.1093/biostatistics/kxq028. [p]
- A. Eckert. Building and testing R packages with latest R-Devel, Feb. 2018. URL <https://alexandereckert.com/post/testing-r-packages-with-latest-r-devel/>. [p]
- D. Eddelbuettel. *sanitizers: C/C++ source code to trigger Address and Undefined Behaviour Sanitizers*, 2014. URL <https://CRAN.R-project.org/package=sanitizers>. R package version 0.1.0. [p]
- D. Eddelbuettel. *STAT430: Data Science Programming Methods*, 2019. URL <https://stat430.com>. Fourth and fifth year topics course, Department of Statistics, University of Illinois at Urbana-Champaign. [p]
- D. Eddelbuettel and A. Barbehenn. *ttdo: Extend 'tinytest' with 'diffobj'*, 2019a. URL <https://CRAN.R-project.org/package=ttdo>. R package version 0.0.4. [p]
- D. Eddelbuettel and A. Barbehenn. *plr: Utility Functions for 'PrairieLearn' and R*, 2019b. URL <https://github.com/stat430dspm/plr>. R package supporting Docker for STAT 430 'Data Science Programming Methods', Department of Statistics, University of Illinois at Urbana-Champaign. [p]
- D. Eddelbuettel and A. Barbehenn. An R Autograder for PrairieLearn, 2020. URL <http://arxiv.org/abs/2003.06500>. [p]
- D. Eddelbuettel and R. Koenker. Debugging with Docker and Rocker –A Concrete Example helping on macOS, Aug. 2019. URL <http://dirk.eddelbuettel.com/blog/2019/08/05/>. [p]
- M. Edmondson. R on Kubernetes - serverless Shiny, R APIs and scheduled scripts, May 2018. URL <https://code.markedmondson.me/r-on-kubernetes-serverless-shiny-r-apis-and-scheduled-scripts/>. [p]
- M. Edmondson. *googleComputeEngineR: R Interface with Google Compute Engine*, 2019. URL <https://CRAN.R-project.org/package=googleComputeEngineR>. R package version 0.3.0. [p]
- M. Edmondson. *googleCloudRunner: R Scripts in the Google Cloud via Cloud Run, Cloud Build and Cloud Scheduler*, 2020. URL <https://CRAN.R-project.org/package=googleCloudRunner>. R package version 0.1.1. [p]
- I. Emsley and D. De Roure. A Framework for the Preservation of a Docker Container | International Journal of Digital Curation. *International Journal of Digital Curation*, 12(2), Apr. 2018. doi: 10.2218/ijdc.v12i2.509. [p]
- N. Eubank. Lessons from a decade of replications at the Quarterly Journal of Political Science. *PS: Political Science & Politics*, 49(2):273–276, Apr 2016. ISSN 1049-0965, 1537-5935. doi: 10.1017/S1049096516000196. [p]
- C. Fay. *dockerfiler: Easy Dockerfile Creation from R*, 2019. URL <https://CRAN.R-project.org/package=dockerfiler>. R package version 0.1.3. [p]
- W. Felter, A. Ferreira, R. Rajamony, and J. Rubio. An updated performance comparison of virtual machines and Linux containers. In *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 171–172, Mar. 2015. doi: 10.1109/ISPASS.2015.7095802. [p]
- R. FitzJohn. *stevedore: Docker Client*, 2020. URL <https://CRAN.R-project.org/package=stevedore>. R package version 0.9.3. [p]
- B. Gaslam. *diffobj: Diffs for R Objects*, 2019. URL <https://CRAN.R-project.org/package=diffobj>. R package version 0.2.3. [p]
- GDAL/OGR contributors. *GDAL/OGR Geospatial Data Abstraction software Library*. Open Source Geospatial Foundation, 2019. URL <https://gdal.org>. [p]

- R. Gentleman and D. T. Lang. Statistical Analyses and Reproducible Research. *Journal of Computational and Graphical Statistics*, 16(1):1–23, Mar. 2007. ISSN 1061-8600. doi: 10.1198/106186007X178663. [p]
- R. C. Gentleman, V. J. Carey, D. M. Bates, B. Bolstad, M. Dettling, S. Dudoit, B. Ellis, L. Gautier, Y. Ge, J. Gentry, K. Hornik, T. Hothorn, W. Huber, S. Iacus, R. Irizarry, F. Leisch, C. Li, M. Maechler, A. J. Rossini, G. Sawitzki, C. Smith, G. Smyth, L. Tierney, J. Y. Yang, and J. Zhang. Bioconductor: open software development for computational biology and bioinformatics. *Genome Biology*, 5(10):R80, Sept. 2004. ISSN 1474-760X. doi: 10.1186/gb-2004-5-10-r80. [p]
- J.-P. S. Glouzon, J.-P. Perreault, and S. Wang. Structureexplor: a platform for the exploration of structural features of RNA secondary structures. *Bioinformatics*, 33(19):3117–3120, Oct. 2017. ISSN 1367-4803. doi: 10.1093/bioinformatics/btx323. [p]
- P. Grosjean. *SciViews-R: A GUI API for R*. UMONS, MONS, Belgium, 2019. URL <http://www.sciviews.org/SciViews-R>. [p]
- V. Guyader, C. Fay, S. Rochette, and C. Girard. *golem: A Framework for Robust Shiny Applications*, 2019. URL <https://CRAN.R-project.org/package=golem>. R package version 0.1. [p]
- J. Harrison. *RSelenium: R Bindings for 'Selenium WebDriver'*, 2019. URL <https://CRAN.R-project.org/package=RSelenium>. R package version 1.7.5. [p]
- N. Haydel, G. Madey, S. Gesing, A. Dakkak, S. G. de Gonzalo, I. Taylor, and W.-m. W. Hwu. Enhancing the Usability and Utilization of Accelerated Architectures via Docker. In *Proceedings of the 8th International Conference on Utility and Cloud Computing*, UCC ’15, pages 361–367. IEEE Press, 2015. ISBN 978-0-7695-5697-0. URL <http://dl.acm.org/citation.cfm?id=3233397.3233456>. [p]
- K. Hornik, U. Ligges, and A. Zeileis. Changes on cran. *The R Journal*, 11(1):438–441, June 2019. URL <http://journal.r-project.org/archive/2019-1/cran.pdf>. [p]
- W. G. Jacoby, S. Lafferty-Hess, and T.-M. Christian. Should journals be responsible for reproducibility? *Inside Higher Ed*, Jul 2017. URL <https://www.insidehighered.com/blogs/rethinking-research/should-journals-be-responsible-reproducibility>. [p]
- P. Jupyter. Jupyter Docker Stacks —docker-stacks latest documentation, 2018. URL <https://jupyter-docker-stacks.readthedocs.io/en/latest/>. [p]
- P. Jupyter, M. Bussonnier, J. Forde, J. Freeman, B. Granger, T. Head, C. Holdgraf, K. Kelley, G. Nalvarte, A. Osherooff, M. Pacer, Y. Panda, F. Perez, B. Ragan-Kelley, and C. Willing. Binder 2.0 - Reproducible, interactive, sharable environments for science at scale. *Proceedings of the 17th Python in Science Conference*, pages 113–120, 2018. doi: 10.25080/Majora-4af1f417-011. [p]
- G. King. Replication, replication. *PS: Political Science & Politics*, 28(3):444–452, Sep 1995. doi: 10.2307/420301. [p]
- L. Kjeldgaard. *dockr: Creation of Lightweight Docker Images for Your Packages*, 2019a. URL <https://CRAN.R-project.org/package=dockr>. R package version 0.8.6. [p]
- L. Kjeldgaard. ’dockr’: easy containerization for R - pRopaganda by smaakagen, Dec. 2019b. URL <http://smaakage85.netlify.com/2019/12/21/dockr-easy-containerization-for-r/>. [p]
- G. M. Kurtzer, V. Sochat, and M. W. Bauer. Singularity: Scientific containers for mobility of compute. *PLOS ONE*, 12(5):e0177459, May 2017. ISSN 1932-6203. doi: 10.1371/journal.pone.0177459. [p]
- W. M. Landau. The drake r package: a pipeline toolkit for reproducibility and high-performance computing. *Journal of Open Source Software*, 3(21), 2018. [p]
- M. Lang, B. Bischl, and D. Surmann. batchtools: Tools for r to work on batch systems. *The Journal of Open Source Software*, 2(10):135, 2 2017. ISSN 2475-9066. doi: 10.21105/joss.00135. [p]
- B. Marwick. How computers broke science –and what we can do to fix it, Nov. 2015. URL <http://theconversation.com/how-computers-broke-science-and-what-we-can-do-to-fix-it-49938>. [p]
- B. Marwick. Research compendium for the 1989 excavations at Madjedbebe rockshelter, NT, Australia, July 2017. [p]

- B. Marwick, C. Boettiger, and L. Mullen. Packaging Data Analytical Work Reproducibly Using R (and Friends). *The American Statistician*, 72(1):80–88, Jan. 2018. ISSN 0003-1305. doi: 10.1080/00031305.2017.1375986. [p]
- T. McPhillips, C. Willis, M. Gryk, S. Nunez-Corrales, and B. Ludäscher. Reproducibility by Other Means: Transparent Research Objects. 2019. doi: 10.5281/zenodo.3382423. To appear in 2019 IEEE 15th International Conference on e-Science (e-Science). [p]
- B. Mecum, M. B. Jones, D. Vieglais, and C. Willis. Preserving reproducibility: Provenance and executable containers in dataone data packages. In *2018 IEEE 14th International Conference on e-Science (e-Science)*, pages 45–49. IEEE, 2018. doi: 10.1109/eScience.2018.00019. [p]
- N. Merchant, E. Lyons, S. Goff, M. Vaughn, D. Ware, D. Micklos, and P. Antin. The iPlant Collaborative: Cyberinfrastructure for Enabling Data to Discovery for the Life Sciences. *PLOS Biology*, 14(1):e1002342, Jan. 2016. ISSN 1545-7885. doi: 10.1371/journal.pbio.1002342. [p]
- Microsoft. CRAN Time Machine - MRAN, 2019a. URL <https://mran.microsoft.com/timemachine>. [p]
- Microsoft. Linux Containers on Windows, Sept. 2019b. URL <https://docs.microsoft.com/en-us/virtualization/windowscontainers/deploy-containers/linux-containers>. [p]
- M. Morgan. *BiocManager: Access the Bioconductor Project Package Repository*, 2019. URL <https://CRAN.R-project.org/package=BiocManager>. R package version 1.30.10. [p]
- S. Muñoz. The history of Docker's climb in the container management market, June 2019. URL <https://searchservervirtualization.techtarget.com/feature/The-history-of-Dockers-climb-in-the-container-management-market>. [p]
- J. Nolis and J. Werdell. Small data, big value, Dec. 2019. URL <https://medium.com/tmobile-tech/small-data-big-value-f783ceca4fdb>. [p]
- D. Nüst and M. Hinz. containerit: Generating Dockerfiles for reproducible research with R. *Journal of Open Source Software*, 4(40):1603, Aug. 2019. ISSN 2475-9066. doi: 10.21105/joss.01603. URL <https://joss.theoj.org/papers/10.21105/joss.01603>. [p]
- D. Nüst, M. Konkol, E. Pebesma, C. Kray, M. Schutzeichel, H. Przybytzin, and J. Lorenz. Opening the Publication Process with Executable Research Compendia. *D-Lib Magazine*, 23(1/2), Jan. 2017. ISSN 1082-9873. doi: 10.1045/january2017-nuest. [p]
- OCI. Open Containers Initiative - About, 2019. URL <https://www.opencontainers.org/about>. [p]
- H. Ooi. *AzureContainers: Interface to 'Container Instances', 'Docker Registry' and 'Kubernetes' in 'Azure'*, 2019. URL <https://CRAN.R-project.org/package=AzureContainers>. R package version 1.2.0. [p]
- H. Ooi, A. de Vries, and Microsoft. *checkpoint: Install Packages from Snapshots on the Checkpoint Server for Reproducibility*, 2020. URL <https://CRAN.R-project.org/package=checkpoint>. R package version 0.4.9. [p]
- J. Ooms. OpenCPU - Why Use Docker with R? A DevOps Perspective, Oct. 2017. URL <https://www.opencpu.org/posts/opencpu-with-docker/>. [p]
- J. Ooms. *sys: Powerful and Reliable Tools for Running System Commands in R*, 2019. URL <https://CRAN.R-project.org/package=sys>. R package version 3.3. [p]
- E. Pebesma. Simple Features for R: Standardized Support for Spatial Vector Data. *The R Journal*, 10(1):439–446, 2018. doi: 10.32614/RJ-2018-009. [p]
- R Core Team. 1999. URL <https://cran.r-project.org/doc/manuals/r-devel/R-exts.html>. [p]
- R-hub project. R-hub Docs, 2019. URL <https://docs.r-hub.io/>. [p]
- R Special Interest Group on Databases (R-SIG-DB), H. Wickham, and K. Müller. *DBI: R Database Interface*, 2019. URL <https://CRAN.R-project.org/package=DBI>. R package version 1.1.0. [p]
- RStudio Support. Running rstudio server with a proxy, Jan. 2020. URL <https://support.rstudio.com/hc/en-us/articles/200552326-Running-RStudio-Server-with-a-Proxy>. [p]

- W. Saelens, R. Cannoodt, H. Todorov, and Y. Saeys. A comparison of single-cell trajectory inference methods. 37. ISSN 15461696. doi: 10.1038/s41587-019-0071-9. [p]
- L. Savini, L. Candeloro, S. Perticara, and A. Conte. EpiExploreR: A Shiny Web Application for the Analysis of Animal Disease Data. *Microorganisms*, 7(12):680, Dec. 2019. doi: 10.3390/microorganisms7120680. [p]
- J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, Jan. 2015. ISSN 0893-6080. doi: 10.1016/j.neunet.2014.09.003. [p]
- T-Mobile, J. Nolis, and H. Nolis. Enterprise Web Services with Neural Networks Using R and TensorFlow, Nov. 2018. URL <https://opensource.t-mobile.com/blog/posts/r-tensorflow-api/>. [p]
- Trestle Technology, LLC. *plumber: An API Generator for R*, 2018. URL <https://CRAN.R-project.org/package=plumber>. R package version 0.4.6. [p]
- S. Urbanek. *Rserve: Binary R server*, 2019. URL <https://CRAN.R-project.org/package=Rserve>. R package version 1.7-3.1. [p]
- K. Ushey. Using renv with Docker, 2019. URL <https://rstudio.github.io/renv/articles/docker.html>. [p]
- K. Ushey. *renv: Project Environments*, 2020. URL <https://CRAN.R-project.org/package=renv>. R package version 0.9.3. [p]
- K. Ushey, J. Allaire, and Y. Tang. *reticulate: Interface to 'Python'*, 2019. URL <https://CRAN.R-project.org/package=reticulate>. R package version 1.14. [p]
- M. van der Loo. *tinytest: Lightweight and Feature Complete Unit Testing Framework*, 2019. URL <https://CRAN.R-project.org/package=tinytest>. R package version 1.1.0. [p]
- L. Vilhuber. Report by the AEA Data Editor. *AEA Papers and Proceedings*, 109:718–729, May 2019. ISSN 2574-0768. doi: 10.1257/pandp.109.718. [p]
- H. Wickham. testthat: Get started with testing. *The R Journal*, 3:5–10, 2011. URL https://journal.r-project.org/archive/2011-1/RJournal_2011-1_Wickham.pdf. [p]
- H. Wickham and E. Ruiz. *dbplyr: A 'dplyr' Back End for Databases*, 2019. URL <https://CRAN.R-project.org/package=dbplyr>. R package version 1.4.2. [p]
- H. Wickham, M. Averick, J. Bryan, W. Chang, L. McGowan, R. François, G. Grolemund, A. Hayes, L. Henry, J. Hester, M. Kuhn, T. Pedersen, E. Miller, S. Bache, K. Müller, J. Ooms, D. Robinson, D. Seidel, V. Spinu, K. Takahashi, D. Vaughan, C. Wilke, K. Woo, and H. Yutani. Welcome to the Tidyverse. *Journal of Open Source Software*, 4(43):1686, Nov. 2019. ISSN 2475-9066. doi: 10.21105/joss.01686. [p]
- H. Wickham, R. François, L. Henry, and K. Müller. *dplyr: A Grammar of Data Manipulation*, 2020. URL <https://CRAN.R-project.org/package=dplyr>. R package version 0.8.4. [p]
- Wikipedia contributors. APT (software), Feb. 2020a. URL [https://en.wikipedia.org/w/index.php?title=APT_\(software\)&oldid=939802209](https://en.wikipedia.org/w/index.php?title=APT_(software)&oldid=939802209). Page Version ID: 939802209. [p]
- Wikipedia contributors. OS-level virtualization, Jan. 2020b. URL https://en.wikipedia.org/w/index.php?title=OS-level_virtualization&oldid=935110975. Page Version ID: 935110975. [p]
- N. Xiao. *liftr: Containerize R Markdown Documents for Continuous Reproducibility*, 2019. URL <https://CRAN.R-project.org/package=liftr>. R package version 0.9.2. [p]
- Y. Xie, J. J. Allaire, and G. Grolemund. *R Markdown: The Definitive Guide*. Chapman and Hall/CRC, 2018. [p]
- H. Ye. Docker Setup for R package Development, 2019. URL <https://haoye.us/post/2019-10-10-docker-for-r-package-development/>. [p]
- C. Zilles, M. West, D. Musselman, and T. Bretl. Making testing less trying: Lessons learned from operating a computer-based testing facility. In *Proceedings of the 2018 Frontiers in Education Conference (FIE 2018)*, 2018. URL <http://lagrange.mechse.illinois.edu/pubs/ZiWeMuBr2018/ZiWeMuBr2018.pdf>. [p]

M. Çetinkaya Rundel and C. Rundel. Infrastructure and Tools for Teaching Computing Throughout the Statistical Curriculum. *The American Statistician*, 72(1):58–65, Jan. 2018. ISSN 0003-1305.
doi: 10.1080/00031305.2017.1397549. [p]

Daniel Nüst
University of Münster
Institute for Geoinformatics
Heisenbergstr. 2
48149 Münster, Germany
 [0000-0002-0024-5046](#)
daniel.nuest@uni-muenster.de

Dirk Eddelbuettel
University of Illinois at Urbana-Champaign
Department of Statistics
Illini Hall, 725 S Wright St
Champaign, IL 61820, USA
 [0000-0001-6419-907X](#)
dirk@eddelbuettel.com

Dom Bennett
Gothenburg Global Biodiversity Centre, Sweden
Carl Skottsbergs gata 22B
413 19 Göteborg, Sweden
 [0000-0003-2722-1359](#)
dominic.john.bennett@gmail.com

Robrecht Cannoodt
Ghent University
Data Mining and Modelling for Biomedicine group
VIB Center for Inflammation Research
Technologiepark 71
9052 Ghent, Belgium
 [0000-0003-3641-729X](#)
robrecht@cannoodt.dev

Dav Clark
Gigantum, Inc.
1140 3rd Street NE
Washington, D.C. 20002, USA
 [0000-0002-3982-4416](#)
dav@gigantum.com

Gergely Darócz
 [0000-0003-3149-8537](#)
daroczig@rapporter.net

Mark Edmondson
IHH Nordic A/S, Google Developer Expert for Google Cloud Platform
Artillerivej 86
2300 København S, Denmark
 [0000-0002-8434-3881](#)
mark@markedmondson.me

Colin Fay
ThinkR
50 rue Arthur Rimbaud
93300 Aubervilliers, France
 [0000-0001-7343-1846](#)
contact@colinfay.me

Ellis Hughes
Fred Hutchinson Cancer Research Center
Vaccine and Infectious Disease
1100 Fairview Ave. N., P.O. Box 19024
Seattle, WA 98109-1024, USA
e hhughes@fredhutch.org

Lars Kjeldgaard
Danish Tax Authorities
Oestbanegade 123
2100, Koebenhavn Oe
lars_kjeldgaard@hotmail.com

Sean Lopp
RStudio, Inc
250 Northern Ave
Boston, MA 02210, USA
sean@rstudio.com

Ben Marwick
University of Washington
Department of Anthropology
Denny Hall 230, Spokane Ln
Seattle, WA 98105, USA
 0000-0001-7879-4531
bmarwick@uw.edu

Heather Nolis
T-Mobile
12920 Se 38th St.
Bellevue, WA, 98006, USA
heather.wensler1@t-mobile.com

Jacqueline Nolis
Nolis, LLC
Seattle, WA, USA
 0000-0001-9354-6501
jacqueline@nolisllc.com

Hong Ooi
Microsoft
Level 5, 4 Freshwater Place
Southbank, VIC 3006, Australia
hongooi@microsoft.com

Karthik Ram
Berkeley Institute for Data Science
University of California
Berkeley, CA 94720, USA
 0000-0002-0233-1757
karthik.ram@berkeley.edu

Noam Ross
EcoHealth Alliance
460 W 34th St., Ste. 1701
New York, NY 10001, USA
 0000-0002-2136-0000
ross@ecohealthalliance.org

Lori Shepherd
Roswell Park Comprehensive Cancer Center
Elm & Carlton Streets

Buffalo, NY, 14263, USA
DOI 0000-0002-5910-4010
lori.shepherd@roswellpark.org

Péter Sólymos
Analythium Solutions
#258 150 Chippewa Road
Sherwood Park, AB, T8A 6A2, Canada
DOI 0000-0001-7337-1740
peter@analythium.io

Tyson Lee Swetnam
University of Arizona
1657 E Helen St.
Tucson, AZ, 85721, USA
DOI 0000-0002-6639-7181
tswetnam@arizona.edu

Nitesh Turaga
Roswell Park Comprehensive Cancer Center
Elm & Carlton Streets
Buffalo, NY, 14263, USA
DOI 0000-0002-0224-9817
nitesh.turaga@roswellpark.org

Charlotte Van Petegem
Ghent University
Department WE02
Krijgslaan 281, S9
9000 Gent, Belgium
DOI 0000-0003-0779-4897
charlotte.vanpetegem@ugent.be

Jason Williams
Cold Spring Harbor Laboratory
1 Bungtown Rd.
Cold Spring Harbor, NY, 11724, USA
DOI 0000-0003-3049-2010
williams@cshl.edu

Craig Willis
University of Illinois at Urbana-Champaign
501 E. Daniel St.
Champaign, IL 61820, USA
DOI 0000-0002-6148-7196
willis8@illinois.edu

Nan Xiao
Seven Bridges Genomics
529 Main St, Suite 6610
Charlestown, MA 02129, USA
DOI 0000-0002-0250-5673
me@nanx.me

S, R, and Data Science

by John M. Chambers

Abstract Data science is increasingly important and challenging. It requires computational tools and programming environments that handle big data and difficult computations, while supporting creative, high-quality analysis. The R language and related software play a major role in computing for data science. R is featured in most programs for training in the field. R packages provide tools for a wide range of purposes and users. The description of a new technique, particularly from research in statistics, is frequently accompanied by an R package, greatly increasing the usefulness of the description.

The history of R makes clear its connection to data science. R was consciously designed to replicate in open-source software the contents of the S software. S in turn was written by data analysis researchers at Bell Labs as part of the computing environment for research in data analysis and collaborations to apply that research, rather than as a separate project to create a programming language. The features of S and the design decisions made for it need to be understood in this broader context of supporting effective data analysis (which would now be called data science). These characteristics were all transferred to R and remain central to its effectiveness. Thus, R can be viewed as based historically on a domain-specific language for the domain of data science.

Note to R Journal readers:

The following paper was published online in the History of Programming Languages (HOPL), Volume 4, in June 2020 (DOI 10.1145/3386334). The content seems likely to be of interest to many R Journal readers, and since HOPL is plausibly not typical reading for data scientists, the editors of the R Journal have kindly offered to republish the paper here. This is possible thanks also to the enlightened policy of the ACM, providing for open distribution through the chosen copyright declaration.

Introduction

R has become a widely used medium for the practice of technically advanced data science; most importantly, a medium in which new applications and new ideas in the practice of data science are very often shared throughout the worldwide community.

The language, data structure and functional capabilities of R, as they were implemented in the late 1990s, were modelled on the S software from Bell Labs, supplemented by some new ideas, reflecting developments in programming language design during this period. To create a free, open-source language based on S, R's original authors, Ihaka and Gentleman (1996), were joined by an international group of volunteers, subsequently known as *R Core Ihaka* (1998).

The necessary definition of S, independent of its proprietary implementation, was taken from two books: Becker et al. (1988) for the general features and Chambers and Hastie (1992) for the fitting and analysis of statistical models plus some extensions to the software, notably to classes and methods. In the R community, these books are nearly always referred to as the *blue book* and the *white book*, respectively, from their covers.

The S software was distinguished from many programming language designs in being motivated by a relatively specific scientific goal; namely, to support research in data analysis at Bell Labs and applications to challenging problems. The goal of the language was to provide interactive analysis using the best current techniques Becker and Chambers (1984) and also a programming interface to software implementing new techniques Becker and Chambers (1985). These goals influenced distinctive characteristics of S: data structures designed for data analysis rather than built up from basic types; interfaces to other software as part of the language design.

Data analysis as practiced at Bell Labs is recognized as the precursor of what would now be described as “data science” Donoho (2017). Subsequent versions of S and of R have retained and extended a design focussed on the needs of data science, so that R can be viewed as a *domain-specific* language for the domain of data science.

This perspective helps to understand both the history and many of the design choices leading to R.

1965–1985: Bell Labs, Data Science and Computing

For a long period in the 20th century, particularly for the three or four decades following the second world war, notable scientific and technological advances came from the research area of AT&T's Bell Telephone Laboratories (aka "Bell Labs").

The most famous of these, the invention of the transistor, was a critical step towards the digital and miniaturization revolution that continues to overwhelm us. Other advances were also key, notably information theory, coding and digital techniques for communication.

It was noted at the time, and even more since then, that the productivity and originality of much Bell Labs work seemed to derive from an organization and research atmosphere not easily found elsewhere. A non-technical account that nevertheless conveys much of the research management style is given in the book by Jon Gertner [Gertner \(2013\)](#).

The management philosophy of Bell Labs research was to hire bright and self-motivated individuals and give them the freedom to come up with their own ideas. At the same time, there was a belief that some of these ideas would be fundamentally valuable for communication, and so for the parent company. Whoever came up with such ideas could expect to be rewarded financially (moderately) and with recognition. Along with ideas leading to the transistor and communication theory, this research environment nurtured an approach to what can now be called data science.

Data Science and Data Analysis

Techniques, applications and teaching for data science have drawn much attention recently, and for good reason. Essentially all branches of science face challenges in studying important questions due to the quantity, complexity or questionable nature of the data.

The term "data science" is relatively recent and is used somewhat loosely at times; we will assume a simple but strict definition:

Data science consists of techniques and their application to derive and communicate scientifically valid inferences and predictions based on relevant data.

(In particular, just the use of "big data" does not qualify the results as data science.)

Although the popularity of the term lay decades in the future, research in data analysis at Bell Labs during the design and evolution of S is widely recognized as the precursor to data science. The fundamental inspiration for this research came originally from John Tukey. The historical summary in the paper "50 Years of Data Science" [Donoho \(2017\)](#) cites his "Future of Data Analysis" paper [Tukey \(1962\)](#) as a point of origin for data science. Tukey's championing of data analysis continued through many later contributions, including the book "Exploratory Data Analysis" [Tukey \(1977\)](#) and beyond. His career was divided between Bell Labs and Princeton University (along with many other activities). Tukey was an enormous influence, not to say inspiration, at Bell Labs.

Bell Labs statistics research was housed in the "Statistics and Data Analysis Research" department, surely the only group of research statisticians with "Data Analysis" in its title at that time. Interesting and potentially rewarding projects could range from the essentially theoretical (though usually with an implication of future application) through more data-analytic methods (for example, data or model visualization [Wilk and Gnanadesikan \(1968\)](#)) to collaborative projects with other groups at Bell Labs and AT&T to obtain insights from particular sources of data.

Data analysis at Bell Labs did not avoid "big data" by the standards of the time (usually meaning one or a few reels of magnetic tape); on the contrary, the challenge of doing analysis in this context was often central to a particularly interesting and important collaboration. For example, rain gauge experiments in the 1960s studying the effect of rainfall on errors in microwave transmission generated data running to several million observations, requiring some "big data" techniques for visualization and summaries ([Freeny and Gabbe \(1969\)](#) and [Jaeckel and Gabbe \(1974\)](#)).

Overall, the combination of opportunities and responsibilities gave data analysis at Bell Labs much of the flavor associated with contemporary discussions of data science: large datasets; iterative, probing analysis including visualization; problems of practical importance and, as a result, challenging computations. Data analysis that was useful and applicable to sizable datasets required advanced computational techniques for the time and good software to implement them.

By the time I first arrived at Bell Labs as a graduate student intern in 1964, advances in computation were already recognized as important for data analysis. S came after more than a decade of involvement in statistical computing.

Before S

The decade or so beginning in the mid-1960s was a determining period for scientific computing, and in particular for computations involving significant amounts of data. Hardware, software and algorithms all broke decisively with the first generation of computing and its emphasis on the physical elements of the computer and individual machine instructions.

By the middle of the 1960s, Bell Labs was involved in a project to create the Multics system, jointly with MIT and General Electric [Corbató and Vyssotsky \(1965\)](#). This was a pioneering and ambitious effort to implement a multi-process, multi-user operating system on a large scale.

The combination of data analysis research and large-scale applications had sensitized management at Bell Labs to the relevance of statistical computing. The first planning for a statistical system began in 1965 (with John Tukey participating in our initial meeting). The system was to be built around the PL-1 language and was predicated on Multics as the operating system environment. The proposed name was BLISS, for Bell Labs Interactive Statistical System (although the “I” was sometimes interpreted as Interim).

Bell Labs dropped out of the Multics project in 1966, for practical reasons. The Murray Hill location of Bell Labs had scheduled the replacement of its IBM 7094 system with hardware from General Electric, with the intention of running Multics. The IBM equipment was promised to a new Bell Labs location in Indian Hill, Illinois.

Not surprisingly, the implementation of Multics took considerably longer than had been predicted. It was clearly not going to be generally usable when the hardware transfer occurred. With the new hardware at Murray Hill but no Multics, most of the Research area of Bell Labs was left with a computer system from a much less experienced company than IBM, an operating system not the one desired (and neither understood by us nor bug-free) and less of a software base than the IBM, let alone what had been expected with Multics.

For data analysis, the immediate computational strategy was largely a rescue mission, to provide a capability to manage and analyze data with the scale and reliability we required, and with access to the numerical capabilities necessary for the analysis. The facility took the form of a subroutine library, callable from Fortran and largely implemented in that language.

The BLISS project was dropped, inevitably since it not only assumed the Multics operating system but was to have been an extension of the PL-1 language planned for Multics but not available otherwise. Of the small group involved in planning BLISS, only I would still be at Bell Labs and involved in statistical computing when work on § began. Only a little prototyping had been completed on BLISS, none of which was relevant to later work.

It would be a decade before the first version of S was implemented to provide an interactive environment for flexible analysis applied to a wide range of data. However, that decade was by no means static. Research in data analysis and collaborative projects continued actively. Providing state-of-the-art computing focused initially on relatively specific methods, implemented as subroutines to be called from Fortran and organized in a subroutine library.

When we came back to create an interactive environment, the computing facilities incorporated in the library had expanded enormously, both for Fortran’s traditional domain of numerical computation and for the other areas that make up data science. Computations for linear algebra, optimization, function approximation, random number generation and data manipulation (e.g., sorting and searching) were among those largely revolutionized by new computer-oriented techniques. These were often implemented in publicly available sources, such as published algorithm sections. The community involved in using and testing these algorithms grew rapidly as well.

Additional areas had been advanced locally, including two of relevance for data science: visualization and data management. A flexible structure for computer graphics in support of data analysis was implemented through Fortran subroutines. This software, referred to as GR-Z [Becker and Chambers \(1976\)](#), provided a structure for graphics later adopted and extended in S and therefore in R.

I wrote (but never described externally) some data management software that supported a general model of data structures defined hierarchically with named components, starting from vectors and scalars of some basic types. The structures were self-defining and extensible. Lengths of components could be queried and modified. Users could create arbitrary new types of structure. With some modification, this software provided the initial implementation for data structures in S.

First Version of S

The first meetings to plan for an interactive statistical system took place in 1976. At this time, powerful software for data analysis existed in the form of an extensive subroutine library. Interactive

use of this software was becoming possible through time-shared terminals.

However, the software in the library could only be used by writing a complete control script, also a Fortran program, that managed the data, carried out the analysis and produced some informative output to be viewed later. This would be run as a “job”, with control information included, in a format little changed from the days when user card decks were submitted for operators to run.

The details needed were sufficiently tricky and extensive to be outside the skills of the principal investigators, particularly for analysis involving serious data in terms of size or complexity. As a result, most data analysis involved a team including programmers. Decisions about new analysis required communicating the ideas to the programming staff, adding to the delay and discouraging repeated changes.

When Rick Becker joined us, he had experience with an interactive system at the National Bureau of Economic Research that, while much less extensive or general than our software, provided truly interactive analysis. Rick and I became convinced that we could build a system combining convenient interactive use with access to the full power of the Fortran-based library. We proposed to create an “interactive environment” (as the title of the first book on S [Becker and Chambers \(1984\)](#) referred to it) that continued to support data analysis but gave the analyst a convenient, direct interface. To achieve this goal, the new software had to provide three features:

1. *Convenience*: compact, straightforward expression for the analysis, with § handling details such as managing the data and providing graphical or formatted output.
2. *Completeness*: the extensive range of summaries, modeling and visualization provided by the Fortran library had to be available;
3. *Extensibility*: we were a data analysis *research* community, so new techniques would need to be available from §.

A fourth requirement was that this be implementable with a relatively modest programming effort; essentially, the two eventual authors with help from various colleagues.

The approach that succeeded in satisfying all the requirements was to build the system around an *interface* to Fortran. From the start of the project, our design was based on writing specialized code to incorporate individual Fortran subroutines into S by writing a specialized interface function for each of them.

Typically for Bell Labs research, we felt free to start the project without any formal approval. The first meeting, on May 5 1976, involved about five people as I recall, none of them management. We presented some ideas and preliminary software. Figure 1 is the first “visual” of the first talk. The upper half of the figure illustrates the concept of the interface implemented in Fortran: a Fortran subroutine (`XABC`, the rectangle) interpreting the user’s interactive expression, passed in as an S object by the argument `INSTR`. Eventually the interface calls the Fortran algorithm (`ABC`, the circle). The result is returned to the interactive user as an S object (through the argument `OUTSTR`, since `XABC` is a subroutine rather than a function). The lower half of the figure sketches the implementation of the objects for the user’s call and the value returned.

The details of how all this would work were somewhat unclear initially, of course, but the design suggested by the Figure is broadly consistent with the implementation over the next couple of years. By the time S was distributed generally, the interface mechanism, as well as being the implementation for most of the system, was provided to users as an interface *language*, [Becker and Chambers \(1985\)](#), the chief mechanism for extensibility. The interface language would be pre-compiled into Fortran.

The essential programming unit was a function definition. Each function would be available in S with the name and arguments defined in the interface code. The arguments in a call to the function would be objects in S. The function would return an S object as its value. The body of the function could contain any Fortran code but typically it would call a subroutine, not dependent on S, to do the actual analysis, visualization or other computation. In particular, this made essentially all the code in our library available for incorporating into S.

The interface language also had some facilities for creating and manipulating certain S objects, mainly vectors, matrices and lists with named components that were themselves S objects. The interface language mapped objects or their components into Fortran arrays, usually numeric. The language also had built-in accessor functions to provide necessary scalar information for Fortran, such as the length of a vector or the dimensions of a matrix.

In addition to the interface routines, top-level code parsed the user language, loaded the code for individual functions and evaluated the call to the function.

The user language consisted of expressions, generally C-style, plus a few extra operators and minus declarations. The essence of the system was in the functions, several hundred of which were

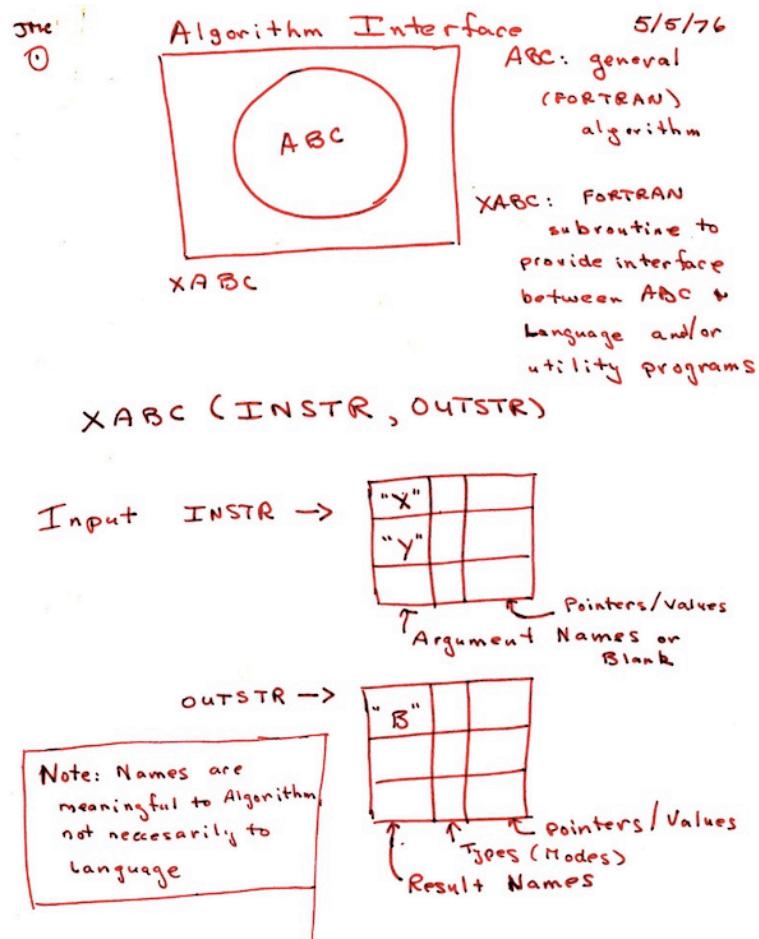


Figure 1: Design for an Interactive System. May 5, 1976

included by the time S was distributed outside AT&T. All function calls took some S objects as arguments and returned an S object as the value.

For example, the function

```
reg(x, y)
```

computed the linear regression of a vector *y* against the columns of a matrix *x*, returning an S object representing the regression, in this case a list with named components.

If the user had, say, read in a matrix and assigned it in S as *myData*, a regression of the first column against the next 3 columns would be:

```
r <- reg(myData[,1], myData[,2:4])
```

Information from the regression object such as *r\$coef* and *r\$resid* for coefficients and residuals could be used for further analysis of the results.

An interface function would have been written for *reg()* specifying the arguments, with *x* to be interpreted as a numeric matrix and *y* as a numeric vector. The interface function would call a Fortran subroutine expecting such arguments, typically also supplying expressions for additional required information. For example, the length of *y* would be available as *LENGTH(y)* in the interface language.

Such Fortran subroutines would usually return their result by filling in output arguments. The interface language included expressions for allocating corresponding objects in S and for returning an object containing the computed results. (See ([Chambers, 2016](#), pp 26–29) for the actual interface function for *reg()*.)

Interface functions were written for many of the analytical techniques and for graphics, including interaction. Basic summaries, data manipulation and simple facilities for data input and report generation were also included. All of these made use of existing code in the Fortran library.

The choice of a C-style user language seems obvious now, but was not standard for a statistical system at the time. For example, some of our colleagues who were Unix authors or users suggested that a shell-style language built around pipes would be just as capable and easier for users to learn. Our feeling was that nearly all the likely early users of S were comfortable with a scientific expression language. Also, general computations in data analysis fairly quickly become more tree-like. Both arguments in the regression example above are themselves function calls (in S, operators are functions), making a pure pipe syntax less convenient.

The user language retained this form throughout the evolution of S and R, although additional structure was added.

In a 2016 book on R, I asserted that its design can be summarized by three principles, ([Chambers, 2016](#), pp 4–11):

objects: Everything that exists in R is an object.

functions: Everything that happens in R is a function call.

interfaces: Interfaces to other languages are a part of R.

These principles did not suddenly appear at a late stage in the evolution of the software; rather, they are broadly visible from the first version of S and explain a number of detailed decisions.

For example, the centrality of function calls is clear from Figure 1 although functions as objects came later, as will be discussed below. The fundamental role of interfaces is also clear from the Figure; initially, only to Fortran but later to C also and to executable code generally.

The uniform approach to dynamically allocated § objects was partly a result of depending on the existing data management routines in the subroutine library. A resulting idiosyncrasy was that S had no scalar types as distinct from general objects. The intention was that low-level computations depending on these would be done through the interface to Fortran. It was also a practical decision: the available programming resources would have been insufficient to implement a language with similar capabilities from the ground up.

By 1978, a version of the system existed and was in use on the Murray Hill computer system (now Honeywell, which had purchased the General Electric computer division). The system survived without a name for its first few months but was eventually called "S" (initially with quotation marks).

S Outside Bell Labs

By the end of the 1970s, S was proving popular with users in statistics research and also in some other research and development organizations at Murray Hill. Only at Murray Hill, however, because it

was implemented in the operating system for our unintended Honeywell computer and built upon the local subroutine library written for that machine.

A desire to enlarge the user community motivated an effort to port S to other hardware and operating systems. The prospect of rewriting the system directly for a variety of targets was daunting. The mechanism in the S main program to swap in and communicate with code for individual functions was a custom-built overloading computation specialized to this operating system. Porting this and other parts of the system management to each target machine seemed likely to be tricky and tedious.

The solution came via another, separate fallout from the Multics debacle. Some of the Bell Labs computer science researchers involved with the Multics effort decided to pursue related operating system ideas, on a deliberately smaller scale. As the ideas crystallized into an actual system, it was called Unix [Ritchie \(1984\)](#).

At first, Unix seemed an unlikely target for S and not particularly helpful for portability. Like S, Unix was initially implemented on the available machine; in its case, the 16-bit PDP-11. Hardly a promising target for substantial data science and the initial uses of Unix did not include sizable numerical computations (no Fortran compiler, for example).

Fortunately, the growing popularity of Unix motivated them, like us, to strive for portability. A new version of Unix was designed for portability with operating system functionality callable from C [Johnson and Ritchie \(1978\)](#). Unix was ported to some 32-bit machines and a Fortran compiler included, through translation into C.

We took advantage of the popularity of Unix and its adoption on many platforms to define portable S to be a Unix implementation. Non-portable operations such as loading the code for functions and transferring control to them were re-implemented in C for Unix. This was technically a second version of S but retaining the existing user and interface languages, with only a few changes in individual functions.

Unix did us a second important service. Through negotiations with the appropriate legal organizations at Bell Labs, Unix was licensed for distribution outside AT&T. We were able to arrange for similar licensing of S.

By 1985, an S community was starting to grow, including particularly welcome interactions with university users, for whom an S license was relatively inexpensive. Statisticians in a number of prominent university departments included some enthusiastic users (e.g., at Wisconsin, Carnegie Mellon, Berkeley and Toronto). Two books described the software: a user's manual [Becker and Chambers \(1984\)](#) and a book describing the extension of S by writing functions in the interface language [Becker and Chambers \(1985\)](#).

1985–2000: S, Leading to R

At the same time that the S community was growing using the current version, the original authors were at work on a new version. The user language retained its grammar and the system supported nearly all of the functional capabilities, but the implementation was based on a new computational model.

The S software in this version was the basis for R. The plan for R was to reproduce the form and analytic capabilities of S, with additional features. This was in fact what happened, complicated by further evolution of S during the implementation of R, from which some new features were also incorporated.

To sort this out, it will be helpful to document the evolution of S during the period (Section 3.1), describe the creation of R (Section 3.2) and summarize the structure of R as it derives from S but also its relevant new features (Section 3.3). Lastly, we will relate the shared computational model of S and R to data science (Section 3.4).

S, Versions 3 and 4

The “new S”, as the title of its 1988 user manual [Becker et al. \(1988\)](#) described it, was not back-compatible. However, it aimed to give the user the same experience of high-level interactive data analysis combined with the ability to incorporate new research.

The subtitle of the original user manual [Becker and Chambers \(1984\)](#) was *An Interactive Environment for Data Analysis and Graphics*. The subtitle of the new book was almost identical except that *An Interactive ...* was replaced by *A Programming* The design aimed to provide a unified and convenient organization for programming in the language and for the organization of data.

The new S retained the three fundamental principles mentioned on page 417, but in a revised form.

objects: The uniformity of objects became explicit for the user, with properties such as the object's class available for programming. New classes of objects extended the analysis.

functions: The key change for programming was to add function definitions to the user language. Programming would now be centered on the creation of new S-language functions. Function definitions were now S objects that could be passed around and computed on.

interfaces: The interface language disappeared but inter-language interfaces remained central, especially for C but still for Fortran as well, along with an interface to the Unix shell.

S continued to evolve during this period, but largely by adding new capabilities or programming features without breaking back-compatibility, documented for the user community mainly by the 1988 book and two subsequent books.

A 1992 book, cite, introduced an approach to fitting and analyzing statistical models, using §'s object-based computations and a version of functional object-oriented programming. The statistical modeling software made use of the flexibility in objects and functions to create a unified and statistically informative approach to fitting models.

Statistical models were described and implemented in several classes (e.g., linear models, smooth curve fitting, ...). Within each class a particular model was defined by a structural *formula*, essentially an expression in S that was a symbolic representation of the particular model, and by the *data* from which the model should be estimated. The data would typically be gathered together in a new class of objects, the *data frame*, representing a sequence of n observations on the same p variables.

The conceptual framework of this software was carried over into R, along with most of the specific functionality described in Chambers and Hastie (1992), and became influential for future work in data analysis, two aspects in particular.: models as objects and the data frame as a representation of data for scientific studies. Viewing fitted models as objects from corresponding classes emphasized the data analysis philosophy encouraging visualization, examination and further modification. In S and R, generic functions for plotting, updating and extracting information will have methods for the various classes of models. The structure has lent itself to research and implementation of new classes of models, with corresponding R packages.

The data frame concept is central. It can be viewed both as a table of named entries (the variables) and as a rectangular array of the individual observations. However, it differs from a general dictionary or table in that each entry must have n elements corresponding to the observations and it differs from a matrix in that the entries may be of different types. In S and R, the data frame is implemented as a class built on a vector of type "list" with attributes defining the variable names and other properties. The rectangular structure is implemented by functional methods for the class; for example, to extract or replace data by matrix-like expressions. The data frame concept has been replicated in other software and languages, such as the `DataFrame` class in the pandas software in Python (<https://pandas.pydata.org>). Section 3.3 will discuss further details of the implementation.

The example of linear regression illustrates the concepts. The original `reg()` function regressed a numeric vector on a numeric matrix, but the new structure allowed more flexible possibilities. So, a fit of `runTime` to runners' `age` from a dataset of racing results Kaplan and Nolan (2015) might have the form

```
r1 <- lm( runTime ~ age, data = cbMen)
```

The first argument is now an expression that effectively produces a symbolic form for the model, typically containing the names of variables in the data frame supplied as the second argument — potentially in general S expressions, e.g. `log(age)`.

The object returned is from a class, "lm" in this case, for which functional methods simplify further study, such as specialized graphical displays:

```
plot(r1)
```

creates a specialized visualization useful for studying linear regression fits. Other statistical modelling techniques would replace `lm()` with `aov()`, for example for an analysis-of-variance model, but giving the user similar features for specifying the model and studying the result. Section 3.3 will relate these techniques to the design of the language.

At the same time that R was being implemented as a "free" S, additional research in statistical computing at Bell Labs produced a new version of S, extending it in a generally back-compatible way. From 1997, this was the version of S as licensed by Bell Labs (by then part of Lucent Technologies

after the further split of AT&T) and generally referred to as Version 4 of S, described in a 1998 book [Chambers \(1998\)](#).

Some of its features were incorporated in the initial version of R or added later. A more general, more formal version of object-oriented programming was provided, although it did not replace the earlier version in the white book. An additional interface to C was provided that passed references to S objects, on which users could program via C macros and utility functions. Classes of `connection` objects were defined to deal more generally with input and output (e.g., fifo and pipe connections). (Section 3.3 will examine these topics in more detail).

The licensing of S had always provided a re-sellers option for those wanting to produce a commercial product extending S. Of several such efforts, the dominant one became software known as S-Plus, managed by a company called MathSoft in 1998. The company changed names a few times until becoming a subsidiary of TibCo (the Wikipedia article, <https://en.wikipedia.org/wiki/S-PLUS> outlines the history). Eventually, the owners of S-Plus obtained an exclusive resale license and in 2004 bought the rights to the S software.

The S-Plus user community existed, and continues to exist to some extent, along with the growing R community. Some desire for compatibility of R with S-Plus was relevant in the development of R, as will be noted in Section 3.3.

The Birth of R

Ross Ihaka and Robert Gentleman published a paper in 1996 [Ihaka and Gentleman \(1996\)](#) describing a “language for data analysis and graphics”. The project had been previously announced through an S community letter and a small but growing group of contributors existed. Interest in R grew following the publication and by 1998 the contributors had expanded to a group of 11 “volunteers”, with write permission on the R source (an informal group that came to be known as *R core* and that has continued, with gradually varying membership, to be responsible for the official versions of R until the present day).

None of us at Bell Labs was consulted, ensuring a valid implementation freely available for distribution. Some notes by Ross Ihaka appear to be the main documentation of the early R core work [Ihaka \(1998\)](#). The goal of the project took its key form at this time: R would be “a free implementation of something ‘close to’ version 3 of the S language”.

Setting the goal of a “GNU S” (as R is still described on the project web site) was a watershed decision. The original form of the language and the basic approach to objects resembled S, with some internal distinctions partly reflecting Lisp-style languages and the writings of Abelson and Sussman [Abelson and Sussman \(1983\)](#). In particular, the data type known in R as a `pairlist` was the traditional Lisp list and was used to manipulate objects and for other basic computations. S function objects and the evaluation of calls used a form of closures.

Implementing a close approximation to S required a different model for objects and for evaluation; respectively, vectors with attributes and so-called *lazy* evaluation (Section 3.3). Pair-list objects and closure semantics are still used internally, but are now largely isolated from the R user or programmer.

As noted, the definition of S used as a model for implementation was taken from the “blue” book and “white” book, since there was no formal definition. The blue book contained a semi-formal model of the language, including the evaluation semantics ([Becker et al., 1988](#), Ch. 11). An appendix, as before, provided the detailed documentation of the functions supplied with S. These two inclusions supplied at least an approximation to a definition of S, available without obtaining the licensed software itself.

The key content of the white book in terms of data frames and model objects was reproduced, along with the more standard of the specific types of models; e.g., linear regression and analysis of variance. Data frames in particular, ([Chambers and Hastie, 1992](#), Ch. 3), became central to many extensions in R, reflecting their basic relevance to data science. A data frame represents a set of n values for p observable variables, the classical form of scientific data. Computationally, it combines the properties of a list (of variables, possibly of different types) with those of an n by p array.

R thus inherited the computational techniques, interactive user interface and programming structure of S, shaped by the data analysis philosophy of Bell Labs. This in turn featured an emphasis on deep engagement with data, exploration and collaboration that constituted a pre-adaptation to the needs of modern data science.

The work of the R core group resulted in the release of version 1.0.0 of R on February 29, 2000.

R and S

This section examines the main technical characteristics of R, noting the features of S it implemented and those where it diverged or added features original with R. In addition to this historical, “vertical” view, we note a few of the main “horizontal” distinctions that differentiate both R and S from some other languages, since these often relate to the data science domain-language aspect of R.

Throughout this section, any descriptions unqualified by mentioning either S or R will indicate characteristics common to both, implemented in R to replicate known behavior in S. When S and R use different terms to refer to the same concept, the R term will be used.

The discussion is organized by the three principles into *Objects*, *Function calls* and *Interfaces*, plus a fourth topic, *Packages*. The impact of R as well as its relation to data science has been strongly influenced by the growth of the R community of users and contributors. For this growth, the R package structure has been the key addition to the software. The packages available from several sites, especially the central CRAN archive, now form an extensive, usable and relatively coherent code base for applications.

Objects

Objects in the original version of S were an extension of Fortran one-way arrays, called *vectors* in S. Unlike Fortran, S allocated vectors dynamically as required in function calls. These vectors contained elements of a single specified type, initially corresponding to the types found in Fortran. Two characteristics distinguished these S vectors from the basic types in other languages: for all types, elements may be “missing”, denoted by `NA`; and there are no scalar types.

Version 3 of S, and therefore R, retained vectors as the core data structure. An extensible facility for defining general object structure was built on this through two features. Vectors could be of type “`list`”, with elements being arbitrary objects; and any vector could have a named list of *attributes* to specify additional information.

Vectors with attributes supported an extensible mechanism for adding specialized structure to simple objects, at first implicitly and later explicitly. Objects essential to data science, such as matrices and multi-way arrays, could be considered built-in without requiring a primitive implementation. A general array is a vector that has an attribute named “`dim`” containing an integer vector of the dimensions. Thus arrays automatically can have any type of data and can allow for missing values. Separating the data from the attributes is helpful for data analysis, separating the logic that depends on the structure from the computations on the specific type of data.

Advances in data analysis led to the need for more specialized data structures and for specialized computations to generate and operate on them. The natural, and perhaps inevitable, language extension to implement this coherently was *functional object-oriented programming*. Functions may be *generic*, with the computational method for particular arguments selected corresponding to the class of the argument(s). Statistical models for data are a natural application and were in fact the motivation for the first implementation (Chambers and Hastie, 1992, Appendix A). A more general and more formal version followed in Chambers (1998), but the simpler one continues to be popular.

The uniformity of objects extends to functions. In particular, functions are simply objects, with a syntactic definition in the language. Note that a name is not part of the function definition, in contrast for example to Python or Julia. Assigning a function is not different from assigning any other object. As noted below, the semantics of function call evaluation are defined directly from the object, regardless of how that object was obtained.

R supports all the object structure of S but with an implementation at the primitive level reflecting influence from Lisp. Language objects such as function definitions and unevaluated function calls are implemented via Lisp-style lists, but these are largely hidden from users who are encouraged to manipulate such objects by conversion to and from vectors of type “`list`”.

A more important influence, specifically from the Scheme form of Lisp, is that R is lexically scoped. In particular, any assignment of a function object incorporates a reference to the *environment*—the other assigned objects existing where the assignment took place. Since everything is an object, this environment is itself an object, of type “`environment`”. As an ordinary object, an environment is effectively a dictionary of objects indexed by character strings. But in an environment, objects are accessed by reference, which deliberately contradicts the usual non-reference semantics of S. Changing an environment, by changing the object associated with a particular string, changes that environment wherever it is currently referenced.

Environments are key to evaluating function calls and to the installation and use of R packages, as discussed under these topics below. Environments can also be used directly in R and have been, for example to implement the usual form of object-oriented programming (Chambers, 2016, Ch.

11). Their reference-style semantics does pose some dangers: the same computation on an object in R might have different results if that object was an environment on one hand or a list with the same elements corresponding to the same names on the other.

Function calls

The “everything that happens is a function call” principle reflects a design goal of S to encourage and support *functional programming* in the language. Combined with the object principle, functional programming implies an explicit conceptual model for a function call. Based on the definition of the function and the objects supplied as arguments, an object is computed and returned as the result of the call. No modifications to the arguments or other side effects should result and the function definition should determine the computations. R is not a pure functional language: various tricks and special computations exist to violate the principal. However, the essential language structure promotes functional programming; in particular, through the implementation of the function call itself which differs from languages that regard the call as simply taking a vector of references to the arguments as objects.

A function call in R, when it is about to be evaluated, essentially consists of two objects: the function definition and an environment containing objects corresponding to each of the formal arguments in that definition. These usually resulted from a function call in the language with the function identified by name and with some number of actual arguments supplied as expressions, but nothing in the evaluation depends on assuming this. The objects corresponding to formal arguments are special objects of type “promise”. A promise object has the expression for the corresponding actual argument (or a special marker for missing arguments), the value of that argument (*if* it has been evaluated), and a flag set when evaluation takes place.

The call is evaluated by evaluating the body of the function “in” the environment of the call; i.e., a name encountered in the evaluation will be searched for there. Ordinary assignments will create objects there. When the name corresponds to a promise object, the promise will be evaluated if it has not been already, in the environment from which the function was called, and the result will be stored in the promise. If the argument was missing and the function definition included a default expression, that expression is evaluated, this time in the environment of the call. So, for example:

```
function(x, scale = sd(x, na.rm = TRUE))
  x/scale
```

is a function that scales an object by the value given and uses the standard deviation of the object (missing values removed) by default.

This evaluation mechanism is often called “lazy” evaluation, but a better term would be evaluate-when-needed. It usually would give the same result as a model where all arguments were evaluated at the start of the call, but some functions depend on the distinction, and might fail without the extra flexibility. For example, default values can use intermediate results computed in the call before the missing argument was needed.

When a name occurs in the evaluation, it is first matched to the environment of the call, then to the parent of that environment, and so on. The parent of the call is the environment of the function definition, which is the environment in which the definition was evaluated. This is used in some computations to create functions inside a call, with the effect that these can then share variables in the original function. More importantly, all the functions in an R package have the same environment, providing a mechanism for sharing specialized tools and data within and between packages.

Interfaces

Effective data analysis today needs to use a variety of powerful tools for modelling and visualization. Well-developed implementations may exist in any of a variety of languages, including C++, Python, Java or Julia as well as R itself and (still) C or Fortran. It is neither practical nor sensible to reprogram the software in a single language; therefore, convenient interfaces from the user’s preferred programming language are essential. R now has interfaces on the CRAN repository to all of the above, with several of them being widely used.

The three interfaces of S were retained, to Fortran and C routines for simple arguments and to C with general pointers to R objects. The latter is generally more in use, lending itself to extensions and general computations. For C-level access to objects, R initially replicated the C macro calls

used with S, but this has been extended and replaced with its own version. There is also a third C interface, using the Lisp-style representation of the argument list.

The **Rcpp** interface to C++ is used extensively in packages based on specialized C++ code. The original **Rcpp** is described in [Eddelbuettel and François \(2011\)](#), but the interface has been much extended in the version now on CRAN. Approximately 10% of the packages on CRAN use **Rcpp**. **Rcpp** includes extensions to C++ to support a high-level programming style with R objects that in many ways resurrects the features of the original interface language of Section 2.3, but now for C++.

Packages

For any open-source software, an important advantage is that experienced users are encouraged to share their extensions and applications, and that the license for the software may enforce these to also be freely available, if distributed. The shared software may just be a folder of source code files, but will be more useful to the community if accompanied by documentation and made straightforward to include and to access from the user's software.

R has an extended definition for shared software, the R package. This has a prescribed hierarchical structure. The structure provides for documentation, R source, data files and some top-level description of the package. There is also a standard structure that simplifies inclusion of code for compilation in C, C++ and Fortran. Further optional structure allows inclusion of essentially anything, notably code from any other language. A collection of tools is used to install, load and invoke software from the package.

Compared to libraries or modules in other languages, the package in R imposes considerable demands on the programmer; for example, where a Python module is essentially just a folder containing code, an R package organizes code in a specific structure of subdirectories that then enables a very general but standardized format. In the basic package-sharing mechanism, a source copy of the package is processed by an **INSTALL** utility into a folder whose contents can then be loaded into an R session, making the code, documentation and any other content available to the user of the session.

The package structure and repositories of contributed packages have played a major role in the usefulness and popularity of R. They put some extra burden on providers of the extended software, particularly if the package is to be accepted by one of the central repositories, notably CRAN, which is by far the largest and most used site and is associated with the R project itself. CRAN enforces standards for the documentation, portability and usability of contributed packages. This is more than compensated by benefits to users in terms of software documentation and testing, and usually benefits the authors also in the long-term evolution of their software.

Data Science

Some of the central and influential features of S as described in the two books of 1988 and 1992 illustrate its nature as a domain-specific language and system for data science. R took over these features, adding some important extensions and improvements but with the focus still on data science.

In 1988, the preface to the blue book [Becker et al. \(1988\)](#), stated:

The primary goal of the S environment is to enable and encourage good data analysis.

This explicitly states the goal as supporting the domain of data analysis, Bell Labs style, very much in the spirit of modern data science.

The domain of data science was also implied (though still not named as such) in the citation when S received the 1998 ACM Software System Award, [ACM \(1998\)](#): § had “forever altered how people analyze, visualize, and manipulate data”.

Some of the capabilities of S important to users trying to do data science are implied in the citation:

- *visualization*, usually referred to as “graphics” in the books. The blue book preface, in listing key features, said “Especially, S is about *graphics*: ... flexible ways of looking at data”.
- *analysis*. S introduced an object-based view of analysis. A linear regression fit, for example, returned an object from a corresponding class. Simple S expressions then produced visual and numerical information, encouraging interactive exploration. This style has become the norm for modern data analysis.

- *data*. Some classes of data introduced have become central to data science in R and beyond.

From a data science perspective, the most important class of objects is the *data frame*, which models the structure in which scientific data has always been recorded: a table indexed by observations and variables in which each observation records corresponding values of the variables. As noted previously, this class was introduced to S in the context of statistical models, but is widely used and remains an active area for new developments, such as the “tidy” version in *R for Data Science* Wickham and Grolemund (2016). Analogous types have been added to software in other languages, such as the `DataFrame` structure in the Pandas software in Python.

From a programming perspective, however, a data frame does not correspond to a standard type of data. It cannot be a two-way array in the sense of Fortran (or R or Julia), because while all the values of a single variable will be constrained to have the same type, different variables can correspond to different types (numeric versus categorical, for example). Neither can it be a simple table or dictionary indexed by the names of the variables. The number and order of values from the variables is linked by their correspondence to particular observations; operations on the object cannot be allowed to revise some elements inconsistently with the rest.

From 2000: R

S did not quite disappear with the initial arrival of R. During the years 1996 to 2000 when the R Core team was preparing the official version of R, another version of S appeared at Bell Labs, as noted, while independently the commercial S-Plus system based on S maintained a significant user base.

After the official launch of R in 2000, open-source R gradually became the dominant source of new software for statistics and data science. S-Plus continued as a commercial product. However, by late 2000 the Bell Labs researchers still involved with S, Duncan Temple Lang Temple Lang (1997) and the present author, had both accepted invitations to join R Core. By the start of the 21st century, the evolution of Bell Labs’ S had ended.

Since its official first version, R has expanded in all measures: users, contributors, citations and public awareness. Some measures have shown literally exponential growth, such as the number of packages in the main repository, CRAN. Although S had accumulated a significant user base by the time of the Software System Award in 1998, the impact of R is on an entirely different scale.

R’s popularity is no doubt due to a number of factors but a principal one is its evident link with data science, which has shown a similar explosion of public interest and involvement. R features in the teaching and practical projects for nearly all programs in data science. Data science is inevitably mentioned in popular articles on R, e.g. Thieme (2018).

R inherited the data science orientation of S by replicating the structure and contents of Version 3 of S. To this R added some key contributions of its own in the internal computational model; for example, the role of closures and of R environments generally. For the future of its contributions to data science, perhaps the most important feature of R was the package structure.

In the balance between effort required from the software developer and the usefulness of the result, R tips the scales toward the user. This is reinforced by the central CRAN repository. A package on CRAN is more visible, easier for users to install and has some extra prestige, motivating developers to spend some effort to comply with the requirements.

That effort has been reduced by tools to assist the creation, modification and distribution of packages, particularly before they are ready to be part of CRAN or similar repositories. Some of the tools are R-independent; for example, github repositories have become virtually standard for circulating a package in a more flexible evolving format.

Integrated development environments (IDEs) specifically for R have also accelerated the creation and revision of packages and other R software. A notable example is the RStudio IDE. This is a desktop integrating the use of R with editing, graphics, documentation and a variety of utilities that typically replace specialized R- or shell-level commands with button clicks or other interactions. RStudio is a commercial enterprise but the IDE and many associated R packages are freely available and open-source (www.rstudio.com). The RStudio IDE has become popular for teaching and specifically for courses associated with data science. It also greatly simplifies editing and installing an R package.

Repositories of contributed packages, and in particular CRAN, have become a key driver in the growth and extension of R as specialized for scientific disciplines or other areas of application. This has relevance for data science: progress will require increasing collaboration between researchers in the scientific disciplines, on one hand, and professionals developing the statistical and computer-science techniques for data science, on the other. R and its package structure are popular ways to

bring data science techniques to a specialized audience; for example, a plethora of “Using R for xxx Data” books have appeared, with corresponding packages.

Data science will increasingly require a widening range of high-quality software for diverse purposes. No single language or environment will be universally suitable. Interfaces between languages have always been part of the design of S and now R. The R package structure facilitates including code from other languages: compiled code from C, C++ and Fortran but also code in any other language to which R has an interface (Chambers, 2016, Part IV). Multi-language IDEs such as Jupyter are complemented by such interfaces, allowing the R programmer to request specific computations from the other software in a natural R style. The end user does not need to do any programming in the language on the other side of the interface, in contrast to the IDE approach.

Acknowledgments

Thanks to Jean-Baptiste Tristan, the conference historian, and the referees for suggestions on early versions of the paper and to members of R Core for correcting some historical details. Most especially, much gratitude to all the members of the S and R teams over these many years. Both languages have always been the product of close collaboration, including all the authors of the related books in the reference list but also those extending S and R, now a community of thousands over different disciplines and applications.

Bibliography

- H. Abelson and G. J. Sussman. *Structure and Interpretation of Computer Programs*. MIT Press, Cambridge, MA, 1983. [p]
- ACM. ACM Software System Award, 1998. https://awards.acm.org/award_winners/chambers_6640862. [p]
- R. A. Becker and J. M. Chambers. GR-Z: A system of graphical subroutines for data analysis. In *Proc. 9th Interface Symp. Computer Science and Statistics*, 1976. [p]
- R. A. Becker and J. M. Chambers. *S: An Interactive Environment for Data Analysis and Graphics*. Wadsworth, Belmont CA, 1984. [p]
- R. A. Becker and J. M. Chambers. *Extending the S System*. Wadsworth, Belmont CA, 1985. [p]
- R. A. Becker, J. M. Chambers, and A. R. Wilks. *The New S Language*. Chapman & Hall, Boca Raton, FL, 1988. [p]
- J. M. Chambers. *Programming with Data: A Guide to the S Language*. Springer, New York, 1998. [p]
- J. M. Chambers. *Extending R*. Chapman & Hall/CRC, 2016. [p]
- J. M. Chambers and T. Hastie, editors. *Statistical Models in S*. Chapman & Hall, Boca Raton, FL, 1992. [p]
- F. J. Corbató and V. A. Vyssotsky. Introduction and overview of the multics system. In *Proceedings of the November 30–December 1, 1965, Fall Joint Computer Conference, Part I*, AFIPS '65 (Fall, part I), pages 185–196, New York, NY, USA, 1965. ACM. doi: 10.1145/1463891.1463912. [p]
- D. Donoho. 50 years of data science. *Journal of Computational and Graphical Statistics*, 26(4): 745–766, 2017. doi: 10.1080/10618600.2017.1384734. [p]
- D. Eddelbuettel and R. François. Rcpp: seamless R and C++ integration. *Journal of Statistical Software*, 40(8):1–18, 2011. doi: 10.18637/jss.v040.i08. URL <http://www.jstatsoft.org/v40/i08/>. [p]
- A. E. Freeny and J. D. Gabbe. A statistical description of intense rainfall. *Bell System Technical Journal*, 48:1789–1851, 1969. [p]
- J. Gertner. *The Idea Factory: Bell Labs and the Great Age of American Innovation*. Penguin, 2013. [p]
- R. Ihaka. *R : Past and Future History*, 1998. (draft for Interface Symp. Computer Science and Statistics): <https://cran.r-project.org/doc/html/interface98-paper/paper.html>. [p]

- R. Ihaka and R. Gentleman. R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, 5:299–314, 1996. [p]
- L. Jaeckel and J. Gabbe. Crawford hill rainfall data. In *Exploring Data Analysis: The Computer Revolution in Statistics*, chapter 3. University of California Press, 1974. [p]
- S. Johnson and D. M. Ritchie. UNIX time-sharing system: portability of C programs and the UNIX system. *Bell System Technical Journal*, 57(6):2021–2048, 1978. [p]
- D. Kaplan and D. Nolan. Modeling runners' times in the cherry blossom race. In D. Nolan and D. Temple Lang, editors, *Data Science in R*, chapter 2, pages 45–103. Chapman and Hall/CRC, 2015. [p]
- D. M. Ritchie. The evolution of the UNIX time-sharing system. *AT&T Bell Laboratories Technical Journal*, 63(8):1577–1593, 1984. [p]
- D. Temple Lang. *A Multi Threaded Extension to a High Level Interactive Statistical Computing Environment*. PhD thesis, University of California, Berkeley, 1997. [p]
- N. Thieme. R Generation. *Significance*, 15(4):14–19, August 2018. [p]
- J. W. Tukey. The future of data analysis. *The Annals of Mathematical Statistics*, 33(1):1–67, 1962. [p]
- J. W. Tukey. *Exploratory Data Analysis*. Addison-Wesley, Reading, Massachusetts, 1977. [p]
- H. Wickham and G. Grolemund. *R for Data Science: Import, Tidy, Transform, Visualize, and Model Data*. O'Reilly, 2016. [p]
- M. B. Wilk and R. Gnanadesikan. Probability plotting methods for the analysis of data. *Biometrika*, 55(1):1–17, 1968. [p]

*John M. Chambers
Stanford University
line 390 Serra Mall
line Stanford, CA 94305 USA*

jmc@stat.stanford.edu

Provenance of R's Gradient Optimizers

by John C Nash

Abstract Gradient optimization methods (function minimizers) are well-represented in both the base and package universe of R (R Core Team, 2019). However, some of the methods and the codes developed from them were published before standards for hardware and software were established, in particular the IEEE arithmetic (IEEE, 1985). There have been cases of unexpected behaviour or outright errors, and these are the focus of the **histoRicalg** project. A summary history of some of the tools in R for gradient optimization methods is presented to give perspective on such methods and the occasions where they could be used effectively.

The task

Ignoring exogenous data (assume that it is supplied when needed), our problem is to find

$$\operatorname{argmin}_x f(x)$$

where x is our set of n parameters and $f()$ is a scalar real function of those parameters. The gradient of $f(x)$ is the vector valued function

$$g(x) = \partial f(x) / \partial x.$$

The Hessian of $f(x)$ is the matrix of second partial derivatives

$$H(x) = \partial^2 f(x) / \partial x^2.$$

We will be considering methods for this problem where a gradient can be supplied, though many of the R tools will use a numerical approximation if needed.

General approaches

The so-called **Newton** method tries to find x that satisfies the first order conditions for an extremum, namely,

$$H(x)\delta = -g(x)$$

and then updates x to $(x + \delta)$.

The main objections to this approach are

- the Hessian is generally expensive to compute, though because an accurate Hessian is so useful, I strongly recommend that workers examine whether it can be computed in a reasonable way;
- it is necessary to apply safeguards on the computations and the size of δ to avoid cases where the Hessian is near singular, poorly computed, or the assumptions of the method are violated.

There are four main approaches to simplifying the Newton method:

- The **variable metric** or **quasi-Newton** methods use clever ways to improve an approximate Hessian or Hessian inverse while finding lower points x on the function surface, using only function and gradient values computed in the process.
- Truncated Newton methods use a linear conjugate gradient method to inexactly solve the Newton equations, thereby reducing memory requirements.
- Conjugate gradient methods aim to search "downhill" for better points on the functional surface using only recent gradient information. The traditional first search direction is that of steepest descents, namely, $-g$, the negative gradient. After a line search selects a suitable step size, we have a new point, a new function value that is lower than the initial point, and a new gradient. Using this information and possibly the last search direction, we compute a new search direction that is somehow "conjugate" to the previous one. For a problem with n parameters, of course, there are only n independent search directions, so we need to restart the procedure on or before that step. Indeed there are a number of strategies for deciding when to restart the conjugacy cycle. And, of course, there are a number of choices for updating the search direction and for performing the line search.
- Limited Memory BFGS methods use reduced memory approaches to variable metric methods, but can also be thought of as extending conjugate gradients methods by using several rather than the two most recent gradients.

The histoRicalg project

The R Consortium awarded some modest funding for **histoRicalg**, a project to document and transfer knowledge of some older algorithms used by R and by other computational systems. These older codes are mainly in Fortran, but some are in C, with the original implementations possibly in other programming languages. This effort was prompted by finding some apparent bugs in codes, which could be either from the original programs or else the implementations. Two examples in particular – in `nlm()` and in `optim--L-BFGS-B` – gave motivation for the project. We welcome interest and participation, and have a project repository <https://gitlab.com/nashjc/histoRicalg>, where several aspects of this work are in various stages of completion.

The function `optim()` in base-R has three of the optimizers from the 1990 edition of Compact Numerical Methods (Nash, 1979), abbreviated CNM henceforth. Given the dates of their conception, these are an obvious focus for **histoRicalg**. The direct-search Nelder-Mead method (Nelder and Mead, 1965) is in fact the default solver i.e., it uses `method="Nelder-Mead"` in the `optim()` call and does not require a gradient routine to be supplied. In fact, Nelder-Mead will be used even if a gradient routine is included unless some other method is suggested. The choice if `method="BFGS"` is the variable metric method of Fletcher (1970). The third choice from CNM is `method="CG"` for conjugate gradients, which is a combination of several similar approaches.

In this article on the provenance of the methods, the details will be discussed only in general terms. However, it is critical to get those details right.

Provenance of the R optim–BFGS solver

If the source code for base R is in a directory that is named R-X.Y.Z (in my case 3.5.1 when this vignette was started) then the calling routine for `optim()` is `src/library/stats/R/optim.R`, but this uses the `.External2` method to call `src/library/stats/src/optim.c`, where the function `vmmmin` is called. However, the source for `vmmmin` is in `src/appl/optim.c`. I will venture that having two files of the same name in different directories is tempting an error.

I will use `optim--BFGS` when using the call `method="BFGS"` in `optim()`, and similar abbreviations for the other methods. To use `optim--BFGS`, the relevant code is a C routine `vmmmin`. In `src/appl/optim.c`, above this routine is the comment

```
{,comment}
/* BFGS variable-metric method, based on Pascal code
in J.C. Nash, 'Compact Numerical Methods for Computers', 2nd edition,
converted by p2c then re-crafted by B.D. Ripley */
```

As author of this work, I can say that the code used is Algorithm 21 of ([Nash, 1979](#)). In the First Edition, this was a step-and-description code which was replaced by Turbo Pascal in the Second Edition. The methods were worked out mainly on a Data General Nova which had partitions of between 3.5 K and 8 K bytes accessible via a 10 character per second teletype. The floating point available had a 24 bit mantissa, in a single level of precision with (as I recall) no guard digit. Programming was in a fairly early and simple form of BASIC.

This machine was at Agriculture Canada. In the late 1970s, it was replaced with a Data General Eclipse, but largely the facilities were the same, except the floating point went to 6 hexadecimal digits with no guard digit. I also implemented some codes on HP 9830 and Tektronix 4051 "programmable calculators". A Fortran translation was made as NASHLIB, which ran mainly on IBM System 360 class computers, but also was tested at least partially on Univac 1108, ICL 1906A and Xerox-Honeywell CP-6 machines. The codes were distributed as Fortran source code. See <https://gams.nist.gov/cgi-bin/serve.cgi/Package/NASHLIB>. I have been informed that the servers supporting this link will not be replaced when they fail.

NASHLIB dates from the 1979-1981 period, though the methods were developed in the mid-1970s at Agriculture Canada to support economic modelling. In 1975, I received an invitation (from Brian Ford) to collaborate with the Numerical Algorithms Group in Oxford, and Agriculture Canada generously allowed me two 6-week periods to do so. During the second of these, in the tail end of the hurricane of January 1976, I drove to Dundee to meet Roger Fletcher. He took an interest in the concept of a program that used minimal memory. We found a printout of the Fortran for the method in ([Fletcher, 1970](#)), and with a ruler and pencil, Roger and I scratched out the lines of code that were not strictly needed. As I recall, the main deletion was the cubic interpolation line search. The resulting method simply used backtracking with an acceptable point condition.

On my return to Ottawa, I coded the method in BASIC and made two small changes:

- I managed to re-use one vector, thereby reducing the storage requirement to a square matrix and five vectors of length n , the number of parameters.
- The very short floating point precision seemed to give early termination on some problems. Often this was due to failure of the update of the approximate inverse Hessian when an intermediate quantity indicated the approximate Hessian was not positive definite. As a quick and dirty fix, I simply checked if

the current search was a steepest descent. If so, then the method is terminated. If not, the inverse Hessian approximation is reset to the unit matrix and the cycle restarted. This "quick fix" has, of course, become permanent. My experience over the years suggests it is a reasonable compromise, but the possibility of better choices is still open.

Evolution of the code

Considering the simplicity of the method, it is surprising to me how robust and efficient it has shown itself to be over the last four decades.

The code does allow of some variations:

- the organization of some of the calculations may have an influence on outcomes. There are opportunities for accumulation of inner products, and the update of the inverse Hessian approximation involves subtractions, so digit cancellation could be an issue.
- tolerances for termination, for the "acceptable point" (Armijo) condition, and other settings could be changed. However, I have found the original settings seem to work as well or better than other choices I have tried.

In the mid-1980s, the BASIC code was extended to allow for masks (fixed parameters) and bounds (or box) constraints on the parameters (Nash and Walker-Smith, 1987). This adds about 50% more code, as well as vector storage for the constraints and indices to manage them (essentially $3 * n$), but does permit a much wider variety of problems to be solved. Even for unconstrained problems that may have difficult properties, imposing loose bounds can prevent extreme steps in the parameters. To add this capability to R, I put the package **Rvmmin** on CRAN in 2011 (Nash, 2018). This code is, as of 2018, part of a new **optimx** (Nash and Varadhan (2011), Nash (2014a)) package on CRAN. It is entirely written in R.

Extensions and related codes

There are other R packages with related capability. In particular, **ucminf** (Nielsen and Mortensen, 2012) is based on the Fortran code of Nielsen (2000). This appears to use a very similar algorithm to **optim--BFGS**, but employs a more sophisticated line search. This package does not, however, allow for constraints in the form of bounds or masks.

The approximate inverse Hessian could also be saved and used to provide some estimates of parameter dispersion. Clearly, the use of a steepest descents direction for the final line search in **Rvmmin** before termination means that the **penultimate** approximation must be saved. In practice, I have found the approximate inverse Hessian bears little or no resemblance to the actual Hessian. This may be because the construction of the approximation maintains the positive definite condition. It is an open question whether the approximation has any utility.

Package **mize** (Melville, 2017) offers some options to build a gradient minimizer and could possibly allow some algorithmic comparisons to be made, but I have not had the time to delve into this.

Rather more sophisticated codes are part of base R in **nlm()** and **nlminb()**. The former is a polyalgorithm of quasi-Newton type for unconstrained problems (Schnabel et al., 1985). The latter allows bounds constraints and is drawn from the PORT library (Fox, 1997) code by Gay (1990), but there are a number of common ideas in both methods. For users, these two functions generally work well with the default settings of their controls. This is fortunate, as even though I am fairly experienced with such programs, I hesitate to play with the numerous control parameters.

To underline this last sentence, there was a thread on the R-help mailing list in 2010 (see, for example, <https://stat.ethz.ch/pipermail/r-help/2010-July/245182.html>) noting that the default behaviour of **nlminb()** at that time was to assume a positive objective function such as a sum of squares. Thus the function would terminate if a non-positive evaluation occurred. This has since been corrected, but was present in R for some years, possibly resulting in erroneous results being published.

Provenance of the R optim–CG and related solvers

The original Algorithm 22 of CNM which was converted to C and included in R as the **method="CG"** choice for **optim()** by Brian Ripley is an approach that has never felt quite right to me. And I am the author! It offers three different search direction updates using the **type** element of the **control** list. The default is **type=1** for the Fletcher-Reeves update (Fletcher and Reeves, 1964), with 2 for Polak–Ribiere (Polak and Ribiere, 1969) and 3 for Beale–Sorenson (Sorenson, 1969), (Beale, 1972).

In the mid-1980's I updated the CG code (in BASIC) to handle bounds and masks. Then in 2009 I incorporated the Yuan/Dai (Dai and Yuan, 2001) search direction update that melds the different formulas used in **optim--CG**. This gives a remarkably effective method that retains a surprisingly short code, and entirely in R, with a version that handles bounds and masks. This is package **Rcgmin** (Nash, 2014c).

There has been a flurry of work on CG-related optimizers in the last decade or so, in particular associated with Hager and Zhang. See (Hager and Zhang, 2006a) and (Hager and Zhang, 2006b). I have experimentally wrapped the **CG-Descent** code, but as yet do not feel the program is ready

for production release. However, collaboration on this and related codes I have been exploring is welcome.

Provenance of the R optim–L-BFGS-B and related solvers

The base-R code `lbfgsb.c` (at the time of writing in R-3.5.2/src/appl/) is commented:

```
/* l-bfgs-b.f -- translated by f2c (version 19991025).

From ?optim:
The code for method "L-BFGS-B" is based on Fortran code by Zhu,
Byrd, Lu-Chen and Nocedal obtained from Netlib (file 'opt/lbfgs_bcm.shar')

The Fortran files contained no copyright information.

Byrd, R. H., Lu, P., Nocedal, J. and Zhu, C. (1995) A limited
memory algorithm for bound constrained optimization.
\emph{SIAM J. Scientific Computing}, \bold{16}, 1190--1208.

*/
```

The paper (Byrd et al., 1995) builds on (Lu et al., 1994). There have been a number of other workers who have followed-up on this work, but R code and packages seem to have largely stayed with codes derived from these original papers. Though the date of the paper is 1995, the ideas it embodies were around for a decade and a half at least, in particular in (Nocedal, 1980) and (Liu and Nocedal, 1989). The definitive Fortran code was published as (Zhu et al., 1997). This is available as `toms/778.zip` on <http://www.netlib.org>.

Besides the ACM TOMS code, there are two related codes from the Northwestern team on NETLIB:

- http://netlib.org/opt/lbfgs_um.shar is for unconstrained minimization, while
- http://netlib.org/opt/lbfgs_bcm.shar handles bounds constrained problems.

To these are attached references (Liu and Nocedal, 1989) and (Byrd et al., 1995) respectively, most likely reflecting the effort required to implement the constraints.

The unconstrained code has been converted to C under the leadership of Naoaki Okazaki (see <http://www.chokkan.org/software/liblbfgs/>, or the fork at <https://github.com/MIRTK/LBFGS>). This has been wrapped for R as the `lbfgs` package (Coppola et al., 2014). I have included this as one of the solvers callable from package `optimx`.

A side-by-side comparison of the main subroutines in the two downloads from Netlib (`toms` and `opt`) unfortunately shows a lot of differences. I have not tried to determine if these affect performance or are simply cosmetic.

More seriously perhaps, there were some deficiencies in the code(s), and in 2011 Nocedal's team published a Fortran code with some corrections (Morales and Nocedal, 2011). Since the R code in C predates this by many years, I prepared package `lbfgsb3` to wrap the Fortran code. However, I did not discover any test cases where the `optim`--L-BFGS-B and `lbfgsb3` gave different output different, though I confess that the tests I ran are not exhaustive.

In 2016, I was at a Fields Institute optimization conference in Toronto for the 70th birthday of Andy Conn. By sheer serendipity, Nocedal did not attend the conference, but sat down next to me at the conference dinner. When I asked him about the key changes, he said that the most important one was to fix the computation of the machine precision, which was not always correct in the 1995 code. Since R gets this number as `.Machine$double.eps`, the offending code is irrelevant.

Within (Morales and Nocedal, 2011), there is also reported an improvement in the subspace minimization that is applied in cases of bounds constraints. In the few tests I have applied with bounds constraints, I have yet to see any substantive differences, but welcome communication should such be found.

Using Rcpp (see Eddelbuettel and François (2011) and the Fortran code in package `lbfgs3`, Matthew Fidler developed package `lbfgsb3c`. As this provides a more standard call and return than `lbfgsb3`, Fidler and I have unified the two packages, but are still checking and cleaning the package at the time of writing.

Provenance of truncated Newton codes for R

There are (at least) two implementations of truncated Newton methods available.

`nloptr--tnewton()` is a wrapper of the NLOpt truncated Newton method translated to C and somewhat modified by Steven G. Johnson in the nlopt project (https://nlopt.readthedocs.io/en/latest/NLOpt_Algorithms/) from Fortran code due to Ladislav Luksan (<http://www.cs.cas.cz/luksan/subroutines.html>). The many layers and translations make it difficult to unravel the particular details in this code, and I do not feel confident to provide a competent overview.

`optimx--tn()` and `optimx--tnbc()` are translations from Matlab source of the Truncated Newton codes of Stephen Nash ([Nash, 1983](#)) that I prepared, initially in package `Rtnmin`. The code is entirely in R. In implementing the codes, the principal awkwardness was in making available to different functions a quite extensive set of variables relating to the function and gradient. My choice was to use the `list2env()` function to create an **environment** to hold this data. Note the survey of truncated Newton methods by Stephen in ([Nash, 2000](#)).

Discussion

This story of some of the function minimizers available to R users is not finished. There are certain to be related methods in packages or collections I have overlooked. Moreover, I have not had the time or energy to fully explore some of the items I have mentioned. Nevertheless, how codes come about is important to understanding their strengths and weaknesses and in maintaining them to a level that users can reliably use them. Such matters were a large consideration for [Nash \(2014b\)](#), but have been more explicitly addressed by `histoRicalg`.

When evaluating such tools, it is important to consider what is wanted. Personally, I value reliability as more important than speed. By this, I mean

- the program will always proceed towards a minimum
- on termination it will provide useful information on the result obtained, such as whether there are good indications of a satisfactory result, or that we have halted for some reason such as exhausting a computational limit.

"Speed" is, of course, desirable, but how this is measured is debatable. Execution time is notoriously sensitive to hardware and software environments, as well as to the way functions and gradients are coded. Number of function and gradient counts is an alternative, since these computations often are the bottleneck of optimization. However, there are some methods where the optimization calculations are demanding either in cycles or memory.

Readers may note that I have highlighted that some codes are written entirely in R. This allows for customization or in-line use of faster code, and I have exchanged notes with several workers wanting to speed up time-consuming optimizations by such ideas. Profiling and debugging tools are generally quite challenging to use when there are multiple programming languages involved. All-R code may also be much easier to read and maintain if well-coded, especially as the programming language support for "old" languages diminishes.

These considerations are, of course, why we have a number of methods, and why Ravi Varadhan and I prepared the `optimx` package with tools to allow for comparison of several methods. But please, do use the comparison for evaluating and choosing a method, not for production minimization by a "try everything" strategy. As new methods are released, we hope to include them in the set `optimx` can call via a unified interface that is very close to the `optim()` function. We note that a somewhat different approach to unifying methods is available in the ROI package of [Hornik et al. \(2011\)](#).

Bibliography

- E. M. L. Beale. A derivation of conjugate gradients. In F. A. Lootsma, editor, *Numerical Methods for Nonlinear Optimization*, pages 39–43. Academic Press, London, 1972. [p]
- R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu. A limited memory algorithm for bound constrained optimization. *SIAM J. Sci. Comput.*, 16(5):1190–1208, Sept. 1995. ISSN 1064-8275. doi: 10.1137/0916069. URL <http://dx.doi.org/10.1137/0916069>. [p]
- A. Coppola, B. Stewart, and N. Okazaki. *lbfgs: Limited-memory BFGS Optimization*, 2014. URL <https://CRAN.R-project.org/package=lbfgs>. R package version 1.2.1. [p]

- Y. H. Dai and Y. Yuan. An efficient hybrid conjugate gradient method for unconstrained optimization. *Annals of Operations Research*, 103(1-4):33–47, 2001. [p]
- D. Eddelbuettel and R. François. Rcpp: Seamless R and C++ integration. *Journal of Statistical Software*, 40(8):1–18, 2011. doi: 10.18637/jss.v040.i08. URL <http://www.jstatsoft.org/v40/i08/>. [p]
- R. Fletcher. A new approach to variable metric algorithms. *Computer Journal*, 13(3):317–322, 1970. [p]
- R. Fletcher and C. M. Reeves. Function minimization by conjugate gradients. *The Computer Journal*, 7(2):149–154, 1964. [p]
- P. Fox. The Port Mathematical Subroutine Library, version 3, 1997. URL <http://www.bell-labs.com/project/PORT/>. [p]
- D. M. Gay. Usage summary for selected optimization routines. Computing Science Technical Report 153. Technical report, AT&T Bell Laboratories, Murray Hill, 1990. [p]
- W. W. Hager and H. Zhang. Algorithm 851: CG_DESCENT, a conjugate gradient method with guaranteed descent. *ACM Transactions on Mathematical Software*, 32(1):113–137, Mar. 2006a. [p]
- W. W. Hager and H. Zhang. A survey of nonlinear conjugate gradient methods. *Pacific Journal of Optimization*, 2:35–58, 2006b. [p]
- K. Hornik, D. Meyer, and S. Theussl. *ROI: R Optimization Infrastructure*, 2011. URL <http://CRAN.R-project.org/package=ROI>. R package version 0.0-7. [p]
- IEEE. IEEE Standard for Binary Floating-Point Arithmetic. *ANSI/IEEE Std 754-1985*, pages 10–11, 1985. doi: 10.1109/IEEEESTD.1985.82928. [p]
- D. C. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45(1-3):503–528, 1989. doi: 10.1007/BF01589116. URL <https://doi.org/10.1007/BF01589116>. [p]
- P. Lu, J. Nocedal, C. Zhu, and R. H. Byrd. A limited-memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16:1190–1208, 1994. [p]
- J. Melville. *mize: Unconstrained Numerical Optimization Algorithms*, 2017. URL <https://CRAN.R-project.org/package=mize>. R package version 0.1.1. [p]
- J. L. Morales and J. Nocedal. Remark on Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound constrained optimization. *ACM Trans. Math. Softw.*, 38(1):7:1–7:4, Dec 2011. URL <http://doi.acm.org/10.1145/2049662.2049669>. [p]
- J. C. Nash. *Compact numerical methods for computers : linear algebra and function minimisation*. Adam Hilger, Bristol, 1979. Second Edition, 1990, Bristol: Institute of Physics Publications. [p]
- J. C. Nash. On best practice optimization methods in R. *Journal of Statistical Software*, 60(2): 1–14, 2014a. URL <http://www.jstatsoft.org/v60/i02/>. [p]
- J. C. Nash. *Nonlinear Parameter Optimization Using R Tools*. John Wiley & Sons: Chichester, May 2014b. ISBN 978-1-118-56928-3. URL <http://www.wiley.com/legacy/wileychi/nash/>. Companion website (see <http://www.wiley.com/legacy/wileychi/nash/>). JNfile: 14nlpor.pdf. [p]
- J. C. Nash. *Rcgmin: Conjugate Gradient Minimization of Nonlinear Functions*, 2014c. URL <https://CRAN.R-project.org/package=Rcgmin>. R package version 2013-2.21. [p]
- J. C. Nash. *Rvmmin: Variable Metric Nonlinear Function Minimization*, 2018. URL <https://CRAN.R-project.org/package=Rvmmin>. R package version 2018-4.17. [p]
- J. C. Nash and R. Varadhan. Unifying optimization algorithms to aid software system users: optimx for R. *Journal of Statistical Software*, 43(9):1–14, 8 2011. ISSN 1548-7660. URL <http://www.jstatsoft.org/v43/i09/>. [p]
- J. C. Nash and M. Walker-Smith. *Nonlinear Parameter Estimation: An Integrated System in BASIC*. Marcel Dekker, New York, 1987. See <http://www.nashinfo.com/nlpe.htm> for an expanded downloadable version. [p]

- S. G. Nash. *Truncated-Newton Methods for Large-Scale Minimization*, pages 91–100. Pergamon, 1983. [p]
- S. G. Nash. A survey of truncated-Newton methods. *Journal of Computational and Applied Mathematics*, 124:45–59, 2000. [p]
- J. A. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7(4):308–313, January 1965. [p]
- H. B. Nielsen. UCMINF - an algorithm for unconstrained, nonlinear optimization. Technical report, Department of Mathematical Modelling, Technical University of Denmark., dec 2000. URL <http://www2.imm.dtu.dk/~hbn/publ/TR0019.ps>. Report IMM-REP-2000-18. [p]
- H. B. Nielsen and S. B. Mortensen. *ucminf: General-purpose unconstrained non-linear optimization*, 2012. URL <http://CRAN.R-project.org/package=ucminf>. R package version 1.1-3. [p]
- J. Nocedal. Updating quasi-newton matrices with limited storage. *Mathematics of Computation*, 35:773–782, 7 1980. doi: 10.1090/S0025-5718-1980-0572855-7. [p]
- E. Polak and G. Ribiere. Note sur la convergence de méthodes de directions conjuguées. *Revue Française d'Informatique et de Recherche Opérationnelle*, 16:35–43, 1969. [p]
- R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2019. URL <https://www.R-project.org/>. [p]
- R. B. Schnabel, J. E. Koonatz, and B. E. Weiss. A modular system of algorithms for unconstrained minimization. *ACM Trans. Math. Softw.*, 11(4):419–440, Dec. 1985. ISSN 0098-3500. doi: 10.1145/6187.6192. URL <http://doi.acm.org/10.1145/6187.6192>. Original report CU-CS-480-82 from 1982, revised. The UNCMIN Manual. [p]
- H. Sorenson. Comparison of some conjugate direction procedures for function minimization. *Journal of The Franklin Institute - Engineering and Applied Mathematics*, 288:421–441, 12 1969. doi: 10.1016/0016-0032(69)90253-1. [p]
- C. Zhu, R. H. Byrd, P. Lu, and J. Nocedal. Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization. *ACM Trans. Math. Softw.*, 23(4):550–560, Dec. 1997. ISSN 0098-3500. doi: 10.1145/279232.279236. URL <http://doi.acm.org/10.1145/279232.279236>. [p]

John C. Nash
Retired Professor
University of Ottawa, Telfer School of Management
Ottawa, Ontario
Canada
<https://orcid.org/0000-0002-2762-8039>
nashjc@uottawa.ca

R Foundation News

by Torsten Hothorn

Donations and members

Membership fees and donations received between 2020-02-24 and 2020-09-08.

Donations

Ichu Cheng (Canada) Mitchell Gail (United States) ken ikeda (Japan) Parag Magunia (United States) John McMahon (United States) Daniel Wollschläger (Germany) 明彦田中 (Japan)

Supporting benefactors

McGill University, Ottawa (Canada) www.ohmybingo.com, Alderley Edge (United Kingdom)

Supporting institutions

University of Iowa, Iowa City (United States)

Supporting members

Diogo Almeida (United Arab Emirates) Paul Artes (United Kingdom) Ashanka Beligaswatte (Australia) Chris Billingham (United Kingdom) Wesley Brooks (United States) Robert Carnell (United States) Luca Cocconcelli (United Kingdom) Rémi Coulaud (France) Al-istair Cullum (United States) Ajit de Silva (United States) Dubravko Dolic (Germany) Gerrit Eichner (Germany) Martin Elff (Germany) Mitch Eppley (United States) Nathan Epstein (United States) cristiano esclapon (Switzerland) Guenter Faes (Germany) Gottfried Fischer (Austria) Jutta Gampe (Germany) Jan Marvin Garbuszus (Germany) Stefano Guazzetti (Italy) Chris Hanretty (United Kingdom) Takehiko Hayashi (Japan) Alessandro Ielpi (Canada) Christian Kampichler (Netherlands) Srikanth Kannan (India) Curtis Kephart (United States) sanghyeon kim (Korea, Republic of) Sebastian Koehler (Germany) Luca La Rocca (Italy) Adrien Le Guillou (France) Seungdoe Lee (Korea, Republic of) Bernhard Lehnert (Germany) Alain Lesaffre (Australia) Eric Lim (United Kingdom) Sharon Machlis (United States) John MacKintosh (United Kingdom) Michal Majka (Austria) harvey minnigh (Puerto Rico) Ernst Molitor (Germany) Jairo Montenegro Arjona (Colombia) David Monterde (Spain) Stefan Moog (Germany) Steffen Moritz (Germany) Jens Oehlschlägel (Germany) Jaesung James Park (Korea, Republic of) Matt Parker (United States) Bill Pikoounis (United States) Robert Selden (United States) Christian Seubert (Austria) Pedro Silva (Brazil) gabriel silver (United States) Berthold Stegemann (Germany) Harald Sterly (Germany) Dag Tanneberg (Germany) Nicholas Turner (United States) Philipp Upravitelev (Russian Federation) Robert van den Berg (Austria) Mark van der Loo (Netherlands) Frans van Dunné (Costa Rica)

Torsten Hothorn

Universität Zürich, Switzerland Torsten.Hothorn@R-project.org