Contents

Editorial ......................................................... 4

Contributed Research Articles

gk: An R Package for the g-and-k and Generalised g-and-h Distributions ........ 7
NlinTS: An R Package For Causality Detection in Time Series ................. 21
Mapping Smoothed Spatial Effect Estimates from Individual-Level Data: MapGAM . 32
mudfold: An R Package for Nonparametric IRT Modelling of Unfolding Processes . 49
tsmp: An R Package for Time Series with Matrix Profile .......................... 76
Individual-Level Modelling of Infectious Disease Data: EpiILM ............... 87
SurvBoost: An R Package for High-Dimensional Variable Selection in the Stratified Proportional Hazards Model via Gradient Boosting, .......................... 105
CoxPhLb: An R Package for Analyzing Length Biased Data under Cox Model .... 118
SortedEffects: Sorted Causal Effects in R .............................................. 131
npordtests: An R Package of Nonparametric Tests for Equality of Location Against Ordered Alternatives .......................... 147
lspartition: Partitioning-Based Least Squares Regression ......................... 172
Skew-t Expected Information Matrix Evaluation and Use for Standard Error Calculations ....................................................... 188
rcosmo: R Package for Analysis of Spherical, HEALPix and Cosmological Data .......................... 206
Tools for Analyzing R Code the Tidy Way .............................................. 226
spinfex: An R Package for Creating a Manual Tour of Low-dimensional Projections of Multivariate Data ....................................................... 243
ari: The Automated R Instructor .............................................................. 258
CopulaCenR: Copula based Regression Models for Bivariate Censored Data in R ....................................................... 266
mistr: A Computational Framework for Mixture and Composite Distributions .......................... 283
difNLR: Generalized Logistic Regression Models for DIF and DDF Detection ........ 300
BayesMallows: An R Package for the Bayesian Mallows Model ............... 324
Variable Importance Plots—An Introduction to the vip Package .......................... 343
SimilaR: R Code Clone and Plagiarism Detection .................................... 367
Linear Fractional Stable Motion with the rlfsm R Package ....................... 386
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, Yale University, USA

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 by John Chamber, documenting the history of R, which was initially published in the History of Programming Languages. Finally, this issue features 27 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

Astronomy

- rcosmo R Package for Analysis of Spherical, HEALPix and Cosmological Data

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 \texttt{rlfsm} R Package
• \texttt{mistr}: A Computational Framework for Mixture and Composite Distributions
• \texttt{mudfold}: An R Package for Nonparametric IRT Modelling of Unfolding Processes
• \texttt{NlinTS}: An R Package For Causality Detection in Time Series
• \texttt{npordtests}: An R Package of Nonparametric Tests for Equality of Location Against Ordered Alternatives
• Skew-t Expected Information Matrix Evaluation and Use for Standard Error Calculations
• \texttt{tsmp}: An R Package for Time Series with Matrix Profile
• The R package \texttt{NonProbEst} for estimation in non-probability surveys

Supervised learning

• \texttt{CopulaCenR}: Copula based Regression Models for Bivariate Censored Data in R
• \texttt{CoxPhLb}: An R Package for Analyzing Length Biased Data under Cox Model
• \texttt{difNLR}: Generalized Logistic Regression Models for DIF and DDF Detection
• \texttt{lspartition}: Partitioning-Based Least Squares Regression
• Mapping Smoothed Spatial Effect Estimates from Individual-Level Data: \texttt{MapGAM}
• \texttt{SemiCompRisks}: An R Package for the Analysis of Independent and Cluster-correlated Semi-competing Risks Data
• \texttt{SortedEffects}: Sorted Causal Effects in R
• \texttt{SurvBoost}: An R Package for High-Dimensional Variable Selection in the Stratified Proportional Hazards Model via Gradient Boosting

\textit{Michael J. Kane}  
michael.kane@r-project.org  
\textit{Yale University}
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

\[
X = A + B G(Z) H(Z), \tag{1}
\]

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 \( X, F_X : \mathbb{R} \to [0, 1] \), 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 $F_X^{-1} : [0, 1] \to \mathbb{R}$. The $g$-and-$k$ and $g$-and-$h$ distributions use a quantile function of the form $F^{-1}(u; \theta) = Q(z(u); \theta)$ where $z(\cdot)$ is the $\mathcal{N}(0, 1)$ quantile function and $\theta$ is a vector of parameters. The $Q$ functions are:

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

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 $Q_{gh}$ 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](http://example.com/fig1.png)

**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).

---

**The R Journal Vol. 12/1, June 2020**

**ISSN 2073-4859**
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 $\Phi$ 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.

Note that an alternative representation of $f(x)$ is $1/q'(u)$ where $g(u)$ 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.

<table>
<thead>
<tr>
<th></th>
<th>Time (microseconds)</th>
<th>Ratio vs normal</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Normal</td>
<td>g-and-k</td>
</tr>
<tr>
<td>Quantile function</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Normal</td>
<td>175</td>
<td>972</td>
</tr>
<tr>
<td>Random sampling</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cdf</td>
<td>150</td>
<td>921</td>
</tr>
<tr>
<td>pdf</td>
<td>369</td>
<td>138381</td>
</tr>
</tbody>
</table>

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.
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.

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 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.

```r
> 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.](image)

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 \(x_1, x_2, \ldots, x_n\) 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 \texttt{gk} help files for a full description of all the arguments available.

MCMC inference The \texttt{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 \(\theta\). We use a Metropolis-Hastings algorithm, in which a proposed new state of the chain \(\theta'\) is sampled by adding a \(\mathcal{N}(0, \Sigma)\) increment to the current state \(\theta_t\). A decision to accept or reject \(\theta'\) is made based on the prior and likelihood values at \(\theta_t\) and \(\theta'\) and a random variable (see steps 3-4 of Algorithm 1.)

Tuning \(\Sigma\) can be difficult. Haynes and Mengersen (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 \(\Sigma\) automatically during its operation. The resulting \(\theta_t\) is 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 \texttt{get_log_prior} which takes a vector of parameters as input and returns the log prior density. We allow the user to reparameterise \(\theta\), using log \(B\) rather than \(B\), via the \texttt{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 \texttt{pgk} or \texttt{gph} command requires \(n\) calls to numerical optimisation. Therefore MCMC becomes computationally expensive for even moderately large datasets.

Algorithm 1 The Adaptive Metropolis MCMC algorithm

\begin{itemize}
  \item \textbf{Input}: observations \(x\), prior density \(\pi(\theta)\), number of iterations to perform \(N\), initial state \(\theta_0\), initial variance matrix \(\Sigma_0\), pre-tuning period \(t_0\), tuning parameter \(\epsilon > 0\).
  \item \textbf{Loop} over \(1 \leq t \leq N\):
    \begin{enumerate}
      \item 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 \(\theta_1, \theta_2, \ldots, \theta_{t-1}\).
      \item Sample \(\theta' \sim \mathcal{N}(\theta_{t-1}, \Sigma_{t-1})\)
      \item Sample \(u \sim U(0, 1)\) and let \(r = \frac{\pi(\theta')L(\theta')}{\pi(\theta_{t-1})L(\theta_{t-1})}\).
      \item If \(u < r\) let \(\theta_t = \theta'\). Otherwise let \(\theta_t = \theta_{t-1}\).
    \end{enumerate}
  \item \textbf{Output}: sample \(\theta_0, \theta_1, \ldots, \theta_N\).
\end{itemize}

ABC inference The \texttt{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 \texttt{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, \texttt{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 \texttt{mcmc}, \texttt{abc} is a Bayesian method and requires a prior distribution for \( \theta \) to be provided. It is convenient for this to be provided in a different form to the \texttt{mcmc} case. A function \texttt{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 \( \theta_i \) from the prior.
3. For \( 1 \leq i \leq N \) simulate summary statistics \( s(x_i) \) given parameters \( \theta_i \). 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 \( d_i = \sum_{j=1}^{q} (s_{ij} - s_0)^2 / v_j \).
6. Find the \( M \) smallest \( d_i \) values and return the corresponding \( \theta_i \).

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 \texttt{abc} (Csilléry et al., 2012) and \texttt{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 \texttt{sumstats} argument: (1) the full order statistics; (2) octiles of the observations, \( E_1, E_2, \ldots, E_7 \); (3) robust estimates of the moments based on the octiles:

\[
S_A = E_4, \quad S_B = E_6 - E_2, \quad S_3 = (E_6 + E_2 - 2E_4) / S_B, \quad S_4 = (E_7 - E_5 + E_3 - E_1) / S_B.
\]

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 \( X_1, X_2, \ldots, X_N \) are \( g \)-and-\( k \) or \( g \)-and-\( h \) variables, and let \( X_{(1)} < X_{(2)} \ldots < X_{(N)} \) denote the order statistics. We replace \( E_i \) with \( E''_i = X_{(ri(N)/s)} \) 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 \( U(0,1) \) distribution using the exponential spacings method (Ripley, 1987). This is implemented by the \texttt{orderstats} function. The uniform order statistics are then substituted into \( F^{-1}(u) \).

**FDSA inference** The \texttt{fdsa} function performs inference using finite difference stochastic approximation (FDSA). FDSA, originally due to Kiefer and Wolfowitz (1952), attempts to find \( \theta^* \) minimising a loss function \( \mathcal{L}(\theta) \) by iteratively calculating estimates \( \theta_1, \theta_2, \ldots \). 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,

\[
\mathcal{L}(\theta) = -\log L(\theta) = -\sum_{i=1}^{n} \log f(x; \theta).
\]

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 constants 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 $\theta_0$, choice of $a_t$ and $c_t$ sequences, function $\hat{L}(\cdot)$ which calculates an unbiased estimate of $L(\cdot)$, number of iterations to perform $N$, vectors of (possibly infinite) upper and lower parameter bounds $\theta^+, \theta^-$. 

**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 $\Delta_i$ be a 4-dimensional vector whose $i$th component is 1 and others are zero.
   
   (b) Let $\phi^+ = P(\theta_i + c_i \Delta_i)$ and $\phi^- = P(\theta_i - c_i \Delta_i)$.
   
   (Here $P(\phi)$ is a projection operator. Its output is $\phi'$ such that $\phi'$ is the closest value to $\phi$ in $[\theta_i^+, \theta_i^-]$. The $i$ subscripts represent $i$th components.)
   
   (c) Let $\hat{g}_t = \frac{1}{|\phi^+ - \phi^-|} [\hat{L}(\phi^+) - \hat{L}(\phi^-)]$.

2. Let $\theta_{t+1} = P(\theta_t - a_t \hat{g}_t)$.

**Output:** Final estimate $\theta_N$.

The unbiased estimate of $L(\theta)$ required by Algorithm 3, $\hat{L}(\theta)$, 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), $\hat{L}(\theta)$ can be calculated using $-\text{sum}(\text{dgk}(y,A,B,g,k,\log=\text{TRUE})) \times n/m$ (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 $\hat{L}$ 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 $\hat{L}(\theta_0)$ 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 $\theta$, 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 $L(\theta)$ (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 $\theta_t$. An update moves $\theta_t$ 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 gg’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 \( \log \left( \frac{x_{t+1}}{x_t} \right) \). 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 \( \gamma \) 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 \( \gamma \) 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.](image)

**ABC**  
We ran ABC as follows:

```r
abc.prior = 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, rtprior = abc.prior, M=200,
            sumstats = 'moment estimates')
```

This simulated 10⁷ 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:

```r
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 = fdsa(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.

**MCMC**

We ran MCMC as follows:

```r
fdsa_est_tf = fdsa_out[1E5, 1:4]
Sigma0 = var(fdsa_out[1E5 + (-1000:0), 1:4])
log_prior = function(theta) {
  if (theta[4] < 0) return(-Inf)
  return(theta[2])
}
mcmcout_tf = mcmc(log_return, N = 1E4, logB = TRUE, get_log_prior = log_prior,
  theta0 = fdsa_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 \( B \mathbb{1}(k > 0) \) on \( (A, \log B, g, k) \) (where \( \mathbb{1} \) 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 $\text{gk}$ 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 is 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 FPSA algorithm which can find MLEs for large datasets in a reasonable time and has not been applied to this problem before.
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.
Appendix A: Formulae

The derivatives of the $Q$ functions are as follows:

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

$$Q'_{gh} (z; A, B, g, h, c) = B \exp \left(hz^2/2\right) R_{gh} (z; g, h),$$  \hspace{1cm} (5)

where

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

$$R_{gh} (z; g, h, c) = \left[1 + c \tanh \left(gz/2\right)\right]\left(1 + hz^2\right) + \frac{cgz}{2 \cosh^2 \left(gz/2\right)}.$$  \hspace{1cm} (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 $1 + c \tanh \left(gz/2\right) = 0$ (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

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

where

$$u = -gz/2,$$

and

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

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 $(u \sech u + \tanh u)^{-1}$ 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


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


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 it 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 $Y$:

$$\text{Model}_1 \quad Y_t = a_0 + \sum_{i=1}^{p} a_i Y_{t-i} + U_t,$$

$$\text{Model}_2 \quad Y_t = a_0 + \sum_{i=1}^{p} a_i Y_{t-i} + \sum_{i=1}^{p} b_i X_{t-i} + U_t,$$

where $p$ is the lag parameter, $[a_0, \ldots, a_p]$ and $[b_0, \ldots, b_p]$ 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 Model$_1$ and Model$_2$. In this case, the Granger causality index (GCI) can be used, and it is expressed as follows:

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

where $\sigma_1^2$ and $\sigma_2^2$ are the variances of the errors of Model$_1$ and Model$_2$ 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:

$$F = \frac{\text{RSS}_1 - \text{RSS}_2}{\text{RSS}_2 / (n - 2p - 1)}.$$

RSS$_1$ and RSS$_2$ are the residual sum of squares related to Model$_1$ and Model$_2$ resp., and $n$ is the size of the lagged variables. Two hypotheses have to be considered:

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

$H_0$ is the hypothesis that $X$ does not cause $Y$. Under $H_0$, $F$ follows the Fisher distribution with $(p, n - 2p - 1)$ 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 $\{Y_1, \ldots, Y_k\}$. 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 the 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:

$$Y_t = \Psi_{nn}\left(Y_{t-1}, \ldots, Y_{t-p}, Y_{k(t-1)}, \ldots, Y_{k(t-p)}\right) + U_t,$$

where $\Psi_{nn}$ 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,

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

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

where $\Psi_{1nn}$ and $\Psi_{2nn}$ are the network functions of Model 1 and Model 2 respectively, 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 respectively. 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:

\[ T_{X \rightarrow Y} = \sum_{Y_i Y_{i-1} X_t} P \left( Y_i, Y_{i-1}, X_t \right) \log \left( \frac{P \left( Y_i | Y_{i-1}, X_t \right)}{P \left( Y_i | X_t \right)} \right) \]

\[ = I \left( Y_t; X_t | Y_{t-l} \right), \]

where \( Z_i = (Z_{t-1}, \ldots, Z_{t-l}) \) 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:

\[ TE_{X \rightarrow Y} = H \left( Y_t | (Y_{t-1}, \ldots, Y_{t-q}) \right) \]
\[ - H \left( Y_t | (Y_{t-1}, \ldots, Y_{t-q}), (X_{t-1}, \ldots, X_{t-p}) \right), \]

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 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 \( \Gamma \) 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:

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

(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:

\[ \hat{H}(X) = \Gamma(n) - \frac{1}{n} \sum_{i=1}^{n} \Gamma(n_x (i) + 1) + \frac{m}{n} \sum_{i=1}^{n} d_i , \]  

(10)

where \( n_x (i) \) 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:

\[ I(X; Y) = \Gamma(k) + \Gamma(n) - \frac{1}{n} \sum_{i=1}^{n} \left( \Gamma(n_x (i) + 1) + \Gamma(n_y (i) + 1) \right) \]  

(11)

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

\[ I(X_1, \ldots, X_l) = \Gamma(k) + (l - 1) \Gamma(n) - \frac{1}{n} \sum_{i=1}^{n} \left( \Gamma(n_1 (i) + 1) + \cdots + \Gamma(n_l (i) + 1) \right) , \]  

(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 \( d_{ji} / 2 \).

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 \( (X, Y) \) 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 \( \{X_1, \ldots, X_l\} \), with \( n \) observations, can be expressed as follows:

\[ I(X_1, \ldots, X_l) = \Gamma(k) + (l - 1) \Gamma(n) - \frac{1}{n} \sum_{i=1}^{n} \left( \Gamma(n_1 (i) + 1) + \cdots + \Gamma(n_l (i) + 1) \right) , \]  

(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,:\n
\[ T_{E_{X \rightarrow Y}} = \hat{H}(Y_{t-q}, Y_{t-p}) - \hat{H}(Y_{t-q}, Y_{t-p}) \]  

(14)

Consider the following notations :

- \( Y^m_t = \{Y_{t-1}, \ldots, Y_{t-q}\} \)
- \( X^m_t = \{X_{t-1}, \ldots, X_{t-p}\} \)
- \( Y^m_t = \{Y_t, Y^m_t\} \)
- \( X^m_t = \{X_t, X^m_t\} \)
- \( Z^m_t = \{Y^m_t, X^m_t\} \)
- \( Z^m_t = \{Y^m_t, Y^m_t, Z^m_t\} \)
We can rewrite then Equation 14 as follows:

\[
\hat{T}_E_{X \rightarrow Y} = \hat{H} (Y_t, Y_p) - \hat{H} (Y_t|X_p, Y_p),
\]

(15)

\[
= \hat{H} (Y_t, Y_p) - \hat{H} (Y_p) - \hat{H} (Y_t, X_p, Y_p) + \hat{H} (X_p, Y_p)
\]

\[
= \hat{H} (Y_t^f) - \hat{H} (Y_t^n) - \hat{H} (Z_t^f) + \hat{H} (Z_t^n).
\]

The maximum joint space is defined by \(Z_t^f = \{Y_t, Y_t^n, X_t^n\}\). 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 \(\hat{H} (Z_t^f)\) using the Kozachenko-Leonenko estimator, and the other terms by projecting the number of neighbors in each marginal space using the Kraskov approach:

\[
\hat{H} (Z_t^f) = -\Gamma (k) + \Gamma (n) + \frac{p + q + 1}{n} \sum_{i=1}^{n} d_i,
\]

\[
\hat{H} (Y_t^f) = -\frac{1}{n} \sum_{i=1}^{n} \Gamma (n_yf (i) + 1) + \Gamma (n) + \frac{p + q}{n} \sum_{i=1}^{n} d_i,
\]

\[
\hat{H} (Y_t^n) = -\frac{1}{n} \sum_{i=1}^{n} \Gamma (n_ym (i) + 1) + \Gamma (n) + \frac{p + q}{n} \sum_{i=1}^{n} d_i,
\]

\[
\hat{H} (Z_t^n) = -\frac{1}{n} \sum_{i=1}^{n} \Gamma (n_zn (i) + 1) + \Gamma (n) + \frac{p + q}{n} \sum_{i=1}^{n} d_i,
\]

where \(n_yf (i)\), \(n_ym (i)\) and \(n_zn (i)\) are the numbers of points where the distance from the point \(Y_t^f\), \(Y_t^n\), and \(Z_t^n\) resp., is strictly less than \(d_i/2\), for \(i \in [1, n]\). By replacing each oh these terms in Equation 15, we obtain:

\[
\hat{T}_E_{X \rightarrow Y} = \hat{H} (Y_t) - \hat{H} (n) - \left( \frac{p + q}{n} - p - \frac{p + q + 1}{n} \right) \sum_{i=1}^{n} d_i
\]

(16)

\[
+ \frac{1}{n} \sum_{i=1}^{n} \left( -\Gamma (n_yf (i) + 1) + \Gamma (n_ym (i) + 1) - \Gamma (n_zn (i) + 1) \right).
\]

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

\[
\hat{T}_E_{X \rightarrow Y} = \hat{H} (k) - \hat{H} (n) + \frac{1}{n} \sum_{i=1}^{n} \left( \Gamma (n_ym (i) + 1) - \Gamma (n_yf (i) + 1) - \Gamma (n_zn (i) + 1) \right).
\]

(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 \(\hat{H} (Y|Y_t^n)\). Thus, the normalized transfer entropy (NTE) can be obtained by dividing TE by its maximum value:

\[
\text{NTE} = \frac{\hat{H} (Y_t|Y_t^n) - \hat{H} (Y_t|Y_t^n, X_t^n)}{\hat{H} (Y_t|Y_t^n)}
\]

(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):

\[
\text{NTE} = \frac{T_{E_{X \rightarrow Y}} - \sum_{p=1}^{n} T_{E_{X_{shuffled} \rightarrow Y}}}{\hat{H} (Y_t|Y_t^n)}
\]

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 \(\hat{H}(Y_t|Y_{m})\) 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):

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

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.

```r
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 \(pvalue\).

```r
# 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).

```r
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 (log2 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_cont 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):

```r
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):

```r
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):

```r
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:

```r
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:

```r
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


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
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
We consider modeling observations that are distributed on a map with covariates. We consider estimating and visualizing covariate-adjusted spatial effects in the context of a Generalized Additive Model (GAM) 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 covariates.

The R Journal Vol. 12/1, June 2020 ISSN 2073-4859

 penalties for multivariate models. mgcv also incorporates ‘coxhazard’ 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., 2006) 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, \( i = 1, \ldots, n \). 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

\[
g(\mu_i) = \eta_i = \beta_0 + X_i^T \beta + f(u_i, v_i),
\]

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 \( \text{Var}(Y_i) = V(\mu_i, \phi) \); a function of...
the mean and nuisance parameter $\phi$. $\beta$ 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

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

where $\tilde{X}_i = [X_i, u_i, v_i]$ and $\tilde{\beta} = [\beta_1^T, \gamma_1, \gamma_2]^T$. 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 $Y = [Y_1, \ldots, Y_n]^T$, which is a function of
$\eta = [\eta_1, \ldots, \eta_n]^T$. To estimate the parameters of the model we seek to maximize the expected log
likelihood:

$$E(l(\hat{\eta}, Y_i)) = \max_\eta E(l(\eta, Y_i)), \text{ for } i = 1, \ldots, n$$

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

$$E[dl/d\eta_i|\eta_0] = 0,$$

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

$$\eta_i^{new} = \eta_i^{old} - E[dl/d\eta_i]|_{\eta^old} / E[d^2l/d\eta_i^2]|_{\eta^old},$$

which is equivalent to

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

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

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

and

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

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

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

Hence $\eta_i$ is updated in the GLM case by

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

Further, letting $Y_i^{old} = [Y_i^{old}, \ldots, Y_i^{old}]^T$ denote the working response computed in terms of $\eta^{old}$ and
$\mu^{old}$ and given by

$$Y_i^{old} = \eta_i + (Y_i - \mu_i) \left( \frac{dl}{d\eta_i} \right)_{\eta^{old}, \eta^{old}}. \tag{11}$$

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

$$E[Y_i^{old}] = \beta_0 + \tilde{X}_i \tilde{\beta}^{new} + s^{new}. \tag{12}$$

The coefficients $\hat{\beta}_{0i}^{new}$, $\hat{\beta}_{0i}^{new}$ and nonparametric term, $s$, must be estimated in order to obtain an updated
value of $\eta_i^{new}$ 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 $Y_i^{old}$
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 $\hat{\beta}_{0i}, \hat{\beta}$ and $s$, as is done in the gam package.
Specifically, we begin by defining $W = [W_1, \ldots, W_n]^\top$ as

$$W_i = \left(c_i/d_i\right)^{2}V_i^{-1}W_{iold}W_{old}$$

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

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

1. Initialize $s = 0$. Initialize $\beta_0$ and $\hat{\beta}$ by fitting a generalized linear regression model with all the adjusted covariates and geolocation parameters included in the model (i.e. omitting $s$).
2. Loop:
   - (a) With the current estimated $\beta_0$, $\hat{\beta}$ and $s$, calculate $\eta_{iold}$ as well as working response $Y_{iold}$ and $W$ using Eq.(2), Eq.(4) and Eq.(13) respectively.
   - (b) Update $\beta_0$, $\hat{\beta}$ 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 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 $f(u_i, v_i)$, $i = 1, \ldots, n$, the mean of $f(u_i, v_i)$, $i = 1, \ldots, n$, 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). `MAmap` 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
R> data("MAdata")
R> data("MAmap")
R> summary(MAdata)
```

<table>
<thead>
<tr>
<th>Case</th>
<th>Xcoord</th>
<th>Ycoord</th>
<th>Smoking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min.</td>
<td>0.00</td>
<td>35354</td>
<td>Min. :778430 Min. :0.000</td>
</tr>
<tr>
<td>1st Qu.:0.00</td>
<td>1st Qu.:111465</td>
<td>1st Qu.:869089</td>
<td>1st Qu.:0.000</td>
</tr>
<tr>
<td>Median</td>
<td>0.00</td>
<td>183100</td>
<td>Median :891067 Median :0.000</td>
</tr>
<tr>
<td>Mean</td>
<td>0.09</td>
<td>175054</td>
<td>Mean :889081 Mean :0.177</td>
</tr>
<tr>
<td>3rd Qu.:0.00</td>
<td>3rd Qu.:236826</td>
<td>3rd Qu.:919684</td>
<td>3rd Qu.:0.000</td>
</tr>
<tr>
<td>Max.</td>
<td>1.00</td>
<td>327861</td>
<td>Max. :954253 Max. :1.000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mercury</th>
<th>Selenium</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min.</td>
<td>0.1418</td>
</tr>
<tr>
<td>1st Qu.:0.7286</td>
<td>1st Qu.:0.8573</td>
</tr>
<tr>
<td>Median</td>
<td>1.0010</td>
</tr>
<tr>
<td>Mean</td>
<td>1.1471</td>
</tr>
<tr>
<td>3rd Qu.:1.4017</td>
<td>3rd Qu.:1.6844</td>
</tr>
<tr>
<td>Max.</td>
<td>5.6298</td>
</tr>
</tbody>
</table>

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

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

![Map of Massachusetts](image_url)

*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()`.

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

Call:

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

Model:

```r
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
R> plot(fit1, exp = TRUE, MAmap, 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 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, $i = 1, \ldots, n$. Let $T_i$ denote the observed followup time and $\delta_i$ denote the indicator of whether or not $T_i$ represents the true failure time for observation $i$. Further, let $\bar{X}_i$ 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

$$\lambda_i(t) = \lambda_0(t) \exp\{\bar{X}_i^T \hat{\beta} + s_i\},$$

where $\lambda_i(t)$ represents the hazard function for observation $i$ evaluated at time $t$ and $\lambda_0(t)$ denotes the baseline hazard (i.e., 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

$$\eta_i = \bar{X}_i^T \hat{\beta} + s_i.$$  

(15)

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

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

(16)

and

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

(17)

respectively, where $D$ represents the set of indices of all unique failures and $R_j = \{ k \mid T_k \geq T_j \}$ 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:

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

(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 $\eta_i$ for observation $i$ as

$$\frac{dl}{d\eta_i} = d_j - \sum_{j \in R_i} \frac{e^{\eta_j}}{\sum_{k \in R_i} e^{\eta_k}},$$

and

$$\frac{d^2l}{d\eta_i^2} = -\sum_{j \in R_i} \frac{e^{\eta_j}}{\sum_{k \in R_i} e^{\eta_k}} + \sum_{j \in R_i} \frac{\sum_{k \in R_i} e^{\eta_k}}{\left(\sum_{k \in R_i} e^{\eta_k}\right)^2}.$$ 

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 $\eta$, 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 $\eta^{old}$, the new estimate for $\eta$ can be obtained using the following two steps:

1. Estimate $E[\frac{d^2l}{d\eta_i^2}]$ by fitting a generalized additive model using $[\frac{d^2l}{d\eta_i^2}]$, $i = 1, \cdots, n$ as responses, including the linear predictor of $X$ and a bivariate smoother of geolocation parameters;

2. Estimate $\eta^{new}$ using the backfitting algorithm described in Section 2.1 with $W_i = -1/E[\frac{d^2l}{d\eta_i^2}]$ as weights and $y_i^{old} = \eta_i^{old} - \left[dl/d\eta_i\right]_{\eta^{old}} / E[\frac{d^2l}{d\eta_i^2}]$ 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 $v$.

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

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

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

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 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.

**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.
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 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)

<table>
<thead>
<tr>
<th></th>
<th>event</th>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min.</td>
<td>0.004068</td>
<td>Min. :0.0000</td>
<td>Min. :1811375</td>
</tr>
<tr>
<td>1st Qu.</td>
<td>1.931247</td>
<td>1st Qu.:0.0000</td>
<td>1st Qu.:2018362</td>
</tr>
<tr>
<td>Median</td>
<td>4.749980</td>
<td>Median :1.0000</td>
<td>Median :2325884</td>
</tr>
<tr>
<td>Mean</td>
<td>6.496138</td>
<td>Mean :0.6062</td>
<td>Mean :2230219</td>
</tr>
<tr>
<td>3rd Qu.</td>
<td>9.609031</td>
<td>3rd Qu.:1.0000</td>
<td>3rd Qu.:2380230</td>
</tr>
<tr>
<td>Max.</td>
<td>24.997764</td>
<td>Max. :1.0000</td>
<td>Max. :2705633</td>
</tr>
</tbody>
</table>

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

![Map of California displaying the geolocations of the observations in CAdata. Depicted in the plot are censored observations (black, ‘o’) and observed event observations (red, ‘x’).](image)

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

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 $G_i Cov(z) G_i^\top$, 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 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 $G_i Cov(z) G_i^\top$. 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.
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.

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.perm` is specified in the function call, where `N.perm` 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 CAdata 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 = CAdata, 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 = CAdata, 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)Oth factor(INS)Uni
coefficients
  0.02657848  0.03657777  0.05251440  0.16770033  0.26790051
factor(INS)Unk
  0.07594159

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.

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 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


Appendix

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

```R
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
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
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
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)
```
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
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/
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 nonparametric 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 (Finserás 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 (2008). 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 Asil-kalan, 2019), which can be downloaded from the Comprehensive R Archive Network (CRAN). 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 \( i, i = 1, 2, \ldots, n \) are asked to state if they do agree or do not with each of \( j = 1, 2, \ldots, N \) statements (i.e. items) towards a unidimensional attitude \( \theta \) 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 \( \theta \). That is, the probability of positive endorsement of item \( j \) from individual \( i \) with latent parameter \( \theta_i \) we write \( P_j(\theta_i) = P\left(X_{ij} = 1|\theta_i\right) \). In PIRT models for unfolding, \( P_j(\theta_i) \) is a parametric unimodal function of the proximity between the subject parameter \( \theta_i \) and the item parameter \( \beta_j \). 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, Post and Snijders (1993) proposed two additional assumptions for the IRFs. The assumptions of the nonparametric unfolding model are:

1^URL: http://CRAN.R-project.org
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 $\theta$, i.e., the joint conditional probability of $N$ responses simplifies into the likelihood form,

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

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

For the sake of clarity, a function $P_j (\theta) : \mathbb{R} \to \mathbb{R}$, is weakly unimodal if there exists a $\beta_j \in (-\infty, +\infty)$ such that, $P_j (\theta)$ is non decreasing for $\theta \leq \beta_j$ and non increasing for $\theta \geq \beta_j$. The location parameter $\beta_j$ 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 $\beta_j$ is not unique).

A4. **Stochastic Ordering (SO):** For any probability distribution $G(\theta)$ of latent trait values and any value $\theta_0$ on the latent scale, $P_G \left( \theta > \theta_0 \mid X_i = 1 \right)$ is non decreasing 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 $\theta$ have a monotone likelihood ratio (MLR) in $\theta$, 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 $P_G \left( \theta_1 < \theta < \theta_2 \mid X_i = 1 \right)$ 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 $\beta_j$ such that $\beta_1 \leq \beta_2 \leq \cdots \leq \beta_N$. The set of assumptions A1-A3 is the core in unfolding IRT models.

Additionally, two assumptions are needed about the individuals $\{i \mid i = 1, \ldots, n\}$ and the distribution $G$ of their latent trait values $\{\theta_i \mid i = 1, \ldots, n\}$ 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 $\theta$ given a positive response to an item located at $\beta_j$ is stochastically ordered by the location $\beta_j$ (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 $\theta_i$ that is most likely close to item location $\beta_j$ 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 (TP2) (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 TP2.

**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
The R Journal Vol. 12/1, June 2020 ISSN 2073-4859

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 item triple \((h,l,k)\) with \(h, l, k\) distinct elements of the set \(\{1,2,\ldots,N\}\), is the frequency of \(\{1,0,1\}\) responses over all individuals. The observed errors can be calculated by \(O_{hlk} = \sum_{i=1}^{n} x_{i,h}(1-x_{i,l})x_{i,k}\) where \(x_{i,j}\) is the realization of random variable \(X_{i}\), and \(x_{i,j} = 1\) if the \(i^{th}\) individual responds positively on item \(j\) otherwise \(x_{i,j} = 0\). 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 \(\{1,2,\ldots,N\}\).

**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 \(h, l, k\) 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 \(j\) as \(p(j) = \sum_{i=1}^{n} \frac{x_{i,j}}{n}\) 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,

\[
H_{hlk} = 1 - \frac{O_{hlk}}{EO_{hlk}}, \quad \forall h,l,k \in \{1,2,\ldots,N\}.
\]

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 \(j \in s\), \(j = 1,2,\ldots,m\) is given by,

\[
H_{j}(s) = 1 - \frac{\sum_{(h,l,k) \in T_{j}(s)} O_{hlk}}{\sum_{(h,l,k) \in T_{j}(s)} EO_{hlk}},
\]

where \(T_{j}(s) = \{s_{h},s_{i},s_{k} \mid s_{h} < s_{i} < s_{k} : j \in \{s_{h},s_{i},s_{k}\}\}\) 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 \(m(m-1)/2\) 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,

\[
H_{total}(s) = 1 - \frac{\sum_{(h,l,k) \in T(s)} O_{hlk}}{\sum_{(h,l,k) \in T(s)} EO_{hlk}},
\]

where \(T(s) = \{s_{h},s_{i},s_{k} \mid s_{h} < s_{i} < s_{k}\}\) 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 \(H_{total}(s) = 1\). The latter means that no error patterns are observed in this scale. Likewise, \(H_{total}(s) = 0\) 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.

```r
# Install and load the mudfold package and the ANDRICH data
download.packages("mudfold")
library(mudfold)
```

The R Journal Vol. 12/1, June 2020 ISSN 2073-4859
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.

```r
experr <- mudfold:::Err_exp(ANDRICH) # errors expected
obserr <- mudfold:::Err_obs(ANDRICH) # errors observed
hcoeft <- 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 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 \texttt{(HIDEOUS, DONTBELIEV, DETERRENT)} and its symmetric permutation \texttt{(DETERRENT, DONTBELIEV, HIDEOUS)}.

```r
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.

```r
## Compare H coefficients
hcoeft[triple_HDODE] == hcoeft[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 $\lambda_1$ 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 $\lambda_1$ is used as a threshold to determine if the triple is good enough to continue the scaling process. Larger values of $\lambda_1$ lead to more strict criteria while lower values of $\lambda_1$ 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 $\lambda_1$, larger values of $\lambda_2$ 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 $h, l, k \in \{1, 2, \ldots, N\}$, 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 \(\lambda_1\). 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 \(\lambda_1\). 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 to 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 \(\lambda_2\) (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 \(H_{hlk}\) coefficient greater than \(\lambda_2\).
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 $\lambda_1$.

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.

```r
## 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_irep <- 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.

```r
## 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_irep[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.

```r
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 $\lambda_2$ (default $\lambda_2 = 0$). We can extract all the triples for this specific scale using the `combinations()` function.

```r
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

```r
hcoeft[ExamplescaleTRIPLES[1:3, ]]
and we can see that the triple \textit{(HIDEOUS, INEFFECTIV, DETERRENT)} has a H coefficient which is lower than \( \lambda_2 \). 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 \( \lambda_2 \) to a negative number will allow more scales to pass this criterion, while setting \( \lambda_2 \) 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 \textit{MUSTHAVEIT} and one scale contains the item \textit{CRIMDESERV}. In this scenario the scale that contains the item \textit{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 \( \lambda_1 \) is the lower bound that have been used also in the first step of the item selection algorithm.

In the \textsc{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 \texttt{mudfold} package, which can be used to assess if a scale \( s \) consisting of \( m \) items, \( j = 1, \ldots, m \) conforms with the assumptions \textbf{A2} to \textbf{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 \textbf{A2} (local independence) holds, is equivalent to testing if the positive response on an item depends solely on the latent trait \( \theta \), i.e. \( P \left( X_j = 1 | \mathbf{X}_{-j}, \theta \right) = P \left( X_j = 1 | \theta \right) \). If \( p_j = P \left( X_j = 1 \right) \) denotes the probability of positive response to item \( j \), testing this hypothesis implies fitting the following regularized logistic regression model,

\[
\log \left( \frac{p_j}{1-p_j} \right) = \beta_0 + \sum_{k=1}^{m-1} \beta_k \mathbf{X}_{-jk} + \beta_\theta \hat{\theta},
\]

where \( \mathbf{X}_{-jk} \) denotes the \( k \)th column of \( \mathbf{X}_{-j} \) and \( \hat{\theta} = (\hat{\theta}_1, \ldots, \hat{\theta}_n) \) is a nonparametric estimate of the latent attitude with regression parameter \( \beta_\theta \). The response regression parameters \( \beta_k \) are penalized using the least absolute shrinkage and selection operator (LASSO, Tibshirani, 1996). LASSO shrinks the coefficients \( \beta_k \) of the regression in (4) towards zero. If \( \beta_k = 0 \) for all \( k = 1, \ldots, m \) then the local independence assumption if fulfilled and the probability of positive response on the item \( j \) depends only on \( \theta \). 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 \texttt{cv.glmnet()} that is available with the package \texttt{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 \( \theta \), that is,

\[
\log \frac{p_j}{1 - p_j} = \beta_0 + \beta_\theta f_\theta(\hat{\theta}), \tag{5}
\]

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 \( P(X_j = 1 \mid X_k = 1) \) 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,

\[
\text{CAM}_{jk} = \frac{\sum_{i=1}^{n} x_{ij} x_{ik}}{\sum_{i=1}^{n} x_{ik}} / \frac{\sum_{i=1}^{n} x_{ik}}{n}, \text{ for } j \neq k. \tag{6}
\]

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 \( \beta_j \). 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

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

where \( M_j \) is the position of the maximum in the jth 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 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. \( \text{MAX}_{\text{total}} = \)
The MAX statistic can be calculated using the function \texttt{MAX()} from the R package \texttt{mudfold}, which takes as input either a fitted MUDFOLD object obtained from the main \texttt{mudfold()} function, or an object of class "cam.mdf" calculated from the function \texttt{CAM()}. The argument ‘type’ of the \texttt{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 \texttt{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\(_{\text{total}}\)).

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.

\[
ISO_j = \sum_{h \leq k \leq m^*} \max \left(0, CAM_{jh} - CAM_{jk} \right) + \sum_{m^* \leq h \leq k} \max \left(0, CAM_{jk} - CAM_{jh} \right). \tag{8}
\]

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. ISO\(_{\text{total}} = \sum_{j=1}^{m} ISO_j\). 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 \texttt{ISO()}, which takes as input outputs either from the \texttt{mudfold()} function, or from the function \texttt{CAM()} and returns a vector with the ISO\(_j\)'s for each \(j \in \{1,2,\ldots,m\}\) or the sum of this vector if type = 'scale'.

All the diagnostic tests discussed in this section are implemented in the function \texttt{diagnostics()} of the \texttt{mudfold} package. The function \texttt{diagnostics()} can be used with fitted objects from the main \texttt{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 \(O_i(s), EO_i(s), H_i(s)\), and the total scale such as the \(O_{\text{total}}, EO_{\text{total}}, H_{\text{total}}\) are bootstrapped \(R\) times. The bootstrap procedure implemented in \texttt{mudfold} depends on the function \texttt{boot()} from the R package \texttt{boot} (Canty and Ripley, 2017). Using the \texttt{boot} package allows the user of \texttt{mudfold} package to obtain different types of confidence intervals for assessing uncertainty using the function \texttt{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 \( \hat{\theta}^\beta_i \) of the \( i \)-th respondent location parameter given a vector of known item location parameters \( \beta = (\beta_1, \beta_2, \ldots, \beta_m)^T \) was defined as,

\[
\hat{\theta}^\beta_i = \frac{\sum_{j=1}^{m} \beta_j x_{ij}}{\sum_{j=1}^{m} x_{ij}},
\]

where \( x_{ij} \) is the response of person \( i \) on item \( j \). The parameter estimate \( \hat{\theta}^\beta_i \) for each \( i \) takes values within the item parameter range. In MUDFOLD however, the item parameters vector \( \beta \) is unknown, thus we need to estimate it. In order to do so, we make use of two alternative estimates for \( \beta \)'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 \( \beta = (\beta_1, \beta_2, \ldots, \beta_m)^T \). Since MUDFOLD estimates only the rank order of the parameter vector, i.e. \( r = (r_1, r_2, \ldots, r_m)^T \) one can define a rank estimate

\[
\hat{\beta}_j^r = r_j,
\]

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,

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

Alternatively Johnson’s quantile estimator bounds both estimates for \( \theta \)'s and \( \beta \)'s within a unit interval. This estimator uses the item ranks divided by the length of the scale \( m \) as approximations for the \( \beta \) 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
for 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,

```r
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 (`estimation = "quantile"`), which is accessible by setting `estimation = "quantile"`.

- **lambda1**: The parameter $\lambda_1$, $0 \leq \lambda_1 \leq 1$ 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 $\lambda_1$ lead to more strict criteria in the item selection procedure.

- **lambda2**: Parameter $\lambda_2$, $-\infty < \lambda_2 \leq 1$ 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 \texttt{?mice:::quickpred} for more details. To be used when mice becomes problematic due to co-linear terms.

\ldots: Additional arguments to be passed into the boot() function (see \texttt{?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 non-parametric bootstrap with \( R = 100 \) iterations in parallel, you can specify it directly into the mudfold() function as follows.

```r
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 \texttt{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,

```r
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

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

and consists of three arguments:

- \texttt{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
interval, standard errors, biases, calculated from the bootstrap iterations for each diagnostic are given with the output.

default.

default.

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 mudfold 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 \( i = 1, 2, \ldots, n \) on \( N \) proximity items (indexed by \( j \)) with latent parameters \( \theta_i \) and \( \beta_j \) respectively. The vector of item parameters \( \beta = (\beta_1, \ldots, \beta_N) \) 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.

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 \( \theta_i \) and \( \beta_j \). We consider a linear transformation \( \tau_{ij} \) 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 \( \gamma_1 \) (deterministic parameter) and \( \gamma_2 \) (discrimination parameter) are fixed.

Using \( \tau_{ij} \) 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,

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

(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 mudfold 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 \( \gamma_1 \) or discriminative parameter of
the IRF. The higher the parameter the larger the number of items that individuals tend
to endorse if parameter \( \gamma_2 \) is kept constant.

**gamma2**: The deterministic parameter (i.e. \( \gamma_2 \)) 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 mudfold() 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 contain-
ing 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 \(\text{cutoff}\) should be \(\text{NULL}\) and vice versa.

cutoff: The numeric value(s) that will be used as thresholds for the transformation (default \(\text{cutoff}=NULL\)). Any value greater than or equal to the \(\text{cutoff}\) 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 \(\text{byItem}=\text{FALSE}\)) which imposes an explicit cut-off value for each individual in \(x\). If \(\text{byItem}=\text{TRUE}\) 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 \(\text{byItem}=\text{TRUE}\), the transformation is applied on the columns of \(x\). In the default \(\text{byItem}=\text{FALSE}\), the function "picks" items row-wise.

In the default parameter settings of the function \(\text{pick}\)(), the parameters \(k\) and \(\text{cutoff}\) respectively are equal to \(\text{NULL}\). In this case, the mean from \(N\) responses is used as a person-specific cut-off value (if \(\text{byItem}=\text{FALSE}\)). When \(\text{byItem}=\text{TRUE}\) (with \(k, \text{cutoff}\) equal to \(\text{NULL}\)) then the item mean over all individuals is used as an item specific cut-off value. The parameters \(k\) and \(\text{cutoff}\) 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 \(\text{NULL}\).

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 prefered 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 \text{mudfold} package. The first application is from the field of psychometrics while the second example is a linguistic application.

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

Loneliness data

In order to demonstrate the functionality of the \text{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.

\begin{itemize}
  \item A: There is always someone I can talk to about my day to day problems \(+\)
  \item B: I miss having a really close friend \(-\)
  \item C: I experience a general sense of emptiness \(-\)
  \item D: There are plenty of people I can lean on in case of trouble \(+\)
  \item E: I miss the pleasure of company of others \(-\)
  \item F: I find my circle of friends and acquaintances too limited \(-\)
  \item G: There are many people that I can count on completely \(+\)
  \item H: There are enough people that I feel close to \(+\)
  \item I: I miss having people around \(-\)
  \item J: Often I feel rejected \(-\)
  \item K: I can call on my friends whenever I need them \(+\)
\end{itemize}
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 ($n_{\text{miss}} = 69$). 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.

```r
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 $\lambda_1$ parameter in the `mudfold()` function is set to $\lambda_1 = 0.1$ since the default value leads to a minimal scale of length three.

```r
Lonelifit <- 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 `Lonelifit` object.

```r
loneliSummary <- summary(Lonelifit, boot = TRUE)
```

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

```r
loneliScale <- loneliSummary$ITEMSTATS$ITEM_DESCRIP$items
```

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 $H_{jk}$ lower than $\lambda_2$. 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.

```r
loneliSummary$SCALESTATS[1:3,]
```

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 ($ISO_{\text{total}} \approx 0.08$, $se = 0.459$) 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.

```r
par(mfrow = c(1, 2))
# testing for local independence

The R Journal Vol. 12/1, June 2020 ISSN 2073-4859
Local Independence

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.

diagnostics(lonelifit, which = "LI")
# visual inspection of moving maxima
diagnostics(lonelifit, which = "STAR")
par(mfrow=c(2,4))
# visual inspection for IRF unimodality
diagnostics(lonelifit, 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

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

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 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 \texttt{Plato7} data set. This dataset is available from the R package \texttt{smacof} (de Leeuw and Mair, 2009) and has been also included in the \texttt{mudfold} package. The data can be loaded into the R environment with the command \texttt{data("Plato7")}.

\texttt{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 \textit{Republic} and last the \textit{Laws}. In between \textit{Republic} and \textit{Laws}, Plato wrote the \textit{Critias}, \textit{Philebus}, \textit{Politicus}, \textit{Sophist} and \textit{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 \(\theta_i\) on a latent literary style continuum tends to prefer (appear most frequently in) the works of Plato with parameters \(\beta_j\) close to \(\theta_i\).

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 \texttt{any} out of \(N\) response process where the number of items “picked” varies across subjects. We can apply the transformation with the function \texttt{pick()} from the \texttt{mudfold} package in its default settings as follows.

\begin{verbatim}
dat.Plato <- pick(Plato7)
\end{verbatim}

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

We can check the MUDFOLD scale from the summary object.

```r
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   0.746     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 \( H_{\text{total}} = 0.56 \), 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.

```r
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       1.000        0.761    0.097   0.179     70
#>  H(Sophist) 0.41     0.381       1.000        0.656    0.241   0.157     70
#>  H(Politicus) 0.58    0.392       1.000        0.662    0.078   0.161     62
#>  H(Philebus) 0.63    0.429       1.000        0.726    0.094   0.141     83
#>  H(Laws) 0.51     0.394       1.000        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.

```r
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    0.018   0.18      65
#>  ISO(Philebus) 0.000    0           0.027       0.027    0.117   0.87      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 Plato unfolding scale can be visualized with

```r
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 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`. 

The R Journal Vol. 12/1, June 2020 ISSN 2073-4859
Summary

In this paper we introduced an R package named **mudfold** (Balafas et al., 2019). The latter is available under general public license (GPL ≥ 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 multivariate 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


Spyros E. Balafas
Bernoulli Institute for Mathematics, Computer Science & Artificial Intelligence
University of Groningen (RUG)
Bernoulliborg, Rn. 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
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’s 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.
• \textit{s\_size}, controls the \textit{anytime} algorithms. This is just a way to end the algorithm in a controlled manner because the \textit{anytime} algorithms can be stopped \textit{anytime} and the result will be returned.
• \textit{must\_dim}, is an optional argument for the \textit{mstomp} algorithm. See next item.
• \textit{exc\_dim}, as \textit{must\_dim}, is an optional argument for the \textit{mstomp} algorithm. These arguments control which dimensions must be included and which must be excluded from the multidimensional MP.
• \textit{n\_workers}, controls how many threads will be used for the \textit{stamp}, \textit{stomp}, and \textit{mstomp}. Note that for small datasets, multiple threads add an overhead that makes it slower than just one thread.
• .\texttt{keep\_data}, TRUE by default, keeps the input data inside the output object. This is useful for chained commands.

\textbf{Example data}

We think that the best and simple example to demonstrate the \textit{tsmp} package is the motif search.

The \textit{tsmp} package imports the \texttt{\%\%} (pipe) operator from the \textit{magrittr} package that makes the \textit{tsmp} workflow easier.

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

\begin{verbatim}
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()
\end{verbatim}

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

\textbf{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.

\textbf{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 \textit{anytime} property is achieved using a random approach where the best-so-far MP is computed using the DP that have been already calculated.

\textbf{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 \textit{tsmp} package) (Zhu et al., 2016). The \textit{anytime} property is achieved by randomly selecting elements from the best-so-far MP.

\textbf{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 \textit{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 naïvely, 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
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:
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
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
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
R> chains <- mp_gait_data %>% tsmp(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.

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.

**Speed**

While this new method for TS data mining is extremelly 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))
### 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.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>R Time $^*$</th>
<th>MATLAB Time $^*$</th>
<th>Threads</th>
</tr>
</thead>
<tbody>
<tr>
<td>scrimp</td>
<td>45.30</td>
<td>27.49</td>
<td>1</td>
</tr>
<tr>
<td>stomp</td>
<td>52.72</td>
<td>10.27</td>
<td>8</td>
</tr>
<tr>
<td>stomp</td>
<td>136.01</td>
<td>16.91</td>
<td>1</td>
</tr>
<tr>
<td>stamp</td>
<td>140.25</td>
<td>55.57</td>
<td>8</td>
</tr>
<tr>
<td>stamp</td>
<td>262.03</td>
<td>113.18</td>
<td>1</td>
</tr>
</tbody>
</table>

### 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., 2018; 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).

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. [p76]


*Francisco Bischoff*
CINTESIS - 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. Plácido Costa, s/n
4200-450 Porto, Portugal
ORCID: 0000-0002-5301-8672
fbischoff@med.up.pt

*Pedro Pereira Rodrigues*
CINTESIS - 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. Plácido Costa, s/n
4200-450 Porto, Portugal
ORCID: 0000-0001-7867-6682
pprodrigues@med.up.pt
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 disease 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 by 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 an 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 \( t = 1, 2, \ldots, t_{\text{end}} \), where (typically) \( t = 1 \) is the time when the first infection is observed and \( t_{\text{end}} \) 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 \( P(i, t) \) as follows:

\[
P(i, t) = 1 - \exp\{-\sum_{j \in I(t)} \Omega_T(j) \kappa(ij) - \epsilon\}, \quad \Omega_X(i) > 0, \quad \Omega_T(j) > 0, \quad \epsilon > 0
\]

where: \( \Omega_X(i) \) 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; \( \epsilon \) is a sparsk term which represents infections originating from outside the population being observed or some other unobserved infection mechanism; and \( \kappa(ij) \) 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), \( \Omega_X(i) \) is treated as a linear function of the covariates, i.e., \( \Omega_X(i) = a_0 + a_1 X_1(i) + a_2 X_2(i) + \cdots + a_n X_n(i) \), where \( X_1(i), \ldots, X_n(i) \) denote \( n \) covariates associated with susceptible individual \( i \), along with susceptibility parameters \( a_0, \ldots, a_n > 0 \). Note that, if the model does not contain any susceptibility covariates then \( \Omega_X(i) = a_0 \) 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., \( \Omega_T(j) = \phi_1 X_1(j) + \phi_2 X_2(j) + \cdots + \phi_n X_n(j) \), where \( X_1(j), \ldots, X_n(j) \) denote \( n \) covariates associated with infectious individual \( j \), along with transmissibility parameters \( \phi_1, \ldots, \phi_n > 0 \). 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 \( \kappa(ij) \): 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 \( \beta \) 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 \( C_{ij}^{(j)} \) 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 \( \beta(j) \)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 \( C_{ij}^{(j)} \in \mathbb{R}^+ \) or \( C_{ij}^{(j)} \in [0,1] \) 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
\[ f(S, I, R|\theta) = \prod_{t=1}^{t_{\text{max}}} f_t(S, I, R|\theta) \]
(2)
where
\[ f_t(S, I, R|\theta) = \left[ \prod_{i \in I(t+1) \setminus I(t)} \mathbb{P}(i,t) \right] \left[ \prod_{i \in S(t+1)} (1 - \mathbb{P}(i,t)) \right] \]
(3)
and where, \( \theta \) 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 \( t_{\text{max}} \leq t_{\text{end}} \) 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 \( \text{Sformula} = \sim \text{model} \) is used to specify the covariate information, separated by + and - operators similar to the R generic function formula(). For example, \( \Omega_S(i) = \alpha_0 + \alpha_1 X(i), \) \( i = 1, \ldots, n \) can be passed through the argument Sformula as Sformula = \( \sim \) 1 + X. In a similar way, the user can control the transmissibility function \( \Omega_T(i) \) through the 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 = \( \sim \) -1 + X. The spatial (or network), susceptibility, transmissibility, and spark (if any) parameters are passed through arguments beta, alpha, phi, and spark, respectively.
Table 1: Description of functions and their output in the EpiILM package

<table>
<thead>
<tr>
<th>Function</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>epiBR0</td>
<td>Calculates the basic reproduction number for a specified SIR model</td>
</tr>
<tr>
<td>epidata</td>
<td>Simulates epidemic for the specified model type and parameters</td>
</tr>
<tr>
<td>plot.epidata</td>
<td>Produces spatial plots of epidemic progression over time as well as various epidemic curves of epidata object</td>
</tr>
<tr>
<td>epidic</td>
<td>Computes the deviance information criterion for a specified individual-level model</td>
</tr>
<tr>
<td>epilike</td>
<td>Calculates the log-likelihood for the specified model and data set</td>
</tr>
<tr>
<td>epimcmc</td>
<td>Runs an MCMC algorithm for the estimation of specified model parameters</td>
</tr>
<tr>
<td>summary.mcmc</td>
<td>Produces the summary of epimcmc object</td>
</tr>
<tr>
<td>plot.mcmc</td>
<td>Plots epimcmc object</td>
</tr>
<tr>
<td>pred.epi</td>
<td>Computes posterior predictions for a specified epidemic model</td>
</tr>
<tr>
<td>plot.pred.epi</td>
<td>Plot posterior predictions</td>
</tr>
</tbody>
</table>

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 consists 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 analysis 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 \( \epsilon = 0 \), and \( t_{\text{max}} = 15 \), the model is given by:

\[
P(i,t) = 1 - \exp\left\{-0.3 \sum_{j \in I(t)} d_{ij}^{-5}\right\}, \ t = 1, \ldots, 15
\]

where \( d_{ij} \) is the Euclidean distance between individuals \( i \) and \( j \), with their locations specified through \( x \) and \( y \).

First, we install the \texttt{EpiILM} package and call the library.

```R
R> install.packages("EpiILM")
R> library("EpiILM")
```

Then, let us simulate \((x, y)\) coordinates uniformly across a \(10 \times 10\) unit square.

```R
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
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 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 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 \( t_{\text{max}} = 15 \). 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 \( \text{SI.dis$inftime} \) provides the times at which individuals enter the infectious state, with 0 representing individuals who are still susceptible at time \( t_{\text{max}} \). Figures 1 and 2 show the summary graphics for the simulated epidemic, which are produced using the S3 method \texttt{plot.epidata()} as follows:

```R
R> plot(SI.dis, plottype = "curve", curvetype ="complete")
R> plot(SI.dis, plottype = "spatial")
```
**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:

\[
C_{ij} = \begin{cases} 
1 & \text{if a directed edge exists between } i \text{ and } j \\
0 & \text{otherwise}
\end{cases}
\]

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

\[
P(i, t) = 1 - \exp\left\{-(\alpha_0 + \alpha_1 Z_i) \sum_{j \in I(t)} C_{ij}\right\}, \quad t = 1, \ldots, t_{\text{max}}
\]

where \( \alpha_0 \) is the baseline susceptibility and \( \alpha_1 \) 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 \( t_{\text{max}} = 15 \). We simulate a directed
Figure 2: Simulated epidemic from (4) for 100 individuals, where open circles represent susceptible individuals and filled red circles represent infected individuals.
network using the following code:

```r
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
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)](image)

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
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 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

The R Journal Vol. 12/1, June 2020 ISSN 2073-4859
Figure 4 shows the epidemic curve, obtained via the following code.

\[
\text{R> plot(SI.contact, plottype = "curve", curvetype ="complete")}
\]

![Epidemic Curves](image)

**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 argument spark, 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

$$I_P(i,t) = 1 - \exp\left\{-\left(\alpha_0 + \alpha_1 A(i)\right) \sum_{j \in I(t)} d_{ij}^{-\beta}\right\}, \quad t = 1, \ldots, t_{\text{max}}, \quad (7)$$

where the susceptibility covariate $A$ represents the number of animals on each farm, $\alpha_0$ is the baseline susceptibility, $\alpha_1$ is the number of animals effect, and $\beta$ 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 $t_{\text{max}} = 50$. Considering an SI compartmental framework for this situation, the epidemic is simulated using the following command:

```r
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
R> t_end <- max(SI.dis.cov$inftime)
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 $(\alpha_0, \alpha_1)$ and $\beta$ 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 $\alpha_0, \alpha_1$, and $\beta$. 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
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
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:

<table>
<thead>
<tr>
<th></th>
<th>Mean</th>
<th>SD</th>
<th>Naive SE</th>
<th>Time-series SE</th>
</tr>
</thead>
<tbody>
<tr>
<td>alpha.1</td>
<td>0.2868</td>
<td>0.2351</td>
<td>0.0011758</td>
<td>0.012449</td>
</tr>
<tr>
<td>alpha.2</td>
<td>0.1997</td>
<td>0.07544</td>
<td>0.0003772</td>
<td>0.002393</td>
</tr>
<tr>
<td>beta.1</td>
<td>5.6790</td>
<td>0.48188</td>
<td>0.0024094</td>
<td>0.014631</td>
</tr>
</tbody>
</table>

2. Quantiles for each variable:

<table>
<thead>
<tr>
<th></th>
<th>2.5%</th>
<th>25%</th>
<th>50%</th>
<th>75%</th>
<th>97.5%</th>
</tr>
</thead>
<tbody>
<tr>
<td>alpha.1</td>
<td>0.02018</td>
<td>0.1141</td>
<td>0.2226</td>
<td>0.3917</td>
<td>0.9072</td>
</tr>
<tr>
<td>alpha.2</td>
<td>0.08680</td>
<td>0.1486</td>
<td>0.1875</td>
<td>0.2395</td>
<td>0.3769</td>
</tr>
<tr>
<td>beta.1</td>
<td>4.81361</td>
<td>5.3377</td>
<td>5.6704</td>
<td>5.9872</td>
<td>6.6588</td>
</tr>
</tbody>
</table>

Figure 5: MCMC traceplot for the estimation of model (7) parameters
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:

\[
P(i, t) = 1 - \exp\left\{-\left(\alpha_1 X_1(i) + \alpha_2 X_2(i)\right) \sum_{j \in I(t)} \left[\phi_1 X_1(j) + \phi_2 X_2(j)\right] C_{ij}\right\}, \quad t = 1, \ldots, t_{\text{max}} \tag{8}
\]

with susceptibility parameters, \((\alpha_1, \alpha_2)\), transmissibility parameters, \((\phi_1, \phi_2)\), 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
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
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 \(t_{\text{max}} = 25\), \(\alpha_1 = 0.003\), \(\alpha_2 = 0.01\), \(\phi_1 = 0.0003\), 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
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 gamma prior distribution for the model parameters \((\alpha_2, \phi_1, \phi_2)\) while fixing \(\alpha_1\) 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
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, 0.01), trans.par.ini = c(0.0003, 0.0002),
+                                Sformula = ~ -1 + X1 + X2, Tformula = ~ -1 + X1 + X2,
+                                prior_sus.dist = "gamma", prior_trans.dist = "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 \((\alpha_2, \phi_1, \phi_2)\) 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 \(\hat{\phi}_2 = 0.0002\) \((0.00006, 0.0003)\).
**Figure 6:** Epidemic curve of the contact network model (8) for 500 farms

R> summary(mcmcout_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:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Mean</th>
<th>SD</th>
<th>Naive SE</th>
<th>Time-series SE</th>
</tr>
</thead>
<tbody>
<tr>
<td>alpha.2</td>
<td>0.0093899</td>
<td>0.0030070</td>
<td>1.504e-05</td>
<td>1.866e-04</td>
</tr>
<tr>
<td>phi.1</td>
<td>0.0004123</td>
<td>0.0001218</td>
<td>6.089e-07</td>
<td>4.483e-06</td>
</tr>
<tr>
<td>phi.2</td>
<td>0.0001851</td>
<td>0.0000732</td>
<td>3.660e-07</td>
<td>1.911e-06</td>
</tr>
</tbody>
</table>

2. Quantiles for each variable:
Figure 7: MCMC traceplot for the estimation of model (8) parameters.

<table>
<thead>
<tr>
<th>Variable</th>
<th>2.5%</th>
<th>25%</th>
<th>50%</th>
<th>75%</th>
<th>97.5%</th>
</tr>
</thead>
<tbody>
<tr>
<td>alpha.2</td>
<td>5.130e-03</td>
<td>0.0071826</td>
<td>0.0088076</td>
<td>0.0110550</td>
<td>0.0168404</td>
</tr>
<tr>
<td>phi.1</td>
<td>2.123e-04</td>
<td>0.0003226</td>
<td>0.0004020</td>
<td>0.0004879</td>
<td>0.0006800</td>
</tr>
<tr>
<td>phi.2</td>
<td>6.646e-05</td>
<td>0.0001322</td>
<td>0.0001763</td>
<td>0.0002299</td>
<td>0.0003463</td>
</tr>
</tbody>
</table>

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.
et al., 2005, Pokharel and Deardon, 2016). Thus, an SIR model is fitted using the following code:

```r
R> data(tswv)
R> x <- tswv$x
R> y <- tswv$y
R> inftime <- tswv$inftime
R> removaltime <- tswv$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
R> epidat.tswv <- as.epidata(type = "SIR", n = 520, x = x, y = y,
+     inftime = inftime, infperiod = infperiod)
R> plot(epidat.tswv, plottype = "spatial", tmin = 2)
```

![Temporal plot](image)

**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 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 $\alpha$ and $\beta$, 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 $\alpha$ and $\beta$. 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
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 $\alpha$ and $\beta$ 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.

![Trace of alpha.1](image1.png) ![Trace of beta.1](image2.png)

**Figure 9:** MCMC traceplots of the posterior samples for TSWV data

**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).

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


**Vineetha Warriyar. K. V.**  
Faculty of Veterinary Medicine  
University of Calgary  
Canada  
vineethawarriyar.kod@ucalgary.ca

**Waled 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 $i = 1, \ldots, n$, denote by $T_i$ the observed time of event or censoring for subject $i$, and $\delta_i$ 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 $j = 1, \ldots, p$, let $x_{ij}$ be the covariate for subject $i$. For stratum $g = 1, \ldots, G$, 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 $\beta$ 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 assumption; one approach is to allow the baseline hazard to differ by group across the observations. 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.

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 $\beta = (\beta_1, \ldots, \beta_p)^T$, $X_i = (X_{i1}, \ldots, X_{ip})^T$ and $R_{i,g} = \{ \ell : T_\ell \geq T_i, g_\ell = g \}$ 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 $S_{k,g}(i,j) = \sum_{\ell \in R_{k,g}} X_{\ell,j} \exp \{ X_{\ell,g}^T \beta \}$ for $k = 0, 1, 2$.

<table>
<thead>
<tr>
<th>Data: ${T_i, \delta_i, g_i, X_i}_{i=1}^n$; Number of iterations $M$; Updating rate $v$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Result:</strong> $\hat{\beta}$</td>
</tr>
<tr>
<td><strong>begin</strong></td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td><strong>end</strong></td>
</tr>
<tr>
<td><strong>end</strong></td>
</tr>
</tbody>
</table>

**Algorithm 1:** 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 $\beta$ 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. Buhlmann and Hothorn’s algorithm uses only the first derivative to update the estimated $\beta$ 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.

$$BIC = -2 \left\{ l_j(\hat{\theta}) - l_0(\hat{\theta}_0) \right\} + (p_j - p_0) \log(d),$$

where $l_j(\hat{\theta})$ 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

$$EBIC = -2 l_j(\hat{\theta}) + p_j \log(d) + 2 \gamma \log \left( \frac{p}{p_j} \right),$$

where $(\frac{p}{p_j})$ is the size of the class of models that model $j$ belongs to, $p$ is the total number of variables. The value of $\gamma$ 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 $\beta$ selected variables. The number of uncensored events is $d$.

$$CV_k(m) = -[\ell(\beta_{-k}(m)) - \ell(\beta_{-k}(m))],$$

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.

$$\Delta \ell = -[\ell(\beta(m)) - \ell(\beta(m + 1))] < \alpha,$$

Where $\alpha$ 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.

<table>
<thead>
<tr>
<th>stopping method</th>
<th>number selected</th>
<th>Se</th>
<th>Sp</th>
<th>FDR</th>
<th>MSE</th>
<th>number of iterations</th>
<th>runtime (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SurvBoost fixed</td>
<td>111 (5)</td>
<td>0.90 (.02)</td>
<td>0.99 (.00)</td>
<td>0.19 (.04)</td>
<td>382 (1)</td>
<td>500 (0)</td>
<td>163 (20)</td>
</tr>
<tr>
<td>mboost</td>
<td>99 (4)</td>
<td>0.82 (.03)</td>
<td>1.00 (.00)</td>
<td>0.17 (.04)</td>
<td>387 (1)</td>
<td>500 (0)</td>
<td>396 (254)</td>
</tr>
<tr>
<td>SurvBoost cv</td>
<td>379 (24)</td>
<td>1.00 (.00)</td>
<td>0.93 (.01)</td>
<td>0.74 (.02)</td>
<td>333 (3)</td>
<td>3896 (275)</td>
<td>5742 (595)</td>
</tr>
<tr>
<td>SurvBoost # selected</td>
<td>101 (0)</td>
<td>0.84 (.03)</td>
<td>1.00 (.00)</td>
<td>0.17 (.03)</td>
<td>385 (2)</td>
<td>385 (40)</td>
<td>163 (24)</td>
</tr>
<tr>
<td>SurvBoost likelihood</td>
<td>121 (6)</td>
<td>0.95 (.02)</td>
<td>0.99 (.00)</td>
<td>0.21 (.04)</td>
<td>378 (1)</td>
<td>632 (15)</td>
<td>242 (33)</td>
</tr>
<tr>
<td>SurvBoost EBIC</td>
<td>140 (7)</td>
<td>0.99 (.01)</td>
<td>0.99 (.00)</td>
<td>0.29 (.04)</td>
<td>370 (1)</td>
<td>993 (10)</td>
<td>624 (60)</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Stopping Method</th>
<th>number selected</th>
<th>Se</th>
<th>Sp</th>
<th>FDR</th>
<th>MSE</th>
<th>number of iterations</th>
<th>runtime (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>fixed</td>
<td>116 (7)</td>
<td>0.71 (.04)</td>
<td>0.95 (.01)</td>
<td>0.39 (.04)</td>
<td>49 (0)</td>
<td>500 (0)</td>
<td>14 (1)</td>
</tr>
<tr>
<td>Stratified</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fixed</td>
<td>198 (16)</td>
<td>0.90 (.04)</td>
<td>0.88 (.02)</td>
<td>0.54 (.03)</td>
<td>50 (2)</td>
<td>1585 (316)</td>
<td>263 (60)</td>
</tr>
<tr>
<td>cv</td>
<td>100 (0)</td>
<td>0.65 (.04)</td>
<td>0.96 (.00)</td>
<td>0.36 (.04)</td>
<td>50 (1)</td>
<td>391 (42)</td>
<td>11 (2)</td>
</tr>
<tr>
<td># selected</td>
<td>112 (8)</td>
<td>0.69 (.04)</td>
<td>0.95 (.01)</td>
<td>0.38 (.03)</td>
<td>49 (1)</td>
<td>472 (33)</td>
<td>14 (2)</td>
</tr>
<tr>
<td>likelihood</td>
<td>161 (9)</td>
<td>0.84 (.03)</td>
<td>0.91 (.01)</td>
<td>0.48 (.04)</td>
<td>44 (1)</td>
<td>41 (45)</td>
<td>29 (3)</td>
</tr>
<tr>
<td>EBIC</td>
<td>122 (7)</td>
<td>0.67 (.04)</td>
<td>0.94 (.01)</td>
<td>0.45 (.04)</td>
<td>49 (1)</td>
<td>500 (0)</td>
<td>13 (1)</td>
</tr>
<tr>
<td>Unstratified</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SB fixed</td>
<td>58 (5)</td>
<td>0.41 (.03)</td>
<td>0.98 (.00)</td>
<td>0.30 (.06)</td>
<td>53 (0)</td>
<td>500 (0)</td>
<td>13 (1)</td>
</tr>
</tbody>
</table>

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.

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 $\beta$ 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.

<table>
<thead>
<tr>
<th>stopping method</th>
<th>number selected</th>
<th>Se</th>
<th>Sp</th>
<th>FDR</th>
<th>MSE</th>
<th>number of iterations</th>
<th>runtime (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SurvBoost fixed</td>
<td>104 (3)</td>
<td>0.94 (.02)</td>
<td>1.00 (.00)</td>
<td>0.10 (.02)</td>
<td>381 (.45)</td>
<td>500 (0)</td>
<td>138 (11)</td>
</tr>
<tr>
<td>mboost</td>
<td>98 (4)</td>
<td>0.89 (.03)</td>
<td>1.00 (.00)</td>
<td>0.10 (.03)</td>
<td>384 (.36)</td>
<td>500 (0)</td>
<td>220 (198)</td>
</tr>
</tbody>
</table>

| 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.
Table 4: Stopping criteria options for boosting_core function.

<table>
<thead>
<tr>
<th>Call</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>boosting_core(formula, data)</td>
<td>fixed mstop = 500</td>
</tr>
<tr>
<td>boosting_core(formula, data, num_iter=1000)</td>
<td>fixed mstop = specified value</td>
</tr>
<tr>
<td>boosting_core(formula, data, control_method=&quot;cv&quot;)</td>
<td>10-fold cross validation</td>
</tr>
<tr>
<td>boosting_core(formula, data, control_method=&quot;num_selected&quot;,</td>
<td>number selected, need to specify number of variables</td>
</tr>
<tr>
<td>control_parameter = 5)</td>
<td></td>
</tr>
<tr>
<td>boosting_core(formula, data, control_method=&quot;likelihood&quot;)</td>
<td>change in likelihood</td>
</tr>
<tr>
<td>boosting_core(formula, data, control_method=&quot;BIC&quot;)</td>
<td>minimum BIC or EBIC</td>
</tr>
<tr>
<td>boosting_core(formula, data, control_method=&quot;AIC&quot;)</td>
<td>minimum AIC</td>
</tr>
</tbody>
</table>

Table 5: Functions available in SurvBoost package. Every function accepts a boosting object input to generate the corresponding result.

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>summary.boosting()</td>
<td>prints summary of variable selection and estimation</td>
</tr>
<tr>
<td>plot.boosting()</td>
<td>plots variable selection frequency</td>
</tr>
<tr>
<td>predict.boosting()</td>
<td>generates predicted hazard ratio for each observation or new data</td>
</tr>
<tr>
<td>post.selection.fitting.boosting()</td>
<td>refits model with only subset of selected covariates</td>
</tr>
</tbody>
</table>

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
R > strata.boosting(data_small$strata_idx, data_small$time)
```

```r
table <- data.frame(
  as.factor(x) = c(1, 2, 3, 4, 5),
  Min = c(0.004677274, 0.005600448, 0.005794314, 0.004251120, 0.001534922),
  Q1 = c(0.1163388, 0.1422992, 0.1371938, 0.1998902, 0.1283325),
  Median = c(0.3108169, 0.5849665, 0.9125127, 1.270754, 0.5896426),
  Q3 = c(1.096236, 1.270754, 1.314191, 1.437124, 1.325094),
  Max = c(1.693283, 1.951286, 1.989180, 1.960646, 1.873137)

table
```

The R Journal Vol. 12/1, June 2020 ISSN 2073-4859
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
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
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.](image-url)
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  | 0.05783  | 6.028 | 1.42e-14 *** |

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

Figure 3: Plot generated by `plot.boosting` function with option “coefficients,” coefficient paths for variables selected by number of boosting iterations.
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")
```

![Survival curves for the two strata based on cancer stage and tumor size.](image)

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.
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).

**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**


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. Isai (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\url{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 \( Z \) 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 \( T \) follows the Cox model,

\[
\lambda(t \mid z) = \lambda_0(t) \exp \left( \beta^\top z \right),
\]

where \( \beta \) 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 \( Z \), and the distribution of \( C \) is independent of \( Z \).

**Figure 2**: A diagram of the right-censored length-biased data.

The length-biased data consist of \( \{(Y_i, A_i, \delta_i, Z_i), i = 1, \ldots, n\} \), 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 \( \text{Cov}(T, A + C \mid Z) = \text{Var}(A \mid Z) + \text{Cov}(A, V \mid Z) > 0 \).
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 $a^0 = 1$, $a^1 = a$, and $a^2 = aa^*$ for any vector $a$. Define

$$S^{(k)}(\beta, t) = n^{-1} \sum_{i=1}^{n} \hat{w}_C(t) R_i(t) \{w_C(Y_i)\}^{-1} Z_i^k \exp(\beta^T Z_i),$$

where the weight function $\hat{w}_C(y) = \int_0^y \hat{S}_C(u) du$, in which $\hat{S}_C(y) = \Pr(C > y)$ is the survival function of the residual censoring variable $C$ for $k = 0, 1, \text{and} 2$. By replacing $\hat{w}_C(y)$ with its consistent estimator, $\hat{w}_C(y) = \int_0^y \hat{S}_C(u) du$, where $\hat{S}_C(\cdot)$ is the Kaplan–Meier estimator of the residual censoring survival function, we have

$$\hat{S}^{(k)}(\beta, t) = n^{-1} \sum_{i=1}^{n} \hat{w}_C(t) R_i(t) \{\hat{w}_C(Y_i)\}^{-1} Z_i^k \exp(\beta^T Z_i)$$

for $k = 0, 1, \text{and} 2$. The regression parameter $\beta_0$ can be estimated by solving the following unbiased estimating equation:

$$\hat{U}(\beta) = \sum_{i=1}^{n} \int_0^\tau (Z_i - \hat{E}(\beta, u))dN_i(u) = 0,$$

where $\tau$ satisfies $\Pr(Y \geq \tau) > 0$ and $\hat{E}(\beta, t) = \hat{S}^{(1)}(\beta, t)/\hat{S}^{(0)}(\beta, t)$. The solution to Equation (2), which is denoted by $\hat{\beta}$, is unique and a consistent estimator to $\beta_0$. Using the Taylor series expansion, it can be shown that the distribution of $\hat{\beta}$ converges weakly to a normal distribution with variance $\Gamma^{-1} \Sigma \Gamma^{-1}$, where $\Gamma = \lim_{n \to \infty} n^{-1} \hat{U}(\beta) / \hat{\beta} \hat{\beta}^T$ and $\Sigma$ is the covariance matrix of $\lim_{n \to \infty} n^{-1/2} \hat{U}(\beta_0)$.

This estimation procedure can be implemented using the coxphlb function in the CoxPhLb package as follows:

```r
coxphlb(formula, data, method = c("Bootstrap","EE"), boot.iter = 500,
    seed.n = round(runif(1,1,1e09)), digits = 3)
```

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(\{\hat{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,

\[ M_i(t) = N_i(t) - \int_0^t w_C(u)R_i(u)\{w_C(Y_i)\}^{-1}\exp(\beta_0^T Z_i)\,d\Lambda_0(u), \]

for \( i = 1, \ldots, n \), where \( \Lambda_0(t) = \int_0^t \lambda_0(s)\,ds \) is the cumulative baseline hazard function. The stochastic process can be estimated by

\[ \hat{M}_i(t) = N_i(t) - \int_0^t \hat{w}_C(u)R_i(u)\{\hat{w}_C(Y_i)\}^{-1}\exp(\hat{\beta}^T Z_i)\,d\hat{\Lambda}_0(\hat{\beta}, u), \]

where

\[ \hat{\Lambda}_0(\hat{\beta}, u) = \int_0^t \frac{\sum_{i=1}^n dN_i(u)}{nS^{(0)}(\hat{\beta}, u)}. \]

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 \( \hat{M}_i(t), i = 1, \ldots, n \) deviate from zero systematically, it may be a sign of model misspecification.

Let

\[ G(t, z) = \sum_{i=1}^n f(Z_i)I(Z_i \leq z)\hat{M}_i(t), \tag{3} \]

where \( f(\cdot) \) is a prespecified smooth and bounded function, and \( I(Z_i \leq z) = I(Z_i \leq z_1, \ldots, Z_i \leq z_p) \) with \( Z_i = (Z_{i1}, \ldots, Z_{ip})^T \) and \( z = (z_1, \ldots, z_p)^T \). 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 \( z = \infty \). To develop analytical test procedures, test statistics can be constructed using the supremum test, \( \sup_{\hat{\beta}}G(t, z) \). Let \( T_1^j = \sup_{\hat{\beta}}G_j^1(\hat{\beta}) \) be the test statistics for checking the functional form of the \( j \)th covariate, where \( G_j^1(\hat{\beta}) = \sum_{i=1}^n I(Z_{ij} \leq z)\hat{M}_i(\tau) \); and \( T_2^j = \sup_{\hat{\beta}}G_j^2(\hat{\beta}) \), where \( G_j^2(\hat{\beta}) = \sum_{i=1}^n I(Z_{ij} \leq z)\hat{M}_i(t) \) for checking the proportional hazards assumption for the \( j \)th covariate. We can also consider the global test statistic \( T_2 = \sup_\beta \sum_{j=1}^p |G_j^2(\beta)| \) 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

\[ S_Z^{(l)}(\beta, t, z) = n^{-1} \sum_{i=1}^n f(Z_i)I(Z_i \leq z)\hat{w}_C(t)R_i(t)\{\hat{w}_C(Y_i)\}^{-1}Z_i \exp(\beta^T Z_i), \]

for \( l = 0, 1 \), \( \tilde{E}_Z(\beta, u, z) = S_Z^{(0)}(\beta, t, z)/\hat{S}_Z^{(0)}(\beta, t) \),

\[ \tilde{H}(\beta, t) = \sum_{i=1}^n \sum_{k=1}^n f(Z_i)I(Z_k \leq z)\hat{w}_C(Y_i)R_k(Y_i)\exp(\beta^T Z_k)\{\hat{w}_C(Y_k)\}^{-1}Z_k \exp(\beta^T Z_k), \]

\[ \hat{\Lambda}_C(t) = I(V_i \leq t, \delta_i = 0) - \int_0^t I(V_i \geq u)\,d\hat{\Lambda}_C(u), \]

\[ \hat{h}_k(t) = I(Y_k \geq t) \int_0^{Y_k} \hat{S}_C(u)\,du, \]

\[ \bar{\beta}(t) = \hat{S}_C(t)\hat{S}_V(t), \]

in which \( \hat{\Lambda}_C(\cdot) \) is the Nelson-Aalen estimator for the residual censoring time and \( \hat{S}_V(\cdot) \) is the Kaplan-
Meier estimator of the residual survival time, and
\[
\hat{G}_m(t, z) = \frac{1}{n} \sum_{i=1}^{n} \left[ \frac{\hat{S}_Z^{(1)}(\beta, u, z) \hat{S}_Z^{(0)}(\beta, u)}{(\hat{S}_Z^{(0)}(\beta, u))^2} \right] dN_i(u),
\]

Define \(\tilde{G}_m(t, z) = \sum_{i=1}^{n} \tilde{G}_i(t, z) V_{mi}\), where \(V_{mi}, i = 1, \ldots, n\), are independent random variables sampled from a standard normal distribution for \(m = 1, \ldots, M\). The simulated realizations of \(\tilde{G}_m(t, z)\) for a large \(M\) approximate the null distribution. For graphical assessment, we can plot a few randomly chosen realizations of \(\tilde{G}_m(t, z)\) 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 \(\sup_{t, z} |\tilde{G}_m(t, z)|\) for \(m = 1, \ldots, M\). 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:

```r
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 at 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:

```r
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:

```r
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 \(F(t) = \text{Pr}(A \leq t)\) and \(G(t) = \text{Pr}(V^* \leq t, \delta = 1)\), where
\[ V^* = \min(V, C). \] Following Wei (1980), the test statistic can be constructed as follows:

\[ W = n^{-2} \sum_{i=1}^{n} \sum_{j=1}^{n} \left\{ \Phi(A_i, V^*_j, \delta_j) - p \right\}, \]

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 \( \text{station.test.plot} \) as follows:

\( \text{station.test.plot}(a,v,delta) \)

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

\( \text{station.test}(a,v,delta,digits = 3L) \)

where the data input arguments are the same as those in the \( \text{station.test.plot} \) 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 \( \text{coxphlb} \), (ii) model checking using functions \( \text{coxphlb.ftest} \) and \( \text{coxphlb.phtest} \), and (iii) stationarity assumption testing using function \( \text{station.test} \). 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.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>S3 methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{coxphlb} )</td>
<td>Fits a Cox model to right-censored length-biased data</td>
<td>print()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>summary()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>coef()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>vcov()</td>
</tr>
<tr>
<td>( \text{coxphlb.ftest} )</td>
<td>Tests the functional form of covariates</td>
<td>print()</td>
</tr>
<tr>
<td>( \text{coxphlb.phtest} )</td>
<td>Tests the proportional hazards assumption</td>
<td>print()</td>
</tr>
<tr>
<td>( \text{station.test} )</td>
<td>Tests the stationarity assumption</td>
<td>print()</td>
</tr>
<tr>
<td>( \text{coxphlb.ftest.plot} )</td>
<td>Returns a graph for testing the functional form of covariates</td>
<td>-</td>
</tr>
<tr>
<td>( \text{coxphlb.phtest.plot} )</td>
<td>Returns a graph for testing the proportional hazards assumption</td>
<td>-</td>
</tr>
<tr>
<td>( \text{station.test.plot} )</td>
<td>Returns a graph for testing the stationarity assumption</td>
<td>-</td>
</tr>
</tbody>
</table>

**Table 1:** Summary of functions in the CoxPhLb 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 \( x_1 \) and \( X_2 \) with continuous values that range from 0 to 1 \( x_2 \). 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.

```r
> data("ExampleData1", package = "CoxPhLb")
> dat1 <- ExampleData1
> station.test.plot(a = dat1$a, v = dat1$y - dat1$a, delta = dat1$delta)
```

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.

```r
> station.test(a = dat1$a, v = dat1$y - dat1$a, delta = dat1$delta)
```

```
<table>
<thead>
<tr>
<th>test.statistic</th>
<th>p.value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-0.375</td>
<td>0.707</td>
</tr>
</tbody>
</table>
```

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.

```r
> 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.err 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.err  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)

![](image)

**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 hazards assumption is reasonable for both covariates. This is further confirmed by the computed p values, which are greater than 0.05.
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.

```r
> 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.

```r
> 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.

```r
> station.test(a = (dat2$ageentry - 65), v = (dat2$age - dat2$ageentry),
+ delta = dat2$death)
```

<table>
<thead>
<tr>
<th>test.statistic</th>
<th>p.value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.261</td>
<td>0.794</td>
</tr>
</tbody>
</table>

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.
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.

```r
> 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.err z.score p.value lower.95 upper.95
  gender -0.112 0.028 0.168 -0.67 0.5 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.

```r
> phtest2 <- coxphlb.phtest(fit = fit.ee2, data = dat2, spec.p = 1, seed.n = 1234)
> 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 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


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
SortedEffects: Sorted Causal Effects in R
by Shuowen Chen, Victor Chernozhukov, Iván Fernández-Val and Ye Luo

Abstract Chernozhukov et al. (2018) proposed the sorted effect method for nonlinear regression models. This method consists of reporting percentiles of the partial effects, the sorted effects, in addition to the average effect commonly used to summarize the heterogeneity in the partial effects. They also propose to use the sorted effects to carry out classification analysis where the observational units are classified as most and least affected if their partial effect are above or below some tail sorted effects. The R package SortedEffects implements the estimation and inference methods therein and provides tools to visualize the results. This vignette serves as an introduction to the package and displays basic functionality of the functions within.

The sorted effects method

Many empirical questions in econometrics, machine learning and statistics boil down to studying how changes in a key variable $T$ affect an outcome variable of interest $Y$, holding fixed some control variables $W$. Such effects are called predictive partial effects (PEs), or treatment or structural partial effects when they have a causal or structural interpretation. Depending on the context, researchers often work with models that feature nonlinearity in the key variable of interest or nonlinearity in parameters. The first kind of models includes mean and quantile regressions in which $T$ is interacted with $W$, while the second type includes generalized linear models such as logit and probit. The methods implemented in SortedEffects are designed to estimate and make inference on PEs in nonlinear models.

In nonlinear models the PEs vary with the underlying control variables. Consider the probit model as an example. The conditional probability of $\{Y = 1\}$ is

$$\Pr (Y = 1 \mid T = t, W = w) = \Phi (t\beta + w'\gamma) ,$$

where $\Phi$ denotes CDF of standard normal distribution. Then, the PE of changing $T$ on $Y$, holding $W$ fixed at $w$, is

$$\Phi (t\beta + w'\gamma) - \Phi (w'\gamma)$$

when $T$ is binary with values 0 and 1, or

$$\phi (t\beta + w'\gamma) \beta, \quad \phi (u) = \partial \Phi (u) / \partial u ,$$

when $T$ is continuous. Since $W$ typically differ among observational units, the PEs are heterogeneous.

More generally, suppose we have a regression function $g (X)$ corresponding to some characteristic of $Y$ conditional on the covariates $X := [T, W]$. Let $\Delta (x)$ denote the PE of $T$ on $Y$, holding $W$ fixed. Then,

$$\Delta (x) = g (t_1, w) - g (t_0, w)$$

if $T$ is discrete and takes values $t_0$ and $t_1$, or

$$\Delta (x) = \partial g (t, w) / \partial t$$

if $T$ is continuous and $t \mapsto g (t, w)$ is differentiable. As $\Delta (x)$ is a function of $x$, the PE of an observational unit, $\Delta (X)$, is a random variable with a distribution induced by the distribution of $X$. A popular statistic to summarize the heterogeneity of $\Delta (X)$ is the average partial effect (APE):

$$E [\Delta (X)] = \int \Delta (x) \, d\mu (x) ,$$

where $\mu$ denotes the distribution of $X$ in the population of interest. However, the APE might provide an incomplete summary of $\Delta (X)$ as it neglects all the heterogeneity by design.

Chernozhukov et al. (2018) proposed the sorted partial effect (SPE) method to provide a more complete summary of $\Delta (X)$. This method consists on reporting the entire set of PEs, sorted in increasing order and indexed by a user-specified ranking in the population of interest. More specifically, the SPEs are defined as percentiles of the PE in the population of interest, that is

$$\Delta^*_p (u) = u^{th} - \text{quantile of } \Delta (X) , \quad X \sim \mu .$$

The SPEs are also useful to conduct classification analysis (CA). This analysis consists of two steps. First, classify the observational units with PEs above or below some thresholds defined by tail SPEs in...
the most or least affected groups. Second, report and compare the moments or distributions of the outcome and covariates in the two groups.

To apply the methods in practice, we need to replace $\Delta$ and $\mu$ by sample analogs $\hat{\Delta}$ and $\hat{\mu}$, and quantify the sampling uncertainty. Chernozhukov et al. (2018) derived the theoretical underpinnings of the resulting empirical SPEs. We refer the interested reader to that paper for details. We expect the method to be helpful to a large audience. For example, medical researchers are often interested in estimating the treatment effect of a drug. Treated units can experience different effects due to individual characteristics such as age and health status. The SortedEffects package thus provides a way to visualize the heterogeneity in the effect. Researchers can also define the most and least affected groups and compare their characteristics.

The rest of the vignette proceeds as follows. We first introduce the main functions within the SortedEffects package and use an application to racial-based discrimination in mortgage lending to illustrate the command options. Then we provide another application to gender wage gap using the Current Population Survey (CPS) data. The empirical results show that the SPE method is effective in uncovering heterogeneous effects and demographic differences between the most and least affected groups in both applications.

The SortedEffects package

Functions in the package

The package contains three main commands: `spe`, `ca`, and `subpop`. The package adds three methods to the generic `plot()` (plot `spe`, plot `ca` and plot `subpop`) and `summary()` (summary `spe`, summary `ca` and summary `subpop`). In practice, users only need to type `plot` and `summary` since the generic commands will automatically dispatch the appropriate method. In this section we explain the options in each function respectively. Lastly we briefly explain the bootstrap procedures for inference and bias correction. We provide the mathematical expressions and an application to racial discrimination in mortgage lending to facilitate the understanding of the options of the commands.

`spe`

The command `spe` provides estimation and inference methods for the SPE and APE. The general syntax is:

```r
spe(fm, data, method = c("ols", "logit", "probit", "QR"),
    var_type = c("binary", "continuous", "categorical"),
    var, compare, subgroup = NULL,
    samp_weight = NULL, us = c(1:9)/10, alpha = 0.1, taus = c(5:95)/100,
    b = 500, parallel = FALSE, ncores = detectCores(), seed = 1, bc = TRUE, 
    boot_type = c("nonpar", "weighted"))
```

The option `fm` stores the user-specified regression formula to estimate the PEs, which assigns the outcome $Y$ on the left-hand-side of a $\sim$ operator, and the covariates $X$ on the right-hand-side, separated by $+$ operators. The option `data` specifies the data set that contains the variables for the analysis. The user needs to specify the variable of interest `var` as a string. The package accommodates four regression methods to estimate the PE, $\Delta(x)$: OLS (default), probit ("probit"), logit ("logit") and quantile regression ("QR").\(^1\) If "QR" is called, then the user needs to further specify the quantile indexes with `taus`. The default is `taus = c(5:95)/100`.

The option `samp_weight` allows the user to input sampling weights. When `samp_weight = NULL`, the default, the package automatically uses a vector of ones, i.e. no sampling weights are used.

The package provides three options for the variable of interest `var`:

1. `var_type = "binary"` if `var` is a binary variable such as a black or female indicator.
2. `var_type = "categorical"` if `var` is a categorical (factor) variable with more than 2 levels. One example is means of transportation with labels "bus", "train" or "bike". In this case, the user needs to further specify the two labels to be compared with the option `compare`. If the user is interested in the effect of changing from bus to bike, then `compare = c("bus","bike")`. If the data only have levels, say "1", "2" and "3", the users need to input levels instead.
3. `var_type = "continuous"` if `var` is a continuous variable. The package obtains the PEs using a

\(^1\)We use the `quantreg` package to conduct quantile regression (Koenker et al., 2018).
central difference numerical derivative of the form:

\[ f'(x) = \lim_{h \to 0} \frac{f(x + h) - f(x - h)}{2h}, \]

where \( h \) is set to be \( 1e-7 \). We avoid the symbolic derivative because it cannot correctly interpret terms involving \( I() \) for transformations of the variables such as powers, and can therefore cause erroneous estimation. The code of this part is inspired by Thomas Leeper’s vignette on the margins package (Leeper et al., 2018).

The option subgroup allows the user to specify the population of interest. The default is NULL, which corresponds to the entire population. If the user is interested in subpopulations, say households whose income is lower than 10,000 dollars, and the data in use is called Data, then the user can set subgroup = Data[,“income”] < 10000. When \( T \) is a binary treatment indicator, the user can specify that the population of interest is the treated population with subgroup = Data[,var] == 1. Note that users cannot input subgroup directly to the data option or using the subset option because the SPE methods use the whole sample to estimate the PEs. The option subgroup only specifies the population for the estimation of \( \mu \), the distribution of \( X \).

The option us specifies the set of quantile indexes corresponding to the estimated SPE to be reported. The mathematical definition of each empirical SPE is

\[ \hat{\Delta}^u_\mu (u) = u^h - \text{quantile of } \hat{\Delta} (X), \quad X \sim \hat{\mu}. \]

The empirical SPE function

\[ \left\{ u \mapsto \hat{\Delta}^u_\mu (u) : u \in U \right\}, \quad U \subset [0,1] \]

then outputs the SPEs with quantile indexes in the set \( U \). The option us specifies \( U \). For example, \( U = c(0.25,0.5,0.75) \) specifies the report the SPEs corresponding to the three quartiles.

The option alpha specifies the significance level of the confidence bands. The default is alpha = 0.1, i.e. 90% confidence level. The option b specifies the number of bootstrap repetitions. The default is \( b = 500 \).

The package supports two types of bootstrap: nonparametric (boot_type = "nonpar") and weighted with standard exponential weights (boot_type = "weighted"). The user can fix the random seed for bootstrap simulation with the option seed and decide whether or not to bias-correct the estimates with the option bc. The default is bc = TRUE. Bootstrap and bias-correction will be discussed in section 2.5. The package features parallel computing, which is convenient to speed up the bootstrap. The user can use the option parallel to turn on or off parallel computing. If parallel = TRUE, the option ncores allows the user to specify the number of cores in the parallel computing. The default is ncores = detectCores(), where detectCores() is a command of the package parallel.

The output of spe is a list containing four components: spe, ape, us and alpha. As the names indicate, spe stores the results for the SPE, and ape stores the results for the APE. Each component is a list with four elements: the estimates, lower and upper bounds of the uniform confidence bands, and bootstrapped standard errors obtained as rescaled interquartile ranges; see Chernozhukov et al. (2018). The other two components respectively store the percentile indices of the SPE and significance level of the confidence bands, which are used in the functions plot.spe and summary.spe.

The plot.spe function plots the result of spe in one graph that includes the SPEs, APE and their corresponding confidence bands. Its general syntax is

\[
\text{plot.spe(object, ylim = NULL, main = NULL, sub = NULL, xlab = "Percentile Index", ylab = "Sorted Effects", ...)}
\]

where object is the output of spe. The range of the x-axis is fixed to be the range of user-specified quantile index us. The options ylim, xlab and ylab respectively denote range of the y-axis and labels of the two axes. The options main and sub allow users to specify the main and sub titles of the plot. The option ... is an argument of the generic plot command that allows for further graphic parameters.

The syntax of summary.spe is as follows

\[
\text{summary.spe(object, result = c("sorted", "average"), ...)}
\]

If result = "sorted", the method provides a table that contain the SPE percentile indexes, estimates, bootstrap standard errors, pointwise and uniform confidence bands. If result = "average", the methods tabulates APE estimate, bootstrap standard error and confidence interval.

We illustrate the usage of the command with an empirical application to racial discrimination in mortgage lending. We use data on mortgage applications in Boston from 1990 (Munnell et al., 1996). The Federal Reserve Bank of Boston collected these data in relation to the Home Mortgage Disclosure
Act (HMDA), which was passed to monitor minority access to the mortgage market. To retrieve the data from the package, issue the command:

```r
data("mortgage")
```

The outcome variable, $Y$, is `deny`, a binary indicator for mortgage denial. The key variable of interest, $T$, is `black`, a binary indicator for the applicant being black, while the control variables, $W$, are financial and demographical characteristics that might affect the mortgage decision of the bank. These characteristics include the debt-to-income ratio (`p_irat`), expenses-to-income ratio (`hse_inc`), bad consumer credit (`ccred`), bad mortgage credit (`mcred`), credit problems (`pubrec`), denied mortgage insurance (`denpmi`), medium loan-to-house value (`ltv_med`), high loan-to-house value (`ltv_high`), self employed (`selfemp`), single (`single`), and high school graduate (`hischl`). The regression formula for the estimation of the PEs is specified as:

```r
fm <- deny ~ black + p_irat + hse_inc + ccred + mcred + pubrec + ltv_med + ltv_high + denpmi + selfemp + single + hischl
```

We invoke the `spe` command to calculate the bias-corrected estimates of the SPE at the quantile indexes $\{0.02, 0.03, \ldots, 0.98\}$ for the entire population using a logit model.

```r
test <- spe(fm = fm, data = mortgage, var = "black", method = "logit", us = c(2:98)/100, b = 500, bc = TRUE)
```

The output `test` includes the estimates and confidence bands for the APE and SPE in the entire population. We use `plot` to visualize the results.

```r
plot(x = test, ylim = c(0, 0.25), ylab = "Change in Probability", main = "APE and SPE of Being Black on the Prob of Mortgage Denial", sub = "Logit Model")
```

![APE and SPE of Being Black on the Prob of Mortgage Denial](image)

**Figure 1:** Output of `plot.spe` function. Estimates and 90% confidence bands for the APE and SPE of black indicator on probability of mortgage denial. Confidence bands obtained by bootstrap with 500 replications.

The result in Figure (1) shows significant heterogeneity in the SPEs, with the PEs ranging from 0 to 14%. The APE misses this heterogeneity, and therefore provides an incomplete picture of the effects.

We can also tabulate the result using `summary`. First, we apply the command to the APE and display the results in Table (1).

```r
summary(test, result = "average")
```

Next, we obtain the results for the SPE. To save space we only show the first 15 rows of `summary(test)` in Table (2). The columns labelled as PLB and PUB correspond to the lower and upper pointwise confidence bands, whereas the columns labelled as ULB and UUB correspond to their uniform counterparts.

```r
summary(test)
```
Table 1: Bias-corrected estimate and 90% band for APE of black indicator on probability of mortgage denial.

<table>
<thead>
<tr>
<th></th>
<th>Est</th>
<th>SE</th>
<th>90% LB</th>
<th>90% UB</th>
</tr>
</thead>
<tbody>
<tr>
<td>APE</td>
<td>0.051</td>
<td>0.019</td>
<td>0.021</td>
<td>0.081</td>
</tr>
</tbody>
</table>

Table 2: Bias-corrected estimates and 90% pointwise and uniform bands for SPE of black indicator on probability of mortgage denial.

<table>
<thead>
<tr>
<th></th>
<th>Est</th>
<th>SE</th>
<th>90% PLB</th>
<th>90% PUB</th>
<th>90% ULB</th>
<th>90% UUB</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.02</td>
<td>0.011</td>
<td>0.005</td>
<td>0.003</td>
<td>0.018</td>
<td>0.001</td>
<td>0.020</td>
</tr>
<tr>
<td>0.03</td>
<td>0.012</td>
<td>0.005</td>
<td>0.004</td>
<td>0.020</td>
<td>0.002</td>
<td>0.022</td>
</tr>
<tr>
<td>0.04</td>
<td>0.013</td>
<td>0.006</td>
<td>0.004</td>
<td>0.022</td>
<td>0.002</td>
<td>0.024</td>
</tr>
<tr>
<td>0.05</td>
<td>0.014</td>
<td>0.006</td>
<td>0.004</td>
<td>0.023</td>
<td>0.002</td>
<td>0.025</td>
</tr>
<tr>
<td>0.06</td>
<td>0.014</td>
<td>0.006</td>
<td>0.004</td>
<td>0.024</td>
<td>0.002</td>
<td>0.026</td>
</tr>
<tr>
<td>0.07</td>
<td>0.015</td>
<td>0.007</td>
<td>0.004</td>
<td>0.026</td>
<td>0.002</td>
<td>0.028</td>
</tr>
<tr>
<td>0.08</td>
<td>0.016</td>
<td>0.007</td>
<td>0.005</td>
<td>0.027</td>
<td>0.003</td>
<td>0.029</td>
</tr>
<tr>
<td>0.09</td>
<td>0.017</td>
<td>0.007</td>
<td>0.005</td>
<td>0.029</td>
<td>0.003</td>
<td>0.031</td>
</tr>
<tr>
<td>0.1</td>
<td>0.018</td>
<td>0.007</td>
<td>0.006</td>
<td>0.030</td>
<td>0.003</td>
<td>0.032</td>
</tr>
<tr>
<td>0.11</td>
<td>0.018</td>
<td>0.008</td>
<td>0.005</td>
<td>0.031</td>
<td>0.003</td>
<td>0.033</td>
</tr>
<tr>
<td>0.12</td>
<td>0.019</td>
<td>0.008</td>
<td>0.006</td>
<td>0.031</td>
<td>0.004</td>
<td>0.034</td>
</tr>
<tr>
<td>0.13</td>
<td>0.019</td>
<td>0.008</td>
<td>0.006</td>
<td>0.032</td>
<td>0.004</td>
<td>0.035</td>
</tr>
<tr>
<td>0.14</td>
<td>0.020</td>
<td>0.008</td>
<td>0.007</td>
<td>0.034</td>
<td>0.004</td>
<td>0.036</td>
</tr>
<tr>
<td>0.15</td>
<td>0.021</td>
<td>0.008</td>
<td>0.007</td>
<td>0.034</td>
<td>0.004</td>
<td>0.037</td>
</tr>
<tr>
<td>0.16</td>
<td>0.021</td>
<td>0.009</td>
<td>0.007</td>
<td>0.035</td>
<td>0.004</td>
<td>0.038</td>
</tr>
</tbody>
</table>

ca

The command ca provides estimation and inference methods for the CA. The general syntax is:

```r
c(a, fm, data, method = c("ols", "logit", "probit", "QR"),
 var_type = c("binary", "continuous", "categorical"),
 compare, subgroup = NULL, samp_weight = NULL, taus = c(5:95)/100,
 u = 0.1, interest = c("moment", "dist"),
 t = c(1, 1, rep(0, dim(data)[2] - 2)), cl = c("both", "diff"),
 cat = NULL, alpha = 0.1, b = 500, parallel = FALSE,
 ncores = detectCores(), seed = 1, bc = TRUE,
 range_cb = c(1:99)/100, boot_type = c("nonpar", "weighted"))
```

The first step in the CA is to classify the observational units in most and least affected groups based on some tail SPEs. The option `u` specifies the quantile index of the tail SPEs. Thus, the `u`-least affected group includes the observational units with $\hat{\Delta}(X) < \hat{\Delta}^\ast \hat{\mu}(u)$ and the `u`-most affected group the units with $\hat{\Delta}(X) > \hat{\Delta}^\ast \hat{\mu}(1 - u)$. The default is $u = 0.1$ to obtain the 10% least and most affected groups. The option `subgroup` specifies the population of interest and has the same syntax as in the `spe` command.

Let $\hat{\Lambda}^{-u}_{\hat{\Delta}, \hat{\mu}}(t)$ and $\hat{\Lambda}^{+u}_{\hat{\Delta}, \hat{\mu}}(t)$ denote the objects of interest in the least and most affected groups for the CA. Define

$$\hat{\Lambda}^{\pm u}_{\hat{\Delta}, \hat{\mu}}(t) := \left[ \hat{\Lambda}^{-u}_{\hat{\Delta}, \hat{\mu}}(t), \hat{\Lambda}^{+u}_{\hat{\Delta}, \hat{\mu}}(t) \right].$$

These objects are indexed by the vector $t$, which specifies the variables of interest among the outcome and covariates. The option `t` is a vector that specifies $t$. Suppose the data has 5 variables ("a", "b", "c", "d", "e") and we are interested in "a" and "c", then we can either set $t = c("a", "c")$ directly or $t = c(1, 0, 1, 0, 0)$. The second approach requires the user to know the order of the variables in the data set, which can be found with the command `View`.

Let $Z$ denote the set of variables of interest. The package provides two types of objects of interest. If `interest = "moment"`, then $\hat{\Lambda}^{\pm u}_{\hat{\Delta}, \hat{\mu}}(t)$ include the means of the variables in $Z$ for the least and most affected groups. If `interest = "dist"`, then $\hat{\Lambda}^{\pm u}_{\hat{\Delta}, \hat{\mu}}(t)$ includes the distributions of the variables in $Z$ for the least and most affected groups.

If `interest = "moment"`, ca estimates and makes inference on features of the chosen variables of interest in the least and most affected groups. These features are specified with the option `cl`. For example, if `cl = "both"`, the command estimates the mean of the variables in the two affected groups.
On the other hand, cl = "diff" estimates the differences of the means of the variables between the two groups.

If interest = "dist", the option range_cbb specifies the region of interest for the domain of the distribution\textsuperscript{2}. For example, if the variable of interest x is discrete, we can specify range_cbb = NULL. If x is continuous, we can specify range_cbb = c(1:99)/100 if we are interested in the percentiles with indexes \{0.01, 0.03, \ldots, 0.99\}. The default is range_cbb = c(1:99)/100. The choice range_cbb = NULL shuts down this feature by setting the region of interest as all the distinct values of the variable.

The output of ca depends on the choice of interest. For interest = "moment", the output is a list containing the estimates of \( \hat{\Lambda}_u \), \( \hat{\Delta} \), \( \hat{\mu}(t) \), bootstrapped standard errors, pointwise and adjusted p-values. The null hypothesis for the p-values is that the estimated coefficient is zero. The p-values are adjusted for multiplicity to account for joint testing for all variables. In addition, users can adjust the pointwise p-values to account for joint testing for all simultaneous tests of categories within a factor. For example, if the variables of interest include a marital status factor "ms" with labels ("nevermarried", "married", "divorced", "separated", "widowed"), then users could consider adjusting the pointwise p-values within this factor. To illustrate how to define the option cat, suppose we have selected specified 3 variables of interest: \( t = c(\"a\", \"b\", \"c\") \). Without loss of generality, assume "a" is not a factor, while "b" and "c" are two factors. Then we need to specify cat as cat = c("b", "c"). If cat = NULL, we report the unadjusted pointwise p-values. If interest = "dist", the output is a list containing the rearranged estimates, upper confidence bands and lower confidence bands for the variables of interest in both groups.

When interest = "moment", the user can use method summary.ca to tabulate the output. The general syntax is

```r
summary.ca(object, ...)
```

If cl = "both", the p-values are omitted from the table. When interest = "dist", users can plot the output for better visualization. The general syntax is

```r
plot.ca(object, var, main = NULL, sub = NULL, xlab = NULL, ylab = NULL, ...)
```

The user needs to input the variable for plotting with the option var. Note that the variable must be one of the variables specified in t.

Returning to the mortgage denial example, we classify the 10% least and most affected applicants and compare their characteristics. The variables of interest include deny, black and all the controls. We first specify t to reflect the choice of variables of interest\textsuperscript{3}

```r
t <- c("deny", "p_irat", "black", "hse_inc", "ccred", "mcred", "pubrec", "denpmi", "selfemp", "single", "hischl", "ltv_med", "ltv_high")
```

Then we invoke the ca command and summarize the result.

```r
CA <- ca(fm = fm, data = mortgage, var = "black", method = "logit", cl = "both", 
t = t, b = 500, bc = TRUE)
```

```r
summary(CA)
```

Table (3) shows that the 10% of the applicants most affected by the racial mortgage denial gap are more likely to have either of the following characteristics relative to the 10% of the least affected applicants: mortgage denied, high debt-to-income ratio, black, high expense-to-income ratio, bad consumer or credit scores, credit problems, self employed, single, no high school diploma, and medium or high loan-to-income ratio.

Next we test if the differences in the characteristics between the two groups are statistically significant. To do so we set cl = "diff", which means taking difference between the two groups. The full command is as follows

```r
CAdiff <- ca(fm = fm, data = mortgage, var = "black", t = t, method = "logit", cl = "diff", b = 500, bc = TRUE)
```

```r
summary(CAdiff)
```

Table (4) shows the results. The joint p-values account for the fact that we conduct simultaneous inference on 13 differences of variables. We employ the so-called “single-step” methods for controlling the family-wise error rate and obtain the p-values by bootstrap. We find that 8 differences are jointly statistically different from zero at the 5% level and 9 at the 10% level.

\textsuperscript{2}Note that cl doesn’t have any bearing when interest = "dist".

\textsuperscript{3}Alternatively, we can use View(mortgage) to locate the variables and set t <- c(rep(1, 4), 0, rep(1, 7), 0, 0, 1, 1).
Table 3: Bias-corrected estimates and standard errors for the average characteristics of the groups with the 10% most and least effected applicants by the racial mortgage denial gap.

<table>
<thead>
<tr>
<th></th>
<th>Most SE</th>
<th>Least SE</th>
</tr>
</thead>
<tbody>
<tr>
<td>deny</td>
<td>0.45 0.03</td>
<td>0.09 0.04</td>
</tr>
<tr>
<td>p_irat</td>
<td>0.39 0.01</td>
<td>0.25 0.02</td>
</tr>
<tr>
<td>black</td>
<td>0.38 0.03</td>
<td>0.06 0.02</td>
</tr>
<tr>
<td>hse_inc</td>
<td>0.28 0.01</td>
<td>0.21 0.02</td>
</tr>
<tr>
<td>ccred</td>
<td>4.80 0.26</td>
<td>1.28 0.09</td>
</tr>
<tr>
<td>mcred</td>
<td>2.01 0.06</td>
<td>1.56 0.10</td>
</tr>
<tr>
<td>pubrec</td>
<td>0.46 0.05</td>
<td>0.05 0.02</td>
</tr>
<tr>
<td>denpmi</td>
<td>0.01 0.01</td>
<td>0.04 0.03</td>
</tr>
<tr>
<td>selfemp</td>
<td>0.17 0.04</td>
<td>0.04 0.03</td>
</tr>
<tr>
<td>single</td>
<td>0.61 0.06</td>
<td>0.09 0.07</td>
</tr>
<tr>
<td>hischl</td>
<td>0.93 0.03</td>
<td>1.00 0.01</td>
</tr>
<tr>
<td>ltv_med</td>
<td>0.59 0.06</td>
<td>0.05 0.04</td>
</tr>
<tr>
<td>ltv_high</td>
<td>0.12 0.04</td>
<td>0.01 0.01</td>
</tr>
</tbody>
</table>

Table 4: Bias-corrected estimates, standard errors, and p-values for the differences in average characteristics between the groups with the 10% most and least affected applicants by the racial mortgage denial gap. P-values are for the hypothesis that the difference is zero. P-vals are pointwise p-values for each difference and JP-vals are joint p-values that account for simultaneous inference on all the differences.

<table>
<thead>
<tr>
<th></th>
<th>Estimate</th>
<th>SE</th>
<th>JP-vals</th>
<th>P-vals</th>
</tr>
</thead>
<tbody>
<tr>
<td>deny</td>
<td>0.36</td>
<td>0.05</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>p_irat</td>
<td>0.14</td>
<td>0.02</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>black</td>
<td>0.32</td>
<td>0.04</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>hse_inc</td>
<td>0.07</td>
<td>0.02</td>
<td>0.13</td>
<td>0.00</td>
</tr>
<tr>
<td>ccred</td>
<td>3.52</td>
<td>0.28</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>mcred</td>
<td>0.65</td>
<td>0.15</td>
<td>0.01</td>
<td>0.00</td>
</tr>
<tr>
<td>pubrec</td>
<td>0.41</td>
<td>0.05</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>denpmi</td>
<td>-0.03</td>
<td>0.04</td>
<td>1.00</td>
<td>0.23</td>
</tr>
<tr>
<td>selfemp</td>
<td>0.13</td>
<td>0.06</td>
<td>0.48</td>
<td>0.02</td>
</tr>
<tr>
<td>single</td>
<td>0.53</td>
<td>0.10</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>hischl</td>
<td>-0.06</td>
<td>0.03</td>
<td>0.48</td>
<td>0.02</td>
</tr>
<tr>
<td>ltv_med</td>
<td>0.54</td>
<td>0.07</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>ltv_high</td>
<td>0.10</td>
<td>0.03</td>
<td>0.10</td>
<td>0.00</td>
</tr>
</tbody>
</table>

We also plot the distributions of monthly debt-to-income ratio (p_irat) and monthly housing expenses-to-income ratio (hse_inc) for both groups. Such plots are useful if the user wants to visualize if there is stochastic dominance between the two groups. To do so we use the ca command and change the interest to "dist".

t2 <- c("p_irat", "hse_inc")
CADist <- ca(fm = fm, data = mortgage, var = "black", method = "logit", t = t2,
b = 500, interest = "dist")
plot(CADist, var = "p_irat", ylab = "Prob", xlab = "Monthly Debt-to-Income Ratio", sub = "logit model")
plot(CADist, var = "hse_inc", ylab = "Prob", xlab = "Monthly Housing Expenses-to-Income Ratio", sub = "logit model")

Figure (2) shows that for both variables the distribution in the most affected group first-order stochastically dominates the distribution in least affected group.

subpop

In addition to means and distributions, we can conduct inference on the sets of most and least affected units. Let $Z$ be a compact subset of the support of the outcome and covariates. Define

$$M^{-u} \equiv \{(x, y) \in Z : \Delta(x) \leq \Delta^*_u(u)\}$$
as the set of the least affected units and

\[ \mathcal{M}^{-u} = \{ (x, y) \in \mathcal{Z} \colon \Delta(x) \geq \Delta^u_p (1 - u) \} \]

as the set of the most affected units. The command subpop provides estimation and inference methods for these sets. The general syntax is

```r
subpop(fm, data, method = c("ols", "logit", "probit", "QR"),
    var_type = c("binary", "continuous", "categorical"), var, compare,
    subgroup = NULL, samp_weight = NULL, taus = c(5:95)/100, u = 0.1, alpha = 0.1,
    b = 500, seed = 1, parallel = FALSE, ncores = detectCores(),
    boot_type = c("nonpar", "weighted"))
```

No new option is introduced in the command. For theoretical details, we refer the reader to Cher- nozhukov et al. (2015).

The output of subpop is a list containing six components: cs_most, cs_least, u, subgroup, most and least. As the names indicate, cs_most and cs_least denote the confidence sets for the most and least affected groups. u stores the percentile index that defines the most and least affected groups. subgroup stores the indicators for the population of interest specified with the option subgroup. most and least store the estimates of the most and least affected units respectively. The first four components are used in plot.subpop and the last two components can be visualized with summary.subpop.

The general syntax of summary.subpop is

```r
summary.subpop(object, affected = c("most", "least"), vars = NULL, ...)
```

The option object is the output of subpop. The option affected allows users to tabulate either the most or the least affected units, and the option vars provides summary statistics for user-specified variables of interest. The summary statistics include the minimum, 1st quartile, median, mean, 3rd quartile and maximum. The default is NULL, which produces summary statistics of all the variables.

The plot.subpop function plots 2-dimensional projections of the confidence sets for the most and least affected units with respect to two variables. The general syntax is

```r
plot.subpop(object, varx, vary, xlim = NULL, ylim = NULL, main = NULL, sub = NULL,
    xlab = NULL, ylab = NULL, overlap = FALSE, ...)
```

The user needs to specify the two variables for the projection with varx and vary, and object should be specified as the output of subpop. The option overlap allows users to either keep or drop common observations in both confidence sets. The default is overlap = FALSE, which drops the observations.

We estimate the 10% most and least affected applicants in the mortgage application.

```r
set_b <- subpop(fm, data = mortgage, method = "logit", var = "black", u = 0.1,
    alpha = 0.1, b = 500)
```

Using summary, we can estimate the most/least affected applicants and report summary statistics of the variables of interest in the most/least affected groups. Table 5 lists the estimated most affected applicants. For the purpose of illustration we only show the first ten rows and columns.

```r
groups <- summary(set_b, vars = c("p_irat", "hse_inc"))
most_affected <- groups$most_affected
```
Table 5: Elements of the estimated group of the 10% most affected applicants by racial mortgage denial gap.

<table>
<thead>
<tr>
<th>deny</th>
<th>p_irat</th>
<th>black</th>
<th>hse_inc</th>
<th>loan_val</th>
<th>ccred</th>
<th>mcred</th>
<th>pubrec</th>
<th>denpmi</th>
<th>selfemp</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.00</td>
<td>0.46</td>
<td>0.00</td>
<td>0.27</td>
<td>0.84</td>
<td>5.00</td>
<td>2.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>2</td>
<td>1.00</td>
<td>0.38</td>
<td>0.00</td>
<td>0.26</td>
<td>0.88</td>
<td>6.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.00</td>
</tr>
<tr>
<td>3</td>
<td>0.00</td>
<td>0.40</td>
<td>0.00</td>
<td>0.34</td>
<td>0.80</td>
<td>2.00</td>
<td>2.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>4</td>
<td>1.00</td>
<td>0.24</td>
<td>0.00</td>
<td>0.23</td>
<td>0.90</td>
<td>5.00</td>
<td>2.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>5</td>
<td>0.00</td>
<td>0.38</td>
<td>0.00</td>
<td>0.25</td>
<td>0.80</td>
<td>6.00</td>
<td>2.00</td>
<td>1.00</td>
<td>0.00</td>
</tr>
<tr>
<td>6</td>
<td>0.00</td>
<td>0.36</td>
<td>0.00</td>
<td>0.13</td>
<td>0.95</td>
<td>5.00</td>
<td>2.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>7</td>
<td>0.00</td>
<td>0.35</td>
<td>0.00</td>
<td>0.27</td>
<td>0.90</td>
<td>6.00</td>
<td>2.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>8</td>
<td>1.00</td>
<td>0.30</td>
<td>0.00</td>
<td>0.30</td>
<td>0.50</td>
<td>6.00</td>
<td>2.00</td>
<td>1.00</td>
<td>0.00</td>
</tr>
<tr>
<td>9</td>
<td>1.00</td>
<td>0.37</td>
<td>0.00</td>
<td>0.23</td>
<td>0.80</td>
<td>6.00</td>
<td>2.00</td>
<td>1.00</td>
<td>0.00</td>
</tr>
<tr>
<td>10</td>
<td>1.00</td>
<td>0.39</td>
<td>0.00</td>
<td>0.27</td>
<td>0.90</td>
<td>6.00</td>
<td>2.00</td>
<td>1.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

We can also report summary statistics of the variables of interest in the most and least affected groups using the output of summary. Table (6) reports summary statistics for p_irat and hse_inc for the applicants in the most affected group.

sum_stats_most <- groups$stats_most

Table 6: Summary statistics of the estimated group of the 10% most affected applicants by racial mortgage denial gap.

<table>
<thead>
<tr>
<th></th>
<th>p_irat</th>
<th>hse_inc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min</td>
<td>0.16</td>
<td>0.01</td>
</tr>
<tr>
<td>1st Quartile</td>
<td>0.34</td>
<td>0.23</td>
</tr>
<tr>
<td>Median</td>
<td>0.37</td>
<td>0.28</td>
</tr>
<tr>
<td>Mean</td>
<td>0.39</td>
<td>0.28</td>
</tr>
<tr>
<td>3rd Quartile</td>
<td>0.42</td>
<td>0.32</td>
</tr>
<tr>
<td>Max</td>
<td>1.16</td>
<td>0.74</td>
</tr>
</tbody>
</table>

We finally plot the projection of the confidence sets for the most and least affected applicants with respect to p_irat and hse_inc. Figure (3) keeps the overlapped observations and shows that the most affected applicants tend to have higher levels of debt to income and expenses to income ratios.

plot(set_b, varx = mortgage$p_irat, vary = mortgage$hse_inc, xlim = c(0, 1.5), ylim = c(0, 1.5), xlab = "Debt/Income", ylab = "Housing expenses/Income", overlap = TRUE)

Inference

Chernozhukov et al. (2018) derived the asymptotic distributions and bootstrap validity for the estimators of the SPE and classification analysis. The package uses bootstrap to compute standard errors and critical values for tests and confidence bands.

The package features nonparametric and weighted bootstrap. When boot_type = "nonpar", the package draws samples with replacement of the variables and samp_weight and run all estimation commands weighted by samp_weight. When boot_type = "weighted", the package draws weights from the standard exponential distribution and runs all estimation commands weighted by the product of these weights and samp_weight. We use the boot package (Canty and Ripley, 2017), which is flexible enough to accommodate both types.

Inference on SPE

The $(1 - \alpha)$-uniform confidence band of $\Delta_\mu^* (u)$ in $\mathcal{U}$ is

$$\hat{\Delta}_\mu^* - \hat{f}_{1-\alpha} (\mathcal{U}) \hat{\Sigma} (u)^{1/2} / \sqrt{n}, \quad \hat{\Delta}_\mu^* + \hat{f}_{1-\alpha} (\mathcal{U}) \hat{\Sigma} (u)^{1/2} / \sqrt{n},$$

where $\hat{f}_{1-\alpha} (\mathcal{U})$ is a bootstrapped uniform critical value and $\hat{\Sigma} (u)^{1/2}$ is a bootstrapped standard error of $\hat{\Delta}_\mu^* (u)$. To deal with the possibility that the end-point functions of the confidence band $u \mapsto \hat{\Delta}_\mu^* (u) \hat{\Sigma} (u)^{1/2} / \sqrt{n}$ be nonincreasing, we monotonize these functions via rearrangement.

\(^4\)See Algorithm 2.1 in Chernozhukov et al. (2018) for details.
(Chernozhukov et al., 2009). The $(1 - \alpha)$-pointwise confidence band of $\Delta_{\alpha, \beta}(u)$ is obtained replacing $\hat{t}_{1 - \alpha}(T)$ by the $(1 - \alpha/2)$-quantile of the standard normal distribution.

**Inference on CA**  The joint $p$-value for the hypothesis $\Lambda_{\alpha, \beta}^+ (t) = \Lambda_{\alpha, \beta}^- (t)$ for all $t \in T$ is

$$
Pr \left( \bar{t}^u (T) > \sup_{t \in T} \frac{|c' \hat{\Lambda}_{\alpha, \beta}^u (t) c - c' \Lambda_{\alpha, \beta}^u (t) c|}{\sqrt{c' \hat{\Sigma}^u (t) c}} \right),
$$

where $c = (-1, 1)^T$, $\bar{t}^u (T)$ is a bootstrap estimator of

$$
\bar{t}^u (T) = \sup_{t \in T} \frac{|c' \hat{\Lambda}_{\alpha, \beta}^u (t) c - c' \Lambda_{\alpha, \beta}^u (t) c|}{\sqrt{c' \hat{\Sigma}^u (t) c}},
$$

and $\hat{\Sigma}^u (t)$ is a bootstrap estimator of $\Sigma^u (t)$, the asymptotic variance of $\hat{\Lambda}_{\alpha, \beta}^u (t)$.$^5$ The pointwise $p$-value for the hypothesis $\Lambda_{\alpha, \beta}^+ (t) = \Lambda_{\alpha, \beta}^- (t)$ is obtained by setting $T = \{t\}$.

**Inference on sets of least/most affected units**  The outer $(1 - \alpha)$-confidence set for $\mathcal{M}^+ u$ is

$$
\mathcal{C}_u \mathcal{M}^+ u (1 - \alpha) = \left\{ (x, y) \in \mathcal{Z} : \bar{\Sigma}^{-1/2} (x, y) \sqrt{n} \left( \hat{\Lambda} (x) - \hat{\Lambda}_{\alpha}^+ (u) \right) \leq \hat{c} (1 - \alpha) \right\},
$$

where $\bar{\Sigma} (x, u)$ is an estimator of the asymptotic variance of $\hat{\Lambda} (x) - \hat{\Lambda}_{\alpha}^+ (u)$. Similarly the outer $(1 - \alpha)$-confidence set for $\mathcal{M}^- u$ is

$$
\mathcal{C}_u \mathcal{M}^- u (1 - \alpha) = \left\{ (x, y) \in \mathcal{Z} : \bar{\Sigma}^{-1/2} (x, 1 - u) \sqrt{n} \left( \hat{\Lambda}_{\alpha}^- (1 - u) - \hat{\Lambda} (x) \right) \leq \hat{c} (1 - \alpha) \right\}.
$$

The critical value $\hat{c} (1 - \alpha)$ is the $(1 - \alpha)$-quantile of the statistic:

$$
\hat{\nu}_{\alpha}^* \sup_{x \in \mathcal{X} : \hat{\Lambda} (x) = \hat{\Lambda}_{\alpha}^+ (u)} \bar{\Sigma}^{-1/2} (x, y) \sqrt{n} \left( \hat{\Lambda} (x) - \hat{\Lambda}_{\alpha}^+ (u) \right),
$$

while the critical value $\hat{c} (1 - \alpha)$ is the $(1 - \alpha)$-quantile of the statistic:

$$
\hat{\nu}_{\alpha}^* \sup_{x \in \mathcal{X} : \hat{\Lambda} (x) = \hat{\Lambda}_{\alpha}^- (1 - u)} \bar{\Sigma}^{-1/2} (x, 1 - u) \sqrt{n} \left( \hat{\Lambda} (x) - \hat{\Lambda}_{\alpha}^- (1 - u) \right).
$$

$^5$See Algorithm 2.2 in Chernozhukov et al. (2018) for details.
To implement $\sup \{ x \in \mathcal{X} : \hat{\Delta} (x) = \hat{\Delta}_p (u) \}$ in the code, we find the minimum of $|\hat{\Delta} (x) - \hat{\Delta}_p (u)|$ among all $x$’s.

**Bias-correction** Nonlinear estimators are prone to finite-sample bias, and bootstrap methods can estimate the bias up to some asymptotic order. To bias correct the SPE, replace $\hat{\Delta}_p^u$ with $2\hat{\Delta}_p^u - \hat{\Delta}_p^*$, where $\hat{\Delta}_p^*$ is the mean of bootstrap draws. Similarly, for CA we replace $\hat{\Delta}_p^u (t) - \hat{\Delta}_p^{\ast u} (t)$ with $2\hat{\Delta}_p^{u} (t) - \hat{\Delta}_p^{\ast u} (t)$, where $\hat{\Delta}_p^{\ast u}$ is the mean of the bootstrap draws. Bias-corrected estimates and corresponding inference will be reported if $bc = TRUE$, the default.

**Gender wage gap application**

We analyze the gender wage gap using data from the U.S. March Supplement of the Current Population Survey (CPS) in 2015. The gender wage gap measures the difference in wages between female and male workers with the same observable characteristics. The SPE method allows us to look for heterogeneity in the gender wage gap and to identify the characteristics of the most and least affected workers. To retrieve the data, issue the command

```r
data(wage2015)
```

The data contain the following variables: log hourly wages ($\ln w$); a marital status factor $ms$ with 5 categories widowed, divorced, separated, nevermarried, married; CPS sampling weights ($\text{weight}$); a indicator for female worker ($\text{female}$); an education attainment factor $\text{educ}$ with 5 categories 1hs (less than high school graduate), hsg (high school graduate), sc (some college), cg (college) and ad (advanced degree); a region factor $\text{region}$ with 4 categories mw (midwest), so (south), we (west) and ne (northeast); potential work experience $\text{exp1}$ computed as max $\{0, \text{age} - \text{years of educ} - 7\}$; 4 powers of experience ($\text{exp2, exp3, exp4}$); an occupation factor $\text{occ}$ with 5 categories manager, service, sales, construction and production, and an industry factor $\text{ind}$ with 12 categories minery, construction, manufacture, retail, transport, information, finance, professional, education, leisure, services and public.

The CPS data contains sampling weights in the variable weight, so we will set $\text{samp \_weight} = \text{wage2015} \_\text{weight}$. Because women in general earn less than men, the PEs are predominately negative if we use the female indicator. To facilitate the interpretation of most and least affected groups we create an indicator called male, which assigns 0 to female workers instead.

```r
wage2015$male <- 1 - wage2015$female
```

We apply OLS regression to estimate the PEs using the following specification

```r
fmla1 <- lnw ~ male*(ms + (exp1 + exp2 + exp3 + exp4)*educ + occ + ind + region)
```

We first look at the SPE of the gender wage gap at the quantile indexes $\{0.02, 0.03, \ldots, 0.98\}$ in the population of women via the command $\text{spe}$ and plot the result. We specify that the population of interest is female workers with $\text{subgroup} = \text{wage2015}\_\{\text{"female"}\} == 1$.

```r
gap <- spe(fm = fmla1, data = wage2015, samp_weight = wage2015$weight, var = "male", subgroup = wage2015[,"female"] == 1, boot_type = "weighted", us = c(2:98)/100, b = 500, bc = FALSE)
```

Figure (i) shows large heterogeneity in the gender wage gap that is missed if we only report the APE.

We also compare the SPE across subsets of women defined by marital status. We implement this by changing the $\text{subgroup}$ options as follows

```r
fem_mar <- wage2015[, "female"] == 1 & wage2015[, "ms"] == "married"
fem_nev <- wage2015[, "female"] == 1 & wage2015[, "ms"] == "nevermarried"
gap_mar <- spe(fm = fmla1, data = wage2015, samp_weight = wage2015$weight, var = "male", subgroup = fem_mar, us = c(2:98)/100, b = 500, bc = FALSE, boot_type = "weighted")
gap_nev <- spe(fm = fmla1, data = wage2015, samp_weight = wage2015$weight, var = "male", subgroup = fem_nev, us = c(2:98)/100, b = 500, bc = FALSE, boot_type = "weighted")
```
Figure 4: Estimates and 90% confidence bands for the APE and SPE of the gender wage gap for women. Confidence bands obtained by weighted bootstrap with 500 replications.

```r
plot(x = gap_mar, main = "Married Women", sub = "OLS Model", xlab = "Percentile Index", ylab = "Gender Wage Gap", ylim = c(-0.2, 0.45))
plot(x = gap_nev, main = "Never Married Women", sub = "OLS Model", xlab = "Percentile Index", ylab = "Gender Wage Gap", ylim = c(-0.2, 0.45))
```

Figure (5) shows the results for the two subpopulations. Here we find large heterogeneity not only between married and never married women, but also within these more narrowly defined subpopulations. Now we compare the differences in characteristics of the 5% most and least affected women using weighted bootstrap with 500 repetitions. We pick the following variables

```r
tw <- c("lnw", "female", "ms", "educ", "region", "exp1", "occ", "ind")
```

Since many variables are category indicators of the same factor, we can specify `cat` as follows to get categorical p-values.

```r
cat <- c("ms", "educ", "region", "occ", "ind")
```

Then we issue the `ca` command and tabulate the mean characteristics of the two groups

```r
Char <- ca(fm = fmla1, data = wage2015, samp_weight = wage2015$weight, var = "male", t = tw, cl = "both", b = 500, subgroup = wage2015[,"female"] == 1, boot_type = "weighted", bc = FALSE, u = 0.05)
```

Figure 5: Estimates and 90% confidence bands for the APE and SPE of the gender wage gap for married and never married women. Confidence bands obtained by weighted bootstrap with 500 replications.
Table (7) reports the estimates and standard errors obtained by weighted bootstrap with 500 replications. We find that, compared to the 5% least affected women, the 5% most affected women are much more likely to be married, much less likely to be never married, less likely to have an advanced degree, live in the South, don’t live in Northeast and West, possess more potential experience, are more likely to have sales and non-managerial occupations, and work more often in manufacture, retail, transport and finance, and less often in education and leisure industries.

We also test the statistical significance of the mean differences. Table 8 shows that the differences mentioned above are significant after controlling for simultaneous inference within categories, but only the differences in marital status, potential experience and education industry remain jointly significant at the 5% level.

```r
Chardiff <- ca(fm = fmla1, data = wage2015, samp_weight = wage2015$weight, var = "male", t = tw, cl = "diff", b = 500, cat = cat, bc = FALSE, u = 0.05
    subgroup = wage2015[, "female"] == 1, boot_type = "weighted",)
summary(Chardiff)
```

Lastly we use show the functionality of the command subpop. We plot projections of 90% confidence sets for the 5% most and least affected group with respect to two pairs of variables: log wages and potential experience, and marital status and potential experience. The estimated sets are obtained by weighted bootstrap with 500 repetitions and we drop the overlapped observations.
Table 8: Bias-corrected estimates, standard errors, and p-values for the differences in average characteristics between the groups of the 5% most and least affected women by the gender wage gap. P-values are for the hypothesis that the difference is zero. JP-vals are joint p-values that account for simultaneous inference on all the differences and Cat JP-vals are joint p-values that account for simultaneous inference within the categories ms, educ, region, occ and ind.

<table>
<thead>
<tr>
<th>Estimate</th>
<th>SE</th>
<th>JP-vals</th>
<th>Cat P-vals</th>
</tr>
</thead>
<tbody>
<tr>
<td>lnw</td>
<td>-0.06</td>
<td>0.12</td>
<td>1.00</td>
</tr>
<tr>
<td>female</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>exp1</td>
<td>18.12</td>
<td>3.49</td>
<td>0.09</td>
</tr>
<tr>
<td>occ_manager</td>
<td>-0.55</td>
<td>0.18</td>
<td>0.49</td>
</tr>
<tr>
<td>occ_service</td>
<td>-0.08</td>
<td>0.08</td>
<td>1.00</td>
</tr>
<tr>
<td>occ_sales</td>
<td>0.46</td>
<td>0.17</td>
<td>0.67</td>
</tr>
<tr>
<td>occ_construction</td>
<td>-0.01</td>
<td>0.02</td>
<td>1.00</td>
</tr>
<tr>
<td>occ_production</td>
<td>0.18</td>
<td>0.09</td>
<td>0.89</td>
</tr>
<tr>
<td>ind_minery</td>
<td>0.00</td>
<td>0.02</td>
<td>1.00</td>
</tr>
<tr>
<td>ind_construction</td>
<td>-0.01</td>
<td>0.02</td>
<td>1.00</td>
</tr>
<tr>
<td>ind_manufacture</td>
<td>0.18</td>
<td>0.10</td>
<td>0.97</td>
</tr>
<tr>
<td>ind_transport</td>
<td>0.14</td>
<td>0.07</td>
<td>0.90</td>
</tr>
<tr>
<td>ind_information</td>
<td>-0.00</td>
<td>0.04</td>
<td>1.00</td>
</tr>
<tr>
<td>ind_finance</td>
<td>0.41</td>
<td>0.18</td>
<td>0.86</td>
</tr>
<tr>
<td>ind_professional</td>
<td>-0.00</td>
<td>0.08</td>
<td>1.00</td>
</tr>
<tr>
<td>ind_education</td>
<td>-0.56</td>
<td>0.10</td>
<td>0.07</td>
</tr>
<tr>
<td>ind_leisure</td>
<td>-0.21</td>
<td>0.09</td>
<td>0.79</td>
</tr>
<tr>
<td>ind_services</td>
<td>-0.09</td>
<td>0.05</td>
<td>0.98</td>
</tr>
<tr>
<td>ind_public</td>
<td>-0.00</td>
<td>0.06</td>
<td>1.00</td>
</tr>
<tr>
<td>educ_lhs</td>
<td>0.00</td>
<td>0.05</td>
<td>1.00</td>
</tr>
<tr>
<td>educ_hsg</td>
<td>0.22</td>
<td>0.16</td>
<td>1.00</td>
</tr>
<tr>
<td>educ_sc</td>
<td>0.38</td>
<td>0.16</td>
<td>0.77</td>
</tr>
<tr>
<td>educ_cg</td>
<td>-0.21</td>
<td>0.16</td>
<td>1.00</td>
</tr>
<tr>
<td>educ_ad</td>
<td>-0.39</td>
<td>0.12</td>
<td>0.37</td>
</tr>
<tr>
<td>ms_married</td>
<td>0.98</td>
<td>0.05</td>
<td>0.00</td>
</tr>
<tr>
<td>ms_widowed</td>
<td>-0.03</td>
<td>0.07</td>
<td>1.00</td>
</tr>
<tr>
<td>ms_separated</td>
<td>-0.03</td>
<td>0.03</td>
<td>1.00</td>
</tr>
<tr>
<td>ms_divorced</td>
<td>-0.07</td>
<td>0.05</td>
<td>0.99</td>
</tr>
<tr>
<td>ms_nevermarried</td>
<td>-0.85</td>
<td>0.12</td>
<td>0.02</td>
</tr>
<tr>
<td>region_mw</td>
<td>0.01</td>
<td>0.15</td>
<td>1.00</td>
</tr>
<tr>
<td>region_so</td>
<td>0.22</td>
<td>0.17</td>
<td>1.00</td>
</tr>
<tr>
<td>region_we</td>
<td>-0.10</td>
<td>0.11</td>
<td>1.00</td>
</tr>
<tr>
<td>region_ne</td>
<td>-0.13</td>
<td>0.12</td>
<td>1.00</td>
</tr>
</tbody>
</table>

set <- subpop(fm = fmla1, data = wage2015, var = "male", samp_weight = wage2015$weight, boot_type = "weighted", b = 500, subgroup = wage2015[, "male"] == 0, u = 0.05)

plot(set, varx = wage2015$exp1, vary = wage2015$lnw, main = "Projections of Exp-lnw", sub = "OLS", xlab = "Exp", ylab = "Log Wages")

plot(set, varx = wage2015$exp1, vary = wage2015$ms, main = "Projections of Exp-MS", sub = "OLS", xlab = "Exp", ylab = "Marital Status")

Figure (6) shows that there are relatively more least affected women with low experience at all wage levels, more high affected women with high wages with between 15 and 45 years of experience, and more least affected women which are not married at all experience levels.
Figure 6: Projections of the 90% confidence sets for the groups of the 5% most affected women by the gender wage gap with respect to $\exp$ and $\ln w$ (left panel), and $\exp$ and $ms$ (right panel). Confidence regions obtained by weighted bootstrap with 500 replications.

Acknowledgements

We thank the editor Norman Matloff, Thomas Leeper and an anonymous referee for valuable suggestions on the paper and package. We gratefully acknowledge research support from the NSF.

Bibliography


Shuowen Chen
Department of Economics
Boston University
270 Bay State Road, Boston, MA 02215
USA
swchen@bu.edu

Victor Chernozhukov
Department of Economics
Massachusetts Institute of Technology
50 Memorial Drive, Cambridge, MA 02142
USA
vchern@mit.edu

Iván Fernández-Val
Department of Economics
Boston University
270 Bay State Road, Boston, MA 02215
USA
ivanf@bu.edu

Ye Luo
Faculty of Business and Economics
The University of Hong Kong
Cyberport Four, 100 Cyberport Rd, Telegraph Bay
Hong Kong
hurtluo@hku.hk
**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 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, Ferdhiana-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), Neuhäuser 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 $N^* = n_1 \times n_2 \times ... \times n_k$ 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. Ferdhiana 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: $(1, X_{11}), (2, X_{2j}), ..., (k, X_{kj})$, where $X_{ij}$ $i = 1, 2, ..., k, j = 1, 2, ..., n_i$ 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), jGWAS (Lin et al., 2017a), fastJT (Lin et al., 2017b), kSamples (Scholz and Zhu, 2018), StatCharrms (Swintek et al., 2018), PMCMRplus (Pohliert, 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 \(X_{i1}, X_{i2}, ..., X_{im}, i = 1, ..., k\) 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

\[
H_0: F_1(x) = F_2(x) = ... = F_k(x) \quad \forall x. \tag{1}
\]

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.,

\[
H_1: F_1(x) \geq F_2(x) \geq ... \geq F_k(x) \quad \exists x: F_1(x) > F_k(x). \tag{2}
\]

Under \(H_1\), \(X_i\) tends to be smaller than \(X_{i+1}\), \(i = 1, 2, ..., k - 1\), 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., 2013)

\[
H_1: \theta_1 \leq \theta_2 \leq ... \leq \theta_k \quad (\theta_1 < \theta_k). \tag{3}
\]

Similarly, the ordered alternative hypothesis

\[
H_1: F_1(x) \leq F_2(x) \leq ... \leq F_k(x) \quad \exists x: F_1(x) < F_k(x) \tag{4}
\]

states that \(X_i\) tends to be larger than \(X_{i+1}\), \(i = 1, 2, ..., k - 1\), 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

\[
H_1: \theta_1 \geq \theta_2 \geq ... \geq \theta_k \quad (\theta_1 > \theta_k). \tag{5}
\]

**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

\[
U_{ij} = \sum_{i=1}^{n_i} \sum_{m=1}^{n_j} I(X_{il} < X_{jm});
\]

where \(n_i\) and \(n_j\) are the sample sizes for the \(i\)th and \(j\)th populations, respectively, and \(I(\psi) = 1\) if \(\psi\) is true and 0 otherwise. The test statistic JT corresponds to the sum of the \(k(k - 1)/2\) Mann-Whitney statistics, i.e.,

\[
JT = \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} U_{ij}. \tag{6}
\]

The statistic JT is approximately normally distributed under \(H_0\). The mean and variance of this
where $N = n_1 + n_2 + ... + n_k$.

**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:

$$L_N = \sum_{i=1}^{k} \sum_{j=1}^{n_i} c_N(i)a_N(R_{ij})$$  \hspace{1cm} (7)

where $N$ is the combined sample size; $c_N(.)$ are the regression constants; $a_N(.)$ are the scores; $R_{ij}$ is the rank of $X_{ij}$ in the combined data. For an ordered alternative, the following proposal is made:

$$c_N(i) = i, i = 1, 2, ..., k.$$

Under $H_0$, the mean and variance of linear rank statistics are

$$E(L_N) = N\bar{c}_N\bar{a}_N,$$

and

$$V(L_N) = \frac{1}{N-1} \sum_{i=1}^{k} n_i (\bar{c}_N(i) - \bar{c}_N)^2 \sum_{i=1}^{N} (a_N(r) - \bar{a}_N)^2$$

where

$$\bar{c}_N = \frac{1}{N} \sum_{i=1}^{k} n_i c_N(i)$$

and

$$\bar{a}_N = \frac{1}{N} \sum_{i=1}^{N} a_N(r).$$

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 $a_N(.)$ according to the shape of the distribution in the literature as follows

$$a_{LS}(r) = \begin{cases} 
0 & \text{if } r \leq (N + 1)/2 \\
\frac{r - (N + 1)/2}{2} & \text{if } r > (N + 1)/2
\end{cases}$$

These scores are efficient for detecting shifts in distributions that are skewed to the left (Beier and Buning, 1997).

$$a_{ST}(r) = \begin{cases} 
\frac{r - (N + 1)/4}{4} & \text{if } r \leq (N + 1)/4 \\
\frac{(N + 1)/4 - r}{4} & \text{if } (N + 1)/4 < r < 3(N + 1)/4 \\
\frac{r - 3(N + 1)/4}{4} & \text{if } r \geq 3(N + 1)/4
\end{cases}$$

These scores are particularly good for detecting shifts in short-tailed distributions and were proposed by Gastwirth (1965).

$$a_{WS}(r) = r, \quad r = 1, 2, ..., N$$

These scores are efficient for detecting shifts in symmetric distributions with medium to heavy tails (Beier and Buning, 1997).

$$a_{LT}(r) = \begin{cases} 
-\frac{(N/4) + 1}{4} & \text{if } r < (N/4) + 1 \\
\frac{r - (N + 1)/2}{2} & \text{if } (N/4) + 1 \leq r \leq 3(N + 1)/4 \\
\frac{(N/4) + 1}{4} & \text{if } r > 3(N + 1)/4
\end{cases}$$

These scores are efficient for detecting shifts in long-tail distributions and were proposed by Buning.

\[ a_{RS}(r) = \begin{cases} r - (N + 1)/2 & \text{if } r \leq (N + 1)/2 \\ 0 & \text{if } r > (N + 1)/2 \end{cases} \]

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

\[ AT = \begin{cases} LS & \text{if } 0 \leq \hat{S}_1 \leq 0.6, \hat{S}_2 \geq 1 \\ ST & \text{if } 0.6 < \hat{S}_1 \leq 2, 1 \leq \hat{S}_2 \leq 1.5 \\ WS & \text{if } 0.6 < \hat{S}_1 \leq 2, 1.5 < \hat{S}_2 \leq 1.5 \\ LT & \text{if } 0.6 < \hat{S}_1 \leq 2, \hat{S}_2 \geq 2 \\ RS & \text{if } \hat{S}_1 \geq 2, \hat{S}_2 \geq 1 \end{cases} \]  \hspace{1cm} (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,

\[ MJT = \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} (j-i)U_{ij}, \]  \hspace{1cm} (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

\[ E(U_{ij}) = \frac{1}{2}n_in_j, \quad \forall i \neq j \]

\[ V(U_{ij}) = \frac{1}{12}n_in_j(n_i + n_j + 1), \quad \forall i \neq j \]

\[ \text{Cov}(U_{ij}, U_{il}) = \text{Cov}(U_{ij}, U_{il}) = \frac{1}{12}n_in_jn_l, \quad \text{if all } i,j,l \text{ are different} \]

\[ \text{Cov}(U_{ij}, U_{il}) = \text{Cov}(U_{ij}, U_{il}) = -\frac{1}{12}n_in_jn_l, \quad \text{if all } i,j,l \text{ are different} \]

\[ \text{Cov}(U_{ij}, U_{lm}) = 0, \quad \text{if all } i,j,l,m \text{ are different} \]

**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 \( N^* = n_1 \times n_2 \times \ldots \times n_k \) k-tuplets, where a k-tuple includes one observation from each treatment group. More specifically, the Terpstra-Magel (TM) test is based on the following statistic:

\[ TM = \sum_{i_1=1}^{n_1} \ldots \sum_{i_k=1}^{n_k} I(X_{1i_1} \leq X_{2i_2} \leq \ldots \leq X_{ki_k}) \]  \hspace{1cm} (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
where

\[ E(TM) = \frac{N^*}{k!} \]

and

\[ V(TM) = N^* \left( \frac{1}{k!} \right) \left( 1 - \frac{1}{k!} \right) + \sum_{i=1}^{k-1} v_i^2 \]

where

\[ v_i^2 = \sum_{1 \leq l_1 < \ldots < l_i \leq k} N^* \left[ \prod_{s=1}^{k} (n_s - 1)^{I(s \neq l_1) \ldots I(s \neq l_i)} \right] \left[ \frac{2(k-l_i)}{2k-1} \prod_{s=1}^{i} \left( \frac{l_s - l_{s-1} - 1}{l_s - l_{s-1} - 1} \right) - \frac{1}{(k!)^2} \right] \]

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.

\[ FTM = \sum_{i_1=1}^{n_1} \ldots \sum_{i_k=1}^{n_k} \tau(X_{i_1}, X_{2i_2}, \ldots, X_{ki_k}) \tag{11} \]

where \( \tau(X_{i_1}, X_{2i_2}, \ldots, X_{ki_k}) \) denotes the Kendall correlation coefficient based on \((1, X_{i_1}), (2, X_{2i_2}), \ldots, (k, X_{ki_k})\).

Under \( H_0 \), the statistic FTM is approximately normally distributed with zero mean, and its variance is

\[ V(FTM) = \left[ \frac{2N^*}{\sqrt{3k(k-1)}} \right]^2 \left[ \sum_{r=1}^{k-1} \sum_{s=r+1}^{k} \frac{n_r + n_s + 1}{n_r n_s} + 2 \sum_{r=1}^{k-2} \frac{1}{n_r} \left( \frac{k}{2} + \frac{r^2 - (2k-1)r}{2} \right) \right. \]

\[ \left. - 2 \sum_{r=1}^{k-2} \sum_{s=r+1}^{k-1} \frac{k-s}{n_s} + 2 \sum_{r=1}^{k-2} \sum_{s=r+1}^{k-1} \sum_{t=r+1}^{k-1} \frac{1}{n_t} \right] \]

**KTP test**

Terpstra et al. (2011) proposed the k-tuplet Terpstra-Page (KTP) test based on the statistic

\[ KTP = \sum_{i_1=1}^{n_1} \ldots \sum_{i_k=1}^{n_k} r_s(X_{i_1}, X_{2i_2}, \ldots, X_{ki_k}) \tag{12} \]

where \( r_s(X_{i_1}, X_{2i_2}, \ldots, X_{ki_k}) \) denotes the Spearman rank correlation coefficient based on \((1, X_{i_1}), (2, X_{2i_2}), \ldots, (k, X_{ki_k})\).

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

\[ S = \sum_{i_1=1}^{k-1} \sum_{i_2=i_1}^{k} \frac{(i_2 - i_1)^2(n_{i_1} + n_{i_2} + 1)}{12n_{i_1} n_{i_2}} \]

\[ + \sum_{i_1=1}^{k-2} \sum_{i_2=i_1+1}^{k-1} \sum_{i_3=i_2+1}^{k} \frac{(i_2 - i_1)(i_3 - i_2)}{6n_{i_1} n_{i_2}} + \frac{(i_3 - i_2)(i_1 - i_2)}{6n_{i_2}} + \frac{(i_1 - i_2)(i_2 - i_3)}{6n_{i_3}} \]

In the KTP test, Spearman’s rank correlation coefficient \( r_s \) is given by the following formula:
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:

$$r_s = 1 - \frac{6 \sum_{i=1}^{k} d_i^2}{k(k^2 - 1)}$$

$S$ test

Shan et al. (2014) proposed the new rank-based nonparametric test by incorporating the actual differences as follows:

$$S = \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} D_{ij}$$

(13)

where

$$D_{ij} = \sum_{m=1}^{n_i} \sum_{l=1}^{n_j} Z_{ijml}, \quad Z_{ijml} = (R_{jm} - R_{il})I(X_{jm} > X_{il})$$

and $R_{ij}(R_{jm})$ is the rank of observation $X_{ij}(X_{jm})$ in the combined data.

Under $H_0$, the statistic $S$ has a normal distribution with the following mean and variance:

$$E(S) = \frac{N + 1}{6} \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} n_i n_j$$

$$V(S) = \left(\frac{N^2 + N}{12} - \frac{(N + 1)^2}{36}\right) \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} n_i n_j$$

$$+ 2 \left[ \sum_{i=1}^{k-1} n_i \left( \sum_{l=1}^{n_i} \frac{n_l}{2} \right) + \sum_{j=i+1}^{k} n_j \left( \sum_{l=1}^{n_j} \frac{n_l}{2} \right) \right] \text{Cov}A + 2 \left( \sum_{i=1}^{k-2} \sum_{j=i+1}^{k-1} \sum_{l=j+1}^{k} n_i n_j n_l \right) \text{Cov}B$$

where $\text{Cov}A = \frac{2N^2 + N - 1}{90}$, and $\text{Cov}B = \frac{-7N^2 - 11N - 4}{360}$.

Gaur’s $G_c$ test

Let $(w_1, w_2, ..., w_{k-1})$ be suitably selected real positive constants. Gaur (2017) proposed the $G_c$ statistic to test $H_0$ against the ordered alternatives:

$$G_c = \sum_{g=1}^{k-1} w_g V_{g,g+1}$$

(14)

where

$$V_{g,h} = \left[ \begin{pmatrix} n_g \\ c \end{pmatrix} \begin{pmatrix} n_h \\ c \end{pmatrix} \right]^{-1} \sum_{0}^{\min(n_g,n_h)} \phi_{gh}(X_{g_1},...,X_{g_{n_g}},X_{h_{1}},...,X_{h_{n_h}})$$
for $g < h$; $h = 1, 2, ..., k$; \( \sum_{0}^{\sum_{0}} \) is the sum over all combinations \((a_1, ..., a_c)\) of \(c\) integers selected from \((1, ..., n_g)\) and over all combinations \((b_1, ..., b_c)\) of \(c\) integers selected from \((1, ..., n_h)\);

\[
\phi_{gh}(X_{g1}, ..., X_{gk}; X_{h1}, ..., X_{hk}) = \begin{cases} 
1 & \text{if } \max(X_{g1}, ..., X_{gk}) \leq \min(X_{h1}, ..., X_{hk}) \\
-1 & \text{if } \max(X_{h1}, ..., X_{hk}) \leq \min(X_{g1}, ..., X_{gk}) \\
0 & \text{otherwise} 
\end{cases}
\]

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

\[
V(G_c) = w^T \sum w
\]

where \( w^T = (w_1, w_2, ..., w_k) \) and \( \sum = [\sigma_{gh}] \) is the variance-covariance matrix, such as:

\[
\sigma_{gh} = \begin{cases} 
\left( \frac{(c-1)c}{2c-1} \right)^2 \left( \frac{1}{y} + \frac{1}{x_{g+1}} \right) \delta_c & \text{for } g = h, 1, 2, ..., k - 1 \\
- \left( \frac{(c-1)c}{2c-1} \right)^2 \frac{d}{x_{g+1}} & \text{for } h = g + 1; g = 1, 2, ..., k - 2 \\
- \left( \frac{(c-1)c}{2c-1} \right)^2 \frac{d}{x_y} & \text{for } h = g - 1; g = 2, 2, ..., k - 1 \\
0 & \text{otherwise}
\end{cases}
\]

where

\[
\delta_c = -1 + \frac{4}{4c-1} \sum_{i=c}^{2c-1} \sum_{j=c}^{2c-1} \left( \begin{array}{c} 2c-1 \\ i \end{array} \right) \left( \begin{array}{c} 2c-1 \\ j \end{array} \right) \left( \frac{4c-2}{i+j} \right)^{-1}.
\]

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

\[
w_g = \frac{g(k-g)}{2k}, \ g = 1, 2, ..., k - 1.
\]

**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).

Neuhauser’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.

![Boxplots for the datasets.](image)

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.
Tests

Using the datasets which are named jdata and lehmann, demonstration of the tests are given below, respectively.

Jonkheere-Terpstra test: JtTest(...) 

The JtTest function in the npordtests package is used to perform the Jonkheere-Terpstra test.

R> data(jdata)
R> JtTest(Y~X,jdata,alpha=0.05,na.rm=TRUE,verbose=TRUE)

---------------------------------------------------------
Test : Jonckheere-Terpstra Test
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)
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)

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)

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)
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

The R Journal Vol. 12/1, June 2020 ISSN 2073-4859
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)
```


```
```

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. Because this p-value is smaller than \( \alpha = 0.05 \), the hypothesis of the equality of locations against the ordered alternative is rejected.
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 $\frac{KTP}{\sqrt{V(KTP)}}$. 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 $\frac{G_c}{\sqrt{V(G_c)}}$. 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
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
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
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 Gc test. The p-value for the Gc 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 \((k = 3 \text{ and } 4)\),
• average number of observations per group \((n = 5, 10, 20, 30, \text{ and } 50)\),
• 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 \( \log \cdot F(v_1, v_2) \) distributions to generate the random variable \( X_{ij} = \theta_i + \varepsilon_{ij} \), where \( \varepsilon_{ij} \) is the \( \text{iid log} \cdot F \) distribution, and \( \theta_i \) 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.

<table>
<thead>
<tr>
<th>Sample size patterns</th>
<th>Progressive</th>
<th>Equal</th>
<th>One extreme</th>
</tr>
</thead>
<tbody>
<tr>
<td>( k = 3 )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4 9 19 29 49 5 10 20 30 50</td>
<td>2 4 8 12 20</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>5 10 20 30 50 5 10 20 30 50</td>
<td>2 4 8 12 20</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>6 11 21 31 51 5 10 20 30 50</td>
<td>11 22 44 66 110</td>
<td></td>
</tr>
<tr>
<td>Average n</td>
<td>5 10 20 30 50</td>
<td>5 10 20 30 50</td>
<td>5 10 20 30 50</td>
</tr>
<tr>
<td>( k = 4 )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2 7 14 21 35 5 10 20 30 50</td>
<td>3 6 12 18 30</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4 9 18 27 45 5 10 20 30 50</td>
<td>3 6 12 18 30</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>6 11 22 33 55 5 10 20 30 50</td>
<td>3 6 12 18 30</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>8 13 26 39 65 5 10 20 30 50</td>
<td>11 22 44 66 110</td>
<td></td>
</tr>
<tr>
<td>Average n</td>
<td>5 10 20 30 50</td>
<td>5 10 20 30 50</td>
<td>5 10 20 30 50</td>
</tr>
</tbody>
</table>

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 \( \pm 0.5 \alpha \) 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.

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 \( \log \cdot F(5, 5) \), 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.
Table 2: Simulated power values \((1 - \beta)\) of the test for log-F\((5,5)\) distribution.

| k  | Test | 5    | 10   | 20   | 30   | 50   | 5    | 10   | 20   | 30   | 50   |
|----|------|------|------|------|------|------|------|------|------|------|------|------|
|    |      | 5    | 10   | 20   | 30   | 50   | 5    | 10   | 20   | 30   | 50   | 5    | 10   | 20   | 30   | 50   |
| 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 |
|    | M JT | .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 |
|    | F MT | .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 | .906 | .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 | .766 | .963 | .999 | 1.00 | .691 | .999 | 1.00 | .375 | .595 | .847 | .973 | .995 |
|    | AT   | .616 | .939 | .999 | 1.00 | .708 | .952 | .999 | 1.00 | .700 | .953 | 1.00 | .233 | .439 | .757 | .841 | .902 |
|    | M JT | .678 | .961 | .999 | 1.00 | .784 | .978 | .999 | 1.00 | .782 | .967 | 1.00 | .264 | .549 | .872 | .999 | 1.00 |
|    | TM   | .597 | .894 | .927 | .990 | .999 | .586 | .815 | .999 | 1.00 | .999 | 1.00 | .348 | .661 | .969 | .988 | .992 |
|    | F MT | .592 | .958 | .999 | 1.00 | .775 | .965 | .999 | 1.00 | .731 | .951 | 1.00 | .595 | .869 | .992 | 1.00 | 1.00 |
|    | KTP  | .510 | .742 | .934 | .992 | 1.00 | .439 | .716 | .932 | .999 | 1.00 | .375 | .595 | .847 | .973 | .999 |
|    | S    | .485 | .916 | .999 | 1.00 | .630 | .948 | .999 | 1.00 | .455 | .869 | 1.00 | .595 | .869 | .992 | 1.00 | 1.00 |

The R Journal Vol. 12/1, June 2020 ISSN 2073-4859
Table 3: Simulated power values \((1 - \beta)\) of the test for log-F\((1,10)\) distribution.

<table>
<thead>
<tr>
<th>Sample size pattern</th>
<th>Progressiv</th>
<th>Equal</th>
<th>One Extreme</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Average n</td>
<td>Average n</td>
<td>Average n</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>10</td>
<td>20</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>3 JT</td>
<td>.179</td>
<td>.309</td>
<td>.495</td>
</tr>
<tr>
<td></td>
<td>AT</td>
<td>.181</td>
<td>.314</td>
<td>.554</td>
</tr>
<tr>
<td></td>
<td>MJT</td>
<td>.198</td>
<td>.307</td>
<td>.496</td>
</tr>
<tr>
<td></td>
<td>TM</td>
<td>.214</td>
<td>.319</td>
<td>.474</td>
</tr>
<tr>
<td></td>
<td>FTM</td>
<td>.187</td>
<td>.313</td>
<td>.484</td>
</tr>
<tr>
<td></td>
<td>KTP</td>
<td>.193</td>
<td>.303</td>
<td>.499</td>
</tr>
<tr>
<td></td>
<td>S</td>
<td>.188</td>
<td>.303</td>
<td>.481</td>
</tr>
<tr>
<td></td>
<td>Gc</td>
<td>.156</td>
<td>.284</td>
<td>.451</td>
</tr>
<tr>
<td></td>
<td>4 JT</td>
<td>.293</td>
<td>.538</td>
<td>.801</td>
</tr>
<tr>
<td></td>
<td>AT</td>
<td>.282</td>
<td>.578</td>
<td>.851</td>
</tr>
<tr>
<td></td>
<td>MJT</td>
<td>.285</td>
<td>.545</td>
<td>.802</td>
</tr>
<tr>
<td></td>
<td>TM</td>
<td>.302</td>
<td>.473</td>
<td>.761</td>
</tr>
<tr>
<td></td>
<td>FTM</td>
<td>.231</td>
<td>.533</td>
<td>.772</td>
</tr>
<tr>
<td></td>
<td>KTP</td>
<td>.247</td>
<td>.530</td>
<td>.770</td>
</tr>
<tr>
<td></td>
<td>S</td>
<td>.267</td>
<td>.510</td>
<td>.769</td>
</tr>
<tr>
<td></td>
<td>Gc</td>
<td>.173</td>
<td>.446</td>
<td>.710</td>
</tr>
<tr>
<td></td>
<td>3 JT</td>
<td>.212</td>
<td>.323</td>
<td>.500</td>
</tr>
<tr>
<td></td>
<td>AT</td>
<td>.227</td>
<td>.352</td>
<td>.566</td>
</tr>
<tr>
<td></td>
<td>MJT</td>
<td>.209</td>
<td>.317</td>
<td>.499</td>
</tr>
<tr>
<td></td>
<td>FTM</td>
<td>.191</td>
<td>.295</td>
<td>.485</td>
</tr>
<tr>
<td></td>
<td>KTP</td>
<td>.187</td>
<td>.299</td>
<td>.481</td>
</tr>
<tr>
<td></td>
<td>S</td>
<td>.217</td>
<td>.324</td>
<td>.501</td>
</tr>
<tr>
<td></td>
<td>Gc</td>
<td>.164</td>
<td>.288</td>
<td>.442</td>
</tr>
<tr>
<td></td>
<td>4 JT</td>
<td>.264</td>
<td>.349</td>
<td>.555</td>
</tr>
<tr>
<td></td>
<td>AT</td>
<td>.244</td>
<td>.371</td>
<td>.614</td>
</tr>
<tr>
<td></td>
<td>MJT</td>
<td>.223</td>
<td>.331</td>
<td>.521</td>
</tr>
<tr>
<td></td>
<td>TM</td>
<td>.224</td>
<td>.357</td>
<td>.560</td>
</tr>
<tr>
<td></td>
<td>FTM</td>
<td>.140</td>
<td>.257</td>
<td>.546</td>
</tr>
<tr>
<td></td>
<td>KTP</td>
<td>.142</td>
<td>.285</td>
<td>.509</td>
</tr>
<tr>
<td></td>
<td>S</td>
<td>.251</td>
<td>.335</td>
<td>.538</td>
</tr>
</tbody>
</table>

The R Journal Vol. 12/1, June 2020 ISSN 2073-4859
C ONTRIBUTED R ESEARCH A RTICLES

166

Table 4: Simulated power values (1 − β) of the test for log-F(10,1) distribution.

k

Test

5

3

JT
AT
MJT
TM
FTM
KTP
S
Gc
JT
AT
MJT
TM
FTM
KTP
S
Gc

.190
.182
.208
.215
.187
.184
.199
.153
.305
.319
.304
.313
.262
.269
.295
.193

JT
AT
MJT
TM
FTM
KTP
S
Gc
JT
AT
MJT
TM
FTM
KTP
S
Gc

.206
.176
.210
.182
.174
.178
.219
.135
.254
.227
.221
.198
.149
.155
.257
.109

JT
AT
MJT
TM
FTM
KTP
S
Gc
JT
AT
MJT
TM
FTM
KTP
S
Gc

.165
.179
.192
.226
.196
.194
.177
.150
.104
.131
.117
.208
.160
.160
.103
.144

4

3

4

3

4

Sample size pattern
Progressive
Equal
Average n
Average n
10 20 30 50 5
10 20 30 50
ordered alternatives shape=linear
.310 .491 .682 .893 .179 .298 .491 .690 .909
.315 .526 .747 .976 .203 .323 .527 .741 .967
.304 .490 .690 .896 .191 .318 .491 .676 .863
.293 .465 .645 .845 .221 .298 .438 .590 .750
.280 .482 .700 .924 .186 .290 .463 .642 .827
.304 .484 .682 .888 .185 .305 .493 .689 .897
.305 .477 .663 .863 .195 .304 .487 .688 .907
.279 .454 .649 .856 .168 .263 .458 .655 .862
.537 .796 1.00 1.00 .340 .554 .819 1.00 1.00
.570 .834 1.00 1.00 .344 .581 .851 1.00 1.00
.544 .801 1.00 1.00 .342 .550 .819 1.00 1.00
.485 .800 1.00 1.00 .319 .478 .812 1.00 1.00
.535 .812 1.00 1.00 .337 .557 .822 1.00 1.00
.505 .785 1.00 1.00 .339 .566 .835 1.00 1.00
.518 .804 1.00 1.00 .332 .532 .805 1.00 1.00
.442 .697 .960 1.00 .266 .533 .760 .999 1.00
ordered alternatives shape=convex
.310 .505 .714 .933 .167 .286 .474 .670 .880
.307 .510 .757 1.00 .175 .289 .501 .725 .953
.310 .497 .696 .913 .177 .302 .487 .690 .895
.235 .406 .587 .788 .186 .271 .394 .527 .678
.253 .503 .723 .956 .167 .304 .499 .708 .935
.297 .481 .673 .873 .174 .303 .484 .683 .890
.320 .501 .692 .897 .189 .311 .484 .661 .852
.263 .443 .629 .821 .154 .253 .446 .657 .882
.344 .566 .806 1.00 .161 .260 .447 .648 .855
.326 .546 .782 1.00 .156 .262 .451 .642 .837
.317 .522 .747 .992 .171 .276 .452 .662 .882
.208 .540 .892 1.00 .164 .211 .365 .521 .679
.280 .556 .834 1.00 .172 .272 .444 .636 .842
.250 .515 .800 1.00 .195 .268 .450 .652 .870
.354 .572 .898 1.00 .162 .257 .437 .629 .841
.213 .331 .461 .601 .145 .242 .383 .526 .675
ordered alternatives shape=concave
.286 .467 .654 .857 .179 .299 .485 .675 .877
.316 .505 .696 .903 .216 .331 .524 .737 .952
.286 .475 .670 .883 .184 .306 .480 .670 .862
.341 .508 .700 .916 .223 .339 .528 .741 .960
.306 .474 .662 .868 .180 .300 .498 .710 .930
.313 .486 .663 .842 .184 .310 .471 .644 .825
.290 .477 .674 .877 .204 .306 .486 .682 .894
.288 .453 .638 .831 .162 .272 .420 .588 .760
.213 .324 .447 .576 .180 .280 .456 .638 .822
.257 .416 .589 .778 .188 .318 .480 .644 .810
.232 .365 .518 .679 .179 .281 .455 .637 .839
.276 .425 .576 .737 .181 .271 .445 .635 .841
.271 .426 .585 .746 .192 .301 .466 .641 .826
.207 .322 .455 .600 .184 .273 .440 .609 .798
.246 .345 .446 .557 .174 .289 .398 .523 .654

The R Journal Vol. 12/1, June 2020

5

One Extreme
Average n
10 20 30

50

.180
.204
.176
.213
.158
.178
.208
.127
.350
.379
.360
.297
.323
.334
.364
.207

.256
.302
.266
.251
.242
.249
.291
.195
.555
.594
.558
.428
.510
.511
.576
.430

.404
.473
.415
.350
.390
.393
.433
.324
.806
.848
.816
.788
.797
.782
.823
.670

.566
.654
.574
.455
.540
.553
.593
.461
1.00
1.00
1.00
1.00
1.00
1.00
1.00
.916

.742
.847
.753
.564
.698
.721
.773
.614
1.00
1.00
1.00
1.00
1.00
1.00
1.00
1.00

.232
.228
.214
.193
.162
.175
.261
.113
.284
.263
.247
.158
.169
.167
.304
.113

.337
.349
.320
.216
.228
.245
.381
.187
.455
.425
.397
.202
.259
.279
.479
.226

.533
.530
.482
.300
.354
.379
.573
.315
.697
.670
.626
.324
.432
.462
.728
.332

.741
.729
.646
.396
.492
.533
.767
.463
.941
.929
.857
.464
.617
.649
.995
.452

.963
.948
.816
.510
.646
.695
.973
.621
1.00
1.00
1.00
.620
.818
.850
1.00
.580

.139
.176
.153
.187
.160
.183
.157
.152
.138
.167
.156
.186
.171
.172
.140
.141

.178
.249
.218
.265
.211
.250
.208
.181
.191
.263
.232
.287
.269
.283
.200
.213

.265
.375
.329
.398
.385
.393
.304
.353
.295
.406
.356
.441
.410
.432
.302
.329

.354
.521
.458
.593
.573
.548
.412
.541
.405
.561
.500
.613
.561
.601
.424
.455

.445
.673
.605
.782
.771
.707
.532
.731
.535
.720
.662
.795
.724
.778
.554
.583

ISSN 2073-4859


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.

- For the data generated from the log-$F(1, 10)$ distribution which is 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 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 skewed to the right, Table 4 shows that the AT test for ordered alternatives with linear shape, generally, gives 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.

<table>
<thead>
<tr>
<th>Alternative hypothesis shape</th>
<th>Distribution shape</th>
<th>Sample size pattern</th>
<th>Average sample size</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear</td>
<td>symmetric</td>
<td>-</td>
<td>-</td>
<td>MJT</td>
</tr>
<tr>
<td></td>
<td>skewed to left</td>
<td>-</td>
<td>5, 10</td>
<td>TM</td>
</tr>
<tr>
<td></td>
<td>skewed to left</td>
<td>-</td>
<td>20, 30, 50</td>
<td>AT</td>
</tr>
<tr>
<td></td>
<td>skewed to right</td>
<td>-</td>
<td>AT</td>
<td></td>
</tr>
<tr>
<td>Convex</td>
<td>symmetric</td>
<td>-</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td></td>
<td>skewed to left</td>
<td>-</td>
<td>AT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>skewed to right</td>
<td>-</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>Concave</td>
<td>symmetric</td>
<td>Equal</td>
<td>-</td>
<td>KTP</td>
</tr>
<tr>
<td></td>
<td>skewed to left</td>
<td>-</td>
<td>-</td>
<td>TM</td>
</tr>
<tr>
<td></td>
<td>skewed to left</td>
<td>10, 20, 30, 50</td>
<td>KTP</td>
<td></td>
</tr>
<tr>
<td></td>
<td>skewed to right</td>
<td>-</td>
<td>TM</td>
<td></td>
</tr>
</tbody>
</table>

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 Gc 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


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
Ispartition: Partitioning-Based Least Squares Regression
by Matias D. Cattaneo, Max H. Farrell and Yingjie Feng

Abstract Nonparametric partitioning-based least squares regression is an important tool in empirical work. Common examples include regressions based on splines, wavelets, and piecewise polynomials. This article discusses the main methodological and numerical features of the R software package Ispartition, which implements results for partitioning-based least squares (series) regression estimation and inference from Cattaneo and Farrell (2013) and Cattaneo, Farrell, and Feng (2020). These results cover the multivariate regression function as well as its derivatives. First, the package provides data-driven methods to choose the number of partition knots optimally, according to integrated mean squared error, yielding optimal point estimation. Second, robust bias correction is implemented to combine this point estimator with valid inference. Third, the package provides estimates and inference for the unknown function both pointwise and uniformly in the conditioning variables. In particular, valid confidence bands are provided. Finally, an extension to two-sample analysis is developed, which can be used in treatment-control comparisons and related problems.

Introduction

Nonparametric partitioning-based least squares regression estimation is an important method for estimating conditional expectation functions in statistics, economics, and other disciplines. These methods first partition the support of covariates and then construct a set of local basis functions on top of the partition to approximate the unknown regression function or its derivatives. Empirically popular basis functions include splines, compactly supported wavelets, and piecewise polynomials. For textbook reviews on classical and modern nonparametric regression methodology see, among others, Fan and Gijbels (2018), Györfi, Kohler, Krzyżak, and Walk (2002), Ruppert, Wand, and Carroll (2003), and Harezlak, Ruppert, and Wand (2018). For a review on partitioning-based approximations in nonparametrics and machine learning see Zhang and Singer (2010) and references therein.

This article gives a detailed discussion of the software package Ispartition, available for R, which implements partitioning-based least squares regression estimation and inference. This package offers several features which improve on existing tools, leveraging the recent results of Cattaneo and Farrell (2013) and Cattaneo, Farrell, and Feng (2020), and delivering data-driven methods to easily implement partitioning-based estimation and inference, including optimal tuning parameter choices and uniform inference results such as confidence bands. We cover splines, compactly supported wavelets, and piecewise polynomials, in a unified way, encompassing prior methods and routines previously unavailable without manual coding by researchers. Piecewise polynomials generally differ from splines and wavelets in that they do not enforce global smoothness over the partition, but in the special cases of zero-degree bases on a tensor-product partition, the three basis choices (i.e., zero-degree spline, Haar wavelet, and piecewise constant) are equivalent.

The first contribution offered by Ispartition is a data-driven choice of the number of partitioning knots that is optimal in an integrated mean squared error (IMSE) sense. A major hurdle to practical implementation of any nonparametric estimator is tuning parameter choice, and by offering several feasible IMSE-optimal methods for splines, compactly supported wavelets, and piecewise polynomials, Ispartition provides practitioners with tools to overcome this important implementation issue.

However, point estimation optimal tuning parameter choices yield invalid inference in general, and the IMSE-optimal choice is no exception. The second contribution of Ispartition is the inclusion of robust bias correction methods, which allow for inference based on optimal point estimators. Ispartition implements the three methods studied by Cattaneo, Farrell, and Feng (2020), which are based on novel bias expansions therein. Both the bias and variance quantities are kept in pre-asymptotic form, yielding better bias correction and standard errors robust to conditional heteroskedasticity of unknown form. Collectively, this style of robust bias correction has been proven to yield improved inference in other nonparametric contexts (Calonico, Cattaneo, and Farrell, 2018, 2020).

The third main contribution is valid inference, both pointwise and uniformly in the support of the conditioning variables. When robust bias correction is employed, this inference is valid for the IMSE-optimal point estimator, allowing the researcher to combine an optimal partition for point estimation and a “faithful” measure of uncertainty (i.e., one that uses the same nonparametric estimation choices, here captured by the partition). In particular, Ispartition delivers valid confidence bands that cover the entire regression function and its derivatives. These data-driven confidence bands are constructed by approximating the distribution of t-statistic processes, using either a plug-in approach or a bootstrap
approach. Importantly, the construction of confidence bands does not employ (asymptotic) extreme value theory, but instead uses the strong approximation results of Cattaneo, Farrell, and Feng (2020), which perform substantially better in samples of moderate size.

Last but not least, the package also offers a convenient function to implement estimation and inference for linear combinations of regression estimators of different groups with all the features mentioned above. This function can be used to analyze conditional treatment effects in random control trials in particular, or for two-sample comparisons more generally. For example, a common question in applications is whether two groups have the same “trend” in a regression function, and this is often answered in a restricted way by testing a single interaction term in a (parametric) linear model. In contrast, \textit{lspartition} delivers a valid measure of this difference nonparametrically and uniformly over the support of the conditioning variables, greatly increasing its flexibility in applications.

All of these contributions are fully implemented for splines, wavelets, and piecewise polynomials through the following four functions included in the package \texttt{lspartition}:

- \texttt{lsprobust()}. This function implements estimation and inference for partitioning-based least squares regression. It takes the partitioning scheme as given, and constructs point and variance estimators, bias correction, conventional and robust bias-corrected confidence intervals, and simulation-based conventional and robust bias-corrected uniform inference measures (e.g., confidence bands). Three approximation bases are provided: B-splines, Cohen-Daubechies-Vial wavelets, and piecewise polynomials. When the partitioning scheme is not specified, the companion function \texttt{lspkselect()} is used to select a tensor-product partition in a fully data-driven fashion.

- \texttt{lspkselect()}. This function implements data-driven procedures to select the number of knots for partitioning-based least squares regression. It allows for evenly-spaced and quantile-spaced knot placements, and computes the corresponding IMSE-optimal choices. Two selectors are provided: rule of thumb (ROT) and direct plug-in (DPI) rule.

- \texttt{lsplincom()}. This function implements estimation and robust inference procedures for linear combinations of regression estimators of multiple groups based on \texttt{lsprobust()}. Given a user-specified linear combination, it offers all the estimation and inference methods available in the functions \texttt{lsprobust()} and \texttt{lspkselect()}.

- \texttt{lsprobust.plot()}. This function builds on \texttt{ggplot2} (Wickham and Chang, 2016), and is used as a wrapper for plotting results. It plots regression function curves, robust bias-corrected confidence intervals and uniform confidence bands, among other possibilities.

The paper continues as follows. The next section describes the basic setup including a brief introduction to partitioning-based least squares regression and the empirical example to be used throughout to illustrate features of \texttt{lspartition}. The third section discusses data-driven IMSE-optimal selection of the number of knots and gives implementation details. Estimation and inference implementation is covered in the fourth section, including bias correction methods. The last section provides concluding remarks. We defer to Cattaneo, Farrell, and Feng (2020, CFF hereafter) for complete theoretical and technical details. Statements below are sometimes specific versions of a general case therein.

**Setup**

We assume that \((y_i, x_i') : 1 \leq i \leq n\) is an observed random sample of a scalar outcome \(y_i\) and a \(d\)-vector of covariates \(x_i \in \mathcal{X} \subset \mathbb{R}^d\). The object of interest is the regression function \(\mu(x) = \mathbb{E}[y_i | x_i = x]\) or its derivative, the latter denoted by \(\partial^q \mu(x) = \partial^q \mu(x) / \partial x_1^{q_1} \cdots \partial x_d^{q_d}\), for a \(d\)-tuple \(q = (q_1, \ldots, q_d)' \in \mathbb{Z}_+^d\), with \(|q| = \sum_{j=1}^d q_j\).

Estimation and inference is based on least squares regression of \(y_i\) on set of basis functions of \(x_i\), which are themselves built on top of a partition of the support \(\mathcal{X}\). A partition, denoted by \(\Delta = \{\delta_l \in \mathcal{X} : 1 \leq l \leq \kappa\}\), is a collection of \(k\) disjoint open sets such that the closure of their union is \(\mathcal{X}\). For a partition, a set of basis functions, each of order \(m\) and denoted by \(p(x)\), is constructed so that each individual function (i.e., each element of the vector \(p(x)\)) is nonzero on a fixed number of contiguous \(\delta_l\). \texttt{lspartition} allows for three such bases: piecewise polynomials, B-splines, and Cohen-Daubechies-Vial wavelets (Cohen, Daubechies, and Vial, 1993). For the first two bases, the order \(m\) of the basis can be any positive integer, and any derivative of \(\mu\) up to total order \((m-1)\) can be estimated employing such a basis. For wavelets, the current version allows for \(m \leq 4\) (i.e., up to cubic wavelets), and \(q = (0, \ldots, 0)\). The package takes \(m = 2\) (linear basis) as default. To fix ideas, consider \(d = 1\) with piecewise constants. Each \(\delta_l\) is an interval and \(p(x)\) collects all the indicator functions \(1\{x \in \delta_l\}\), \(1 \leq l \leq \kappa\).
Once the basis \( p(x) \) is constructed, the final estimator of \( \partial^q \mu(x) \), for \( |q| < m \), is
\[
\hat{\partial^q \mu}(x) = \partial^q p(x)/\hat{\beta}, \quad \text{where} \quad \hat{\beta} = \arg \min_{b \in \mathbb{R}^k} \sum_{i=1}^n (y_i - p(x_i)'b)^2.
\]

When \( q = 0 \), we write \( \hat{\mu}(\cdot) = \hat{\partial^0 \mu}(\cdot) \) for simplicity.

The approximation power of such estimators increases with the granularity of the partition \( \Delta \) and the order \( m \). We take the latter as fixed in practice. The most popular structure of \( \Delta \) in applications is a tensor-product form, which partitions each covariate marginally into intervals and then sets \( \Delta \) to be the set of all tensor (Cartesian) products of these intervals (CFF consider more general cases). For this type of partition, the user must choose the number and placement of the partitioning knots in each dimension. \texttt{lspartition} allows for three knot placement types: user-specified, evenly-spaced, and quantile-spaced. In the first case, the user has complete freedom to choose both the number and positions of knots for each dimension. In the latter two cases, the knot placement scheme is pre-specified, and hence only the number of subintervals for each dimension needs to be chosen.

We denote the number of knots in the \( d \) dimensions of the regressor \( x_i \) by \( \kappa = (\kappa_1, \ldots, \kappa_d) \in \mathbb{Z}_+^d \), which can be either specified by users or selected by data-driven procedures (see Section 3 below). Moreover, for wavelet bases, motivated by the standard multi-resolution analysis, we provide an option \( J \) for the regression command \texttt{lsprobust()}, which indicates the resolution level of a wavelet basis. This gives \( \kappa_\ell = 2^J \ell, \ell = 1, \ldots, d \), for a resolution \( J \) (see Chui, 1992, for a review). In any case, the tuning parameter to be chosen is \( \kappa = \kappa_1 \times \cdots \times \kappa_d \). In the next section we choose \( \kappa \) to minimize the IMSE of the estimator (1).

**Package and data**

We will showcase the main aspects of \texttt{lspartition} using a running empirical example. The package is available in R and can be installed as follows:

```r
> install.packages("lspartition", dependencies = TRUE)
> library(lspartition)
```

The data we use come from Capital Bikeshare, and is available at http://archive.ics.uci.edu/ml/datasets/Bike+Sharing+Dataset/. For the first 19 days of each month of 2011 and 2012 we observe the outcome \( \text{count} \), the total number of rentals and the covariates \( \text{atemp} \), the “feels-like” temperature in Celsius, and \( \text{workingday} \), a binary indicator for working days (versus weekends and holidays). The data is summarized as follows.

```r
> data <- read.csv("bikesharing.csv", header = TRUE)
> summary(data)
```

<table>
<thead>
<tr>
<th>count</th>
<th>atemp</th>
<th>workingday</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min. : 1.0</td>
<td>Min. :14.997</td>
<td>Min. :0.0000</td>
</tr>
<tr>
<td>1st Qu. : 42.0</td>
<td>1st Qu. : 5.998</td>
<td>1st Qu. :0.0000</td>
</tr>
<tr>
<td>Median :145.0</td>
<td>Median : 15.997</td>
<td>Median :1.0000</td>
</tr>
<tr>
<td>Mean :191.6</td>
<td>Mean : 15.225</td>
<td>Mean :0.6809</td>
</tr>
<tr>
<td>3rd Qu.:284.0</td>
<td>3rd Qu. :24.999</td>
<td>3rd Qu. :1.0000</td>
</tr>
<tr>
<td>Max. :977.0</td>
<td>Max. : 44.001</td>
<td>Max. :1.0000</td>
</tr>
</tbody>
</table>

We will investigate nonparametrically the relationship between temperature and number of rentals and compare the two groups defined by the type of days:

```r
> y <- data$\text{count}
> x <- data$\text{atemp}
> g <- data$\text{workingday}
```

The sample code that follows will use this designation of \( y, x, \) and \( g \).

**Partitioning scheme selection**

We will now briefly describe the IMSE expansion and its use in tuning parameter selection. To differentiate the original point estimator of (1) and the post-bias-correction estimators, we will add a
The bias expansion for the \( \widehat{\theta}_0(x) \) is:

\[
\mathbb{E}[\widehat{\theta}_0(x)|X] - \theta(x) = \frac{1}{n} \sum \gamma_{q,0}(x)^\prime \sum_0 \gamma_{q,0}(x), \quad \text{where} \quad \sum_0 = \mathbb{E}_n \left[ p(x_i) p(x_i)^\prime \sigma^2(x_i) \right].
\]

Only \( \sigma^2(x_i) \) is unknown here, and will be replaced by a residual-based estimator. In particular \texttt{lspartition} allows for the standard Heteroskedasticity-Consistent (HC) class of estimators via the options \texttt{hc0}, \texttt{hc1}, \texttt{hc2}, \texttt{hc3}. See Long and Ervin (2000) for a review in the context of least squares regression.

**Integrated mean squared error**

In general, for a weighting function \( w(x) \), CFF derive the following (conditional) IMSE expansion:

\[
\text{IMSE} [\widehat{\theta}(\cdot)|X] \approx \frac{1}{n} \gamma_{k,q} + \beta_{k,q},
\]

where the \( n \)-varying quantities \( \gamma_{k,q} \) and \( \beta_{k,q} \) correspond to fixed-\( n \) approximations to the variance and squared bias, respectively, and are asymptotically of order \( \gamma_{k,q} \sim n^{1-2q/d} \) and \( \beta_{k,q} \sim n^{-2(q-n)/d} \).

Under regularity conditions on the partition and basis used, CFF derive explicit leading constants in this expansion. \texttt{lspartition} implements IMSE-minimization for the common simple case where \( \Delta \) is a tensor product of marginally formed intervals where the same number of intervals are used for each dimension. Specifically, \( \Delta = \{ X_\ell = t_{\ell,0} < t_{\ell,1} < \cdots < t_{\ell,k-1} < t_{\ell,k} = x_i \} \) partitions \( X_\ell \) into \( k \) subintervals, and the complete partition \( \Delta = \otimes_{\ell=1}^d \Delta_\ell \), where \( \otimes \) denotes tensor (Cartesian) product.

Thus, the IMSE-optimal number of cells of a tensor-product partition is \( k_{\text{IMSE}} = k_n^{\text{IMSE}} \times n^{-1/q} \).

To select \( k_{\text{IMSE}} \), or equivalently \( \beta_{\text{IMSE}} \), assume that the partitioning knots \( \{ 0 = t_{0,0} < t_{1,1} < \cdots < t_{k,1} = 1 \} \) are generated as quantiles of some marginal distributions \( G_\ell(\cdot), \ell = 1, \ldots, d \), that is, for \( l = 0, 1, \ldots, k \) and \( \ell = 1, \ldots, d \),

\[
t_{l,\ell} = G_\ell^{-1} \left( \frac{l}{k} \right),
\]

where \( G_\ell^{-1}(v) = \inf \{ x \in \mathbb{R} : G_\ell(x) \geq v \} \). Then, the IMSE-optimal choice for \( q = 0 \) is

\[
\beta_{\text{IMSE},0} = \left( \frac{2m \beta_{\text{G},0}}{d \beta_0} \right)^{1/m} n^{1/m}.
\]
where \( \lceil x \rceil \) is a ceiling operator that outputs the smallest integer that is no less than \( x \) and \( \mathcal{B}_{G,0} \) is a (squared) bias term that may depend on the marginals \( G_i \) and, as before, is entirely known up to \( m^\text{th} \) order derivatives: \( \partial^m \mu(x), \lceil u \rceil = m \).

### Implementation details

Two popular choices of partitioning schemes are evenly-spaced partitions (\( k\text{type} = "\text{uni}" \)), which sets \( G_i(\cdot) \) to be the uniform distribution over the support of the data, and quantile-spaced partitions (\( k\text{type} = "\text{qua}" \)), which sets \( G_i(\cdot) \) to be the empirical distribution function of each covariate. The package `lspartition` implements both partitioning schemes, and for each case offers two IMSE-optimal tuning parameter selection procedures: rule of thumb (\( \text{imse-rot} \)) and direct plug-in (\( \text{imse-dpi} \)) choices. We close this section with a brief description of the implementation details and an illustration using real data.

#### Rule-of-Thumb Choice

The rule-of-thumb choice is based on the special case of \( q = 0 \). Let the weighting function \( w(x) \) be the density of \( x_i \). The implementation steps are summarized in the following:

- **Bias constant.** The unknown quantities in the bias constants are: \( \partial^m \mu(\cdot), u \in \Lambda_m \), which is estimated by a global polynomial regression of degree \( (m + 2) \); and the density of \( x_i \), which is either assumed to be uniform or estimated by a trimmed-from-below Gaussian reference model (controlled by the option \( \text{rotnorm} \)).
- **Variance constant.** The unknown quantities in the variance constants are: the conditional variance \( \sigma^2(x) = \mathbb{E}[y_i^2|x_i = x] - (\mathbb{E}[y_i|x_i = x])^2 \), which is estimated by global polynomial regressions of degree \( (m + 2) \); and the density of \( x_i \), which is either assumed to be uniform or estimated by a trimmed-from-below Gaussian reference model.
- **Rule-of-thumb \( \hat{k}_{\text{rot}} \).** Using the above results, a simple rule-of-thumb choice of \( \hat{k} \) is

\[
\hat{k}_{\text{rot}} = \left\lceil \left( \frac{2m \hat{\mathcal{B}}_{G,0}}{\hat{\sigma}^2_0} \right)^{\frac{1}{m+2}} \frac{n}{\hat{\sigma}^2_0} \right\rceil,
\]

where \( \hat{\mathcal{B}}_{G,0} \) and \( \hat{\sigma}^2_0 \) are the estimates of bias and variance constants respectively. While this choice of \( \hat{k} \) is obtained under strong parametric assumptions, it still exhibits the correct convergence rate \( (\hat{k}_{\text{rot}} \asymp n^{\frac{1}{m+2}}) \).

The command `lspkselect()` implements the rule-of-thumb selection (\( k\text{select} = "\text{imse-rot}" \)). For example, we focus on a subsample of bike rentals during working days (\( g = 1 \)), and then the selected number of knots are reported in the following:

```r
> summary(lspkselect(y, x, kselect = "imse-rot", subset = (g == 1 + 1)))
```

```r
Call: lspkselect

Sample size (n) = 7412
Basis function (method) = B-spline
Order of basis point estimation (m) = 2
Order of derivative (deriv) = (0)
Order of basis bias correction (m.bc) = 3
Knot placement (ktype) = Uniform
Knot method (kselect) = imse-rot

-------------------------------------
IMSE-ROT
k   k.bc
-------------------------------------
 5   9

-------------------------------------
```

In this example, for the point estimator based on an evenly-spaced partition, the rule-of-thumb estimate of the IMSE-optimal number of knots is \( k = 5 \), and for the derivative estimators used in bias correction for later inference, the rule-of-thumb choice is \( k.bc = 9 \).
Direct Plug-in Choice

Assuming the weighting \( w(x) \) is equal to the density of \( x_i \), the package \texttt{lspartition} implements a direct-plug-in (DPI) procedure summarized by the following steps.

- **Preliminary choice of \( \hat{k} \).** Implement the rule-of-thumb procedure to obtain \( \hat{k}_{\text{rot}} \).
- **Preliminary regression.** Given the user-specified basis, knot placement scheme, and rule-of-thumb choice \( \hat{k}_{\text{rot}} \), implement a partitioning-based regression of order \( (m + 1) \) to estimate all necessary order-\( m \) derivatives; denote these by \( \partial^m \hat{\mu}_{\text{pre}}(\cdot), [u] = m \).
- **Bias constant.** Construct an estimate \( \hat{B}_{m,q}(\cdot) \) of the leading error \( B_{m,q}(\cdot) \) by replacing \( \partial^m \hat{\mu}(\cdot) \) by \( \partial^m \hat{\mu}_{\text{pre}}(\cdot) \). \( \hat{B}_{m,0}(\cdot) \) can be obtained similarly. Then, use the pre-asymptotic version of the conditional bias to estimate the bias constant:

\[
\hat{\kappa}_{k,q} = \frac{1}{n} \sum_{i=1}^{n} \left( \hat{B}_{m,q}(x_i) - \hat{\gamma}_{q,0}(x_i)\mathbb{E}_{\hat{\epsilon}}[p(x_i)\hat{B}_{m,0}(x_i)] \right)^2.
\]

As mentioned before, for the three bases considered in the package \texttt{lspartition}, the second term in the conditional bias is of smaller order under some additional conditions. We employ this property to simplify the estimate of bias constant for wavelets. For splines and piecewise polynomials, however, users may specify whether the projection of the leading error is taken into account in the selection procedure (see option \texttt{proj}).

- **Variance constant.** Implement a partitioning-based series regression of order \( m \) with \( \hat{k} = \hat{k}_{\text{rot}} \), and then use the pre-asymptotic version of the conditional variance to estimate the variance constant. Specifically, let \( \hat{\epsilon}_i \) be the regression residuals and \( \hat{\Sigma}_0 = \mathbb{E}_{\hat{\epsilon}}[p(x_i)p(x_i)'w_i\hat{\epsilon}_i^2] \) be an estimate of \( \hat{\Sigma}_0 = \mathbb{E}[p(x_i)p(x_i)'\hat{\epsilon}_i^2] \), where different weights \( w_i \) are used to construct different HC variance estimators. Then set

\[
\hat{\gamma}_{k,q} = \frac{1}{n} \sum_{i=1}^{n} \hat{\gamma}_{q,0}(x_i)\hat{\Sigma}_0\hat{\gamma}_{q,0}(x_i).
\]

- **Direct plug-in \( \hat{k} \).** Collecting all these results, a direct plug-in choice of \( \hat{k} \) is

\[
\hat{k}_{\text{dpi}} = \left[ \left( \frac{2(m-[q])}{d+[q]} \right)^{1/(d+2[q])} \left( \hat{\kappa}_{k,q} \right)^{1/(d+2[q])} \hat{\gamma}_{k,q} \right]^{1/(d+2[q])}.
\]

The following shows the results of the direct-plug-in procedure based on the real data:

\[
> \text{summary(lspkselect(y, x, kselect = "imse-dpi", subset = (g == + 1)))}
\]

**Call:** \texttt{lspkselect}

Sample size (\( n \)) = 7412  
Basis function (method) = B-spline  
Order of basis point estimation (\( m \)) = 2  
Order of derivative (\( \text{deriv} \)) = (0)  
Order of basis bias correction (\( m.bc \)) = 3  
Knot placement (\( \text{ktype} \)) = Uniform  
Knot method (\( \text{kselect} \)) = imse-dpi

\[
\begin{array}{c|c}
\text{IMSE-DPI} \\
\hline
\kappa & k.bc \\
\hline
8 & 10
\end{array}
\]

The direct plug-in procedure gives more partitioning knots than the rule-of-thumb, leading to a finer partition. For point estimation, \( \hat{k}_{\text{dpi}} = 8 \) knots are suggested, while for bias correction purpose, it selects \( \hat{k}_{\text{dpi}} = 10 \) knots to estimate derivatives in the leading bias. Quantile-spaced knot placement is obtained by adding \texttt{ktype = "qua"}.
Estimation and inference

This section reviews and illustrates the estimation and inference procedures implemented. A crucial ingredient is the bias correction that allows for valid inference after tuning parameter selection.

Point estimation and bias correction

The estimator \( \hat{\varphi}(x) \) is IMSE-optimal from a point estimation perspective when implemented using the choice \( \kappa_{\text{IMSE}} \) to form \( \Delta \), but conventional inference methods based on this resulting point estimator will be invalid. More precisely, the ratio of bias to standard error in the \( t \)-statistic is non-negligible, requiring either ad-hoc undersmoothing or some form of bias correction. In addition to the (uncorrected) point estimate in (1), the package \texttt{lpam} implements the three bias correction options devised by CFF for valid (pointwise and uniform) inference. All these strategies resort to a higher-order basis, \( \tilde{p}(x) \), of order \( m > m \). The partition \( \Delta \) where \( \tilde{p}(x) \) is built on may be different from \( \Delta \) but need not be. These approaches allow researchers to combine an optimal point estimate \( \hat{\varphi}(x) \) based on the IMSE-optimal \( \kappa_{\text{IMSE}} \) with inference based on the same tuning parameter and partitioning scheme choices.

Our bias correction strategies are based on (2) and (3), where the only unknowns are \( \mu(\cdot) \), \( \varphi(\cdot) \), and \( \varphi(\cdot) \) for \( |u| = m \). These are summarized as follows; see CFF for details.

- **Approach 1: Higher-order-basis bias correction.** Use \( \tilde{p}(x) \) to construct a higher-order least squares estimator \( \hat{\varphi}(x) \) which takes exactly the same form as \( \varphi(\cdot) \) but has less bias. If we substitute \( y_i \) and \( \hat{\varphi}(x) \) for \( \mu(x) \) and \( \varphi(\cdot) \) in (2) respectively and subtract this estimated bias from \( \hat{\varphi}(x) \), the resulting “bias-corrected” estimator is equivalent to \( \varphi(\cdot) \). This option is called by \texttt{bc="bc1"}.

- **Approach 2: Least squares bias correction.** Construct \( \hat{\varphi}(x) \) and substitute it for \( \varphi(\cdot) \) in (2), but replace \( \mu(x) \) by \( \hat{\mu}(x) \) rather than \( y_i \). The least squares-biased estimator \( \hat{\varphi}(x) \) is obtained by subtracting this estimated bias from \( \hat{\varphi}(x) \). The supplement to CFF discusses in detail how this approach relates to higher-order-basis bias correction and when they are equivalent. This option is called by \texttt{bc="bc2"}.

- **Approach 3: Plug-in bias correction.** Referring to (3), use \( \tilde{p}(x) \) to construct \( \hat{\mu}(x) \) for all needed \( u \). Substitute \( \hat{\varphi}(x) \) and \( \hat{\mu}(x) \) for \( \varphi(\cdot) \) and \( \mu(x) \) in (2), respectively. Subtracting this estimated bias from \( \hat{\mu}(x) \) leads to a plug-in bias-corrected estimator \( \hat{\mu}(x) \). This option is called by \texttt{bc="bc3"}.

The optimal (uncorrected) point estimator \( (j = 0) \) and the three bias-corrected estimators \( (j = 1, 2, 3) \) can be written in a unified form for a given \( j = 0, 1, 2, 3 \) as

\[
\hat{\mu}(x) = \hat{\gamma}_{q_j}(x)'E_u[\Pi_j(x)y_i].
\]

These estimators only differ in \( \hat{\gamma}_{q_j}(\cdot) \) and \( \Pi_j(\cdot) \), which depend in different ways on \( \tilde{p}(x) \) and \( \hat{p}(x) \). See CFF for exact formulas.

Pointwise inference

Pointwise inference relies on a Gaussian approximation for the \( t \)-statistics, which holds for any \( j = 0, 1, 2, 3 \):

\[
\hat{I}_j(x) = \frac{\hat{\varphi}(x) - \varphi(\cdot)}{\sqrt{\Omega_j(x)/n}} \sim N(0, 1).
\]

where \( \hat{\Omega}_j(x)/n = \hat{\gamma}_{q_j}(x)'E_u[\hat{\gamma}_{q_j}(x)/n] \) is an estimator of the conditional variance of \( \hat{\varphi}(x) \), and \( \sim \) denotes convergence in distribution. \( \hat{\gamma}_{q_j}(x)'E_u[\Pi_j(x)y_i\hat{\gamma}_{q_j}(x)/n] \) is a consistent estimator of \( \Sigma_j = E[\Pi_j(x)]E_{\Pi_j(x)}[y_i^2\hat{\gamma}_{q_j}(x)/n] \), where \( \hat{\gamma}_{q_j}(x) \) and \( \hat{\gamma}_{q_j}(x)/n \) are additional weights leading to various HC variance estimators. Then nominal 100\( (1 - \alpha) \)-percent symmetric confidence intervals are

\[
\hat{I}_j(x) = \frac{\hat{\varphi}(x) - \Phi_{1 - \alpha/2}\sqrt{\hat{\Omega}_j(x)/n}}{\varphi(\cdot)} + \varphi(\cdot)\Phi_{1 - \alpha/2}\sqrt{\hat{\Omega}_j(x)/n},
\]

where \( \Phi_u \) is the \( u^{th} \) quantile of the standard normal distribution.
For conventional confidence intervals \((j = 0)\), (asymptotically) correct coverage relies on under-smoothing \((k \gg k_{\text{IMSE}})\) that renders the bias negligible relative to the standard error in large samples. Though straightforward in theory, it is difficult to implement in a principled way. In comparison, given the IMSE-optimal tuning parameter, all three bias-corrected estimators \((j = 1, 2, 3)\) have only higher-order bias, and thus the corresponding confidence intervals based on these estimators will have asymptotically correct coverage. Importantly, the Studentization quantity \(\hat{Q}_j(x)/n\) also captures the additional variability introduced by bias correction.

We now illustrate the pointwise inference features of \texttt{lsprobust()} using the bike rental data. The previous result of knot selection based on the DPI procedure will be employed. Specifically, we set \(nknot=8\) for point estimation. For higher-order-basis bias correction \((bc = \text{"bc1"})\), the same number of knots is used to correct bias by default, while for plug-in bias correction \((bc = \text{"bc3"})\), we use 10 knots \((bnknot=10)\) to estimate the higher-order derivatives in the leading bias. One may leave these options unspecified and then the command \texttt{lsprobust()} will automatically implement knot selection using the command \texttt{lspkselect()}.

```r
> est_workday_bc1 <- lsprobust(y, x, neval = 20, bc = "bc1", nknot = 8,
+ subset = (g == 1))
> est_workday_bc3 <- lsprobust(y, x, neval = 20, bc = "bc3", nknot = 8,
+ bnknot = 10, subset = (g == 1))
> summary(est_workday_bc1)
```

<table>
<thead>
<tr>
<th>Eval X1</th>
<th>n</th>
<th>Point Est.</th>
<th>Std. Error</th>
<th>Robust B.C.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>[ 95% C.I. ]</td>
</tr>
<tr>
<td>1</td>
<td>-2.998</td>
<td>7412</td>
<td>90.667</td>
<td>5.316</td>
</tr>
<tr>
<td>2</td>
<td>-0.002</td>
<td>7412</td>
<td>110.509</td>
<td>3.909</td>
</tr>
<tr>
<td>3</td>
<td>1.998</td>
<td>7412</td>
<td>123.937</td>
<td>3.580</td>
</tr>
<tr>
<td>4</td>
<td>3.998</td>
<td>7412</td>
<td>137.364</td>
<td>3.580</td>
</tr>
<tr>
<td>5</td>
<td>5.998</td>
<td>7412</td>
<td>148.437</td>
<td>3.627</td>
</tr>
<tr>
<td>6</td>
<td>7.001</td>
<td>7412</td>
<td>153.989</td>
<td>3.571</td>
</tr>
<tr>
<td>7</td>
<td>11.001</td>
<td>7412</td>
<td>173.306</td>
<td>5.690</td>
</tr>
<tr>
<td>8</td>
<td>11.997</td>
<td>7412</td>
<td>174.599</td>
<td>4.600</td>
</tr>
<tr>
<td>9</td>
<td>13.997</td>
<td>7412</td>
<td>177.194</td>
<td>3.771</td>
</tr>
<tr>
<td>10</td>
<td>15.997</td>
<td>7412</td>
<td>179.789</td>
<td>3.600</td>
</tr>
<tr>
<td>11</td>
<td>17.000</td>
<td>7412</td>
<td>182.743</td>
<td>5.708</td>
</tr>
<tr>
<td>12</td>
<td>18.003</td>
<td>7412</td>
<td>189.044</td>
<td>4.662</td>
</tr>
<tr>
<td>13</td>
<td>19.000</td>
<td>7412</td>
<td>195.303</td>
<td>4.070</td>
</tr>
<tr>
<td>14</td>
<td>22.003</td>
<td>7412</td>
<td>214.165</td>
<td>5.899</td>
</tr>
<tr>
<td>15</td>
<td>24.003</td>
<td>7412</td>
<td>231.911</td>
<td>5.770</td>
</tr>
<tr>
<td>16</td>
<td>24.999</td>
<td>7412</td>
<td>243.335</td>
<td>4.760</td>
</tr>
<tr>
<td>19</td>
<td>30.002</td>
<td>7412</td>
<td>298.199</td>
<td>7.278</td>
</tr>
<tr>
<td>20</td>
<td>32.002</td>
<td>7412</td>
<td>313.696</td>
<td>6.596</td>
</tr>
</tbody>
</table>
The above table summarizes the results for pointwise estimation and inference, including point estimates, conventional standard errors, and robust confidence intervals based on higher-order-basis bias correction for 20 quantile-spaced evaluation points. We can use the companion plotting command \texttt{lsprobust.plot()} to visualize the results:

```r
> lsprobust.plot(est_workday_bc1, xlabel = "Temperature", ylabel = "Number of Rentals", + legendGroups = "Working Days") + theme(text = element_text(size = 17), + legend.position = c(0.15, 0.9))
> ggsave("output/pointwise1.pdf", width = 6.8, height = 5.5)
> lsprobust.plot(est_workday_bc3, xlabel = "Temperature", ylabel = "Number of Rentals") + + theme(text = element_text(size = 17), legend.position = "none")
> ggsave("output/pointwise2.pdf", width = 6.8, height = 5.5)
```

![Figure 1](image)

**Figure 1:** Estimated relationship between the number of rentals (y-axis) and temperature (x-axis) during working days. The solid curves are the point estimates, and the shaded regions are robust confidence intervals. 1a shows the results based on higher-order-basis correction, and 1b shows the results based on plug-in bias correction. We see that as the temperature increases, so does the number of rentals, and that \texttt{lspartition} gives a valid visualization of this trend.

The result is displayed in Figure 1. As the temperature gets higher, the number of rentals increases as expected. Both panels show the same point estimator, \( \hat{\mu}_0 \). We plot both the robust confidence intervals based on higher-order-basis bias correction (Figure 1a) and plug-in bias correction (Figure 1b). Since the higher-order-basis approach is equivalent to a quadratic spline fitting, the resulting confidence interval has a smoother shape.

**Uniform inference**

To obtain uniform inference (over the support of \( x \)), \textbf{CFF} establish Gaussian approximations for the whole \( t \)-statistic processes, and propose several sampling-based approximations which are easy to implement in practice. To be concrete, for each \( j = 0, 1, 2, 3 \), there exists a Gaussian process \( Z_j(\cdot) \) such that \( \hat{T}_j(\cdot) \approx_d Z_j(\cdot) \). This Guassian process is given by

\[
Z_j(\cdot) = \gamma_{q,j}(\cdot) \sum_j^{1/2} \sqrt{\Omega_j(\cdot)} N_{K_j},
\]

where \( K_j = \dim(\Pi_j(\cdot)) \propto K_j, \gamma_{q,j}(\cdot) \) and \( \Omega_j(\cdot) \) are population counterparts of \( \hat{\gamma}_{q,j}(\cdot) \) and \( \hat{\Omega}_j(\cdot) \), and \( N_{K_j} \) is a \( K_j \)-dimensional standard normal random vector. The notation \( \approx_d \) means that the two processes are asymptotically equal in distribution in the following sense: in a sufficiently rich probability space, we have identical copies of \( \hat{T}_j(\cdot) \) and \( Z_j(\cdot) \) whose difference converges in probability to zero uniformly.

The Gaussian stochastic process \( Z_j(\cdot) \) is not feasible in practice because it involves unknown population quantities. Thus, the package \texttt{lspartition} offers two options for implementation: plug-in or bootstrap.
• **Plug-in.** Replace all unknowns in \( Z_j(\cdot) \) by some consistent estimators:

\[
\hat{Z}_j(\cdot) = \frac{\tilde{t}_q(\cdot) q_j^{1/2}}{\sqrt{\Omega_j(\cdot)}} N_{K_j}.
\]

**CFF** shows that \( \hat{Z}_j(\cdot) \) delivers a valid distributional approximation to \( \hat{T}_j(\cdot) \). In practice one may obtain many simulated realizations of \( \hat{Z}_j(\cdot) \) by sampling from the \( K_j \)-dimensional standard normal distribution conditional on the data. This option is called by `uni.method="pl"`.

• **Bootstrap.** Construct a bootstrapped version of the approximation process (conditional on the data):

\[
\hat{Z}_j^*(\cdot) = \frac{\tilde{t}_q(\cdot) q_j^{1/2} \mathbb{E}[\Pi_j(x) \hat{e}_{i,j}^*]}{\sqrt{\hat{\Omega}_j^*(\cdot)/n}},
\]

where \( \hat{\Omega}_j^*(\cdot) = \tilde{t}_q(\cdot) \mathbb{E}[\Pi_j(x) (\hat{e}_{i,j}^*)^2] q_j(\cdot), \hat{e}_{i,j}^* = \omega_i \hat{e}_{i,j} \) and \( \{\omega_i\}_{i=1}^n \) is an i.i.d sequence of bounded random variables with zero mean and unit variance. **CFF** show that this bootstrapped process also approximates \( Z_j(\cdot) \) conditional on the data. Thus one can implement bootstrapping by sampling from the distribution of \( \omega_i \) given the data. In the package `lspartition`, the \( \omega_i \)'s are taken to be Rademacher variables, and this option is called by `uni.method="wb"`.

Importantly, these strong approximations apply to the whole \( t \)-statistic processes, and thus can be used to implement general inference procedures based on transformations of \( \hat{T}_j(\cdot) \). The main regression command `lsprobust()` will output the the following quantities for uniform analyses upon setting `uni.out=TRUE`:

- **t. num.pl**. t. num. wb1, t. num. wb2. The numerators of approximation processes except the “simulated components”, which are evaluated at a set of pre-specified grid points \( K \). Suppose that \( K \) contains \( L \) grid points. Then for the plug-in method, the numerator, stored in `t.num.pl`, is the \( L \times K_j \) matrix \( \{q_j(x) q_j^{1/2}/\sqrt{n} : x \in K\} \). For wild bootstrap, the numerator is separated to `t.num.wb1` and `t.num.wb2`, which are \( \{q_j(x) / n : x \in K\} \) and \( \{\Pi_j(x_1), \ldots, \Pi_j(x_L)\} \) respectively.

- **t. denom.** The denominator of approximation processes, i.e., \( \{\sqrt{\hat{\Omega}_j(\cdot)/n} : x \in K\} \), stored in a vector of length \( L \).

- **res.** Residuals from the specified bias-corrected regression (needed for bootstrap-based approximation).

For example, the following command requests the necessary quantities for uniform inference based on the plug-in method:

```r
> est_workday_bc1 <- lsprobust(y, x, bc = "bc1", nknot = 4, uni.method = "pl",
+ uni.ngrid = 100, uni.out = T, subset = (g == 1))
> round(est_workday_bc1$uni.output$t.num.pl[1:5, ], 3)
[1,] 30.549 -4.923  2.311 -1.470  0.779 -0.451  0.121
[2,] 27.104 -3.553  1.746 -1.162  0.620 -0.354  0.090
[3,] 23.856 -2.285  1.236 -0.880  0.474 -0.266  0.062
[4,] 20.803 -1.117  0.780  0.624  0.341 -0.185  0.037
[5,] 17.946 -0.852  0.379 -0.395  0.221 -0.113  0.014
```

We list the first 5 rows of the numerator matrix. Each row corresponds to a grid point. Since we use a linear spline for point estimation and set `nknot=4`, the higher-order-basis bias correction is equivalent to quadratic spline fitting. Thus the numerator matrix has 7 columns corresponding to the quadratic spline basis.

As a special application, these results can be used to construct uniform confidence bands, which builds on the suprema of \( |\hat{T}_j(\cdot)| \). The function `lsprobust()` computes the critical value to construct confidence bands. Specifically, it generates many simulated realizations of \( \hat{Z}_j(\cdot) \) or \( \hat{Z}_j^*(\cdot) \) using the methods described above, and then obtains an estimated 100(1 - \( \alpha \))-quantile of \( \sup_{x \in X} |\hat{T}_j(x)| \) or \( \sup_{x \in X} |\hat{Z}_j^*(x)| \) given the data, denoted by \( q_j(1 - \alpha) \). Then, (1 - \( \alpha \)) confidence band for \( \hat{\phi}_j(x) \) is given by

\[
\hat{\phi}_j(x) \pm q_j(1 - \alpha) \sqrt{\hat{\Omega}_j(x)/n}.
\]

For example, the following command requests a critical value for constructing confidence bands:
Once the critical value is available, the command `lsprobust.plot()` is able to visualize confidence bands:

```r
> lsprobust.plot(est_workday_bc1, CS = "all", xlabel = "Temperature",
+ ylabel = "Number of Rentals", legendGroups = "Working Days") +
+ theme(text = element_text(size = 17), legend.position = c(0.15,
+ 0.9))
> ggsave("output/uniform1.pdf", width = 6.8, height = 5.5)
```

![Figure 2](output/uniform1.pdf)

**Figure 2:** Estimated relationship between the number of rentals (y-axis) and temperature (x-axis) during working days. The solid curve shows the point estimates, the error bars show the robust confidence intervals, and the shaded region shows the robust confidence band based on the plug-in method with higher-order-basis bias correction. We see that the uniform coverage of the confidence band makes it wider than a single pointwise interval and that `lspartition` allows both to be cleanly displayed.

The result is displayed in Figure 2. Since we set `CS="all"`, the command simultaneously plots pointwise confidence intervals (error bars) and a uniform confidence band (shaded region).

It is also possible to specify other bias correction approaches or uniform methods:

```r
> est_workday_bc3 <- lsprobust(y, x, neval = 20, bc = "bc3", nknot = 8,
+ bnknot = 10, uni.method = "wb", subset = (g == 1), band = T)
> est_workday_bc3$sup.cval
  95%
  3.009244
> lsprobust.plot(est_workday_bc3, CS = "all", xlabel = "Temperature",
+ ylabel = "Number of Rentals", legendGroups = "Working Days") +
+ theme(text = element_text(size = 17), legend.position = c(0.15,
+ 0.9))
> ggsave("output/uniform2.pdf", width = 6.8, height = 5.5)
```

The result is displayed in Figure 3. In this example, the critical values based on different methods are quite close, but in general their difference could be more pronounced in finite samples. See `CFF` for some simulation evidence.
Linear combinations

The package \texttt{lspartition} also includes a function \texttt{lsplincom()}, which implements estimation and inference for a linear combination of regression functions of different subgroups. To be concrete, consider a random trial with \( G \) groups. Let \( \mu(x; g) \) be the conditional expectation function (CEF) for group \( g, \ g = 1, \ldots, G \). The parameter of interest is \( \theta(x) = \sum_{q=1}^{G} r_g \partial_q \mu(x; g) \), i.e., a linear combination of CEFs (or derivatives thereof) for different groups. To fix ideas, consider the most common application, the difference between two groups (or the conditional average treatment effect). Here, \( G = 2, q = 0 \), and \((r_1, r_2) = (-1,1)\). Then \( \theta(x) = E[y|x_i = x, g = 1] - E[y|x_i = x, g = 0] \).

To implement estimation and inference for \( \theta(x) \), \texttt{lsplincom()} first calls \texttt{lsprobust()} to obtain a point estimate \( \hat{\theta}_g(x; g) \) and all other objects for each group. The tuning parameter for each group can be selected by the data-driven procedures above. Then the point estimate of \( \theta(x) \) is

\[
\hat{\theta}(x) = \sum_{g=1}^{G} r_g \hat{\theta}_g(x; g).
\]

The standard error of \( \hat{\theta}(x) \) can be obtained simply by taking the appropriate linear combination of standard errors for each \( \hat{\theta}_g(x; g) \) and their estimated covariances. Robust confidence intervals can be similarly constructed as in (3).

\texttt{lsplincom()} also allows users to construct confidence bands for \( \theta(\cdot) \). Specifically, it requests \texttt{lsprobust()} to output the numerators (\texttt{t.num.pl}) for “plug-in”, or \texttt{t.num.wb1} and \texttt{t.num.wb2} for “bootstrap”) and denominators (\texttt{t.denom}) of the feasible approximation processes \( \hat{Z}_j(\cdot) \) or \( \hat{Z}^*(\cdot) \). Let \( U_j(\cdot; g) \) and \( V_j(\cdot; g) \) denote the numerator and denominator from group \( g \) based on bias correction approach \( j, g = 1, \ldots, G \) and \( j = 1, 2, 3 \). The approximation process for the \( t \)-statistic process based on \( \hat{\theta}_j(x) \) is

\[
\hat{Z}_{j, \theta}(\cdot) = \frac{\sum_{g=1}^{G} r_g U_j(\cdot; g) N_{g, K_{jg}}}{\sqrt{\sum_{g=1}^{G} r^2_g V_j(\cdot; g)^2}},
\]

where \( \{N_{g, K_{jg}}\}_{g=1}^{G} \) is a collection of independent standard normal vectors, and \( K_{jg} \) indicates the dimension of \( N_{g, K_{jg}} \). As discussed before, the dimensionality of these normal vectors depends on the particular bias correction approach and may vary across groups since the selected number of knots may be different across groups. The bootstrap approximation process \( \hat{Z}^*_{j, \theta}(\cdot) \) can be constructed similarly.

Given these processes, inference is implemented by sampling from \( G \) standard normal vectors (“plug-in” method) or \( G \) groups of Rademacher vectors given the data. Then critical values used

Figure 3: Estimated relationship between the number of rentals (y-axis) and temperature (x-axis) during working days. The solid curve shows the point estimates, the error bars show the robust confidence intervals, and the shaded region shows the robust confidence band based on the bootstrap method with plug-in bias correction. Comparing to Figure 2, we see that using the bootstrap to measure uncertainty produces similar results, as expected from the theory.
to construct $100(1 - \alpha)$ confidence bands for $\theta(\cdot)$ are estimated similarly by $100(1 - \alpha)$ empirical quantiles of $\sup_{x \in X} \{Z_{j,\theta}(x)\}$ or $\sup_{x \in X} \{\hat{Z}_{j,\theta}(x)\}$.

As an illustration, we compare the number of rentals during working days and other time periods (weekends and holidays) based on linear splines and plug-in bias correction. To begin with, we first estimate the conditional mean function for each group using the command \texttt{lsprobust}.

```r
> est_workday <- lsprobust(y, x, neval = 20, bc = "bc3", nknot = 8, + subset = (g == 1))
> est_nworkday <- lsprobust(y, x, neval = 20, bc = "bc3", nknot = 8, + subset = (g == 0))
> lsprobust.plot(est_workday, est_nworkday, legendGroups = c("Working Days", + "Nonworking Days"), xlabel = "Temperature", ylabel = "Number of Rentals", + lty = c(1, 2)) + theme(text = element_text(size = 17), legend.position = c(0.2, + 0.85))
> ggsave("output/diff1.pdf", width = 6.8, height = 5.5)
```

The pointwise results for each group are displayed in Figure 4. The shaded regions represent confidence intervals. Clearly, when the temperature is low, two regions are well separated, implying that people may rent bikes more during working days than weekends or holidays when the weather is cold.

![Figure 4: Estimated relationships between the number of rentals (y-axis) and temperature (x-axis) during working and nonworking days. The solid and dashed curves show the point estimates, and the shaded regions show the robust confidence intervals for the two groups. We observe that the two groups appear statistically different at some points, but not all, and testing across the entire range requires uniform, not pointwise, results.](output/diff1.pdf)

Next, we employ the command \texttt{lsplincom()} to formally test this result. We specify $R=(-1,1)$, denoting that $-1$ is the coefficient of the conditional mean function for the group $workingday==0$ and $1$ is the coefficient of the conditional mean function for the group $workingday==1$.

```r
> diff <- lsplincom(y, x, data$workingday, R = c(-1, 1), band = T, + cb.method = "pl")
> summary(diff)

Call: lprobust

Sample size (n) = 10886
Num. covariates (d) = 1
Num. groups (G) = 2
Basis function (method) = B-spline
Order of basis point estimation (m) = 2
Order of derivative (deriv) = (0)
Order of basis bias correction (m.bc) = 3
Smoothness point estimation (smooth) = 0
Smoothness bias correction (bsmooth) = 1
```
Knot placement (ktype) = Uniform
Knots method (kselect) = imse-dpi
Confidence band method (cb.method) = Plug-in

<table>
<thead>
<tr>
<th>Eval X1</th>
<th>Point Est.</th>
<th>Std. Error</th>
<th>[95% C.I.]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-2.998</td>
<td>3.2170</td>
<td>5.2970</td>
</tr>
<tr>
<td>2</td>
<td>-0.002</td>
<td>4.6752</td>
<td>9.2473</td>
</tr>
<tr>
<td>3</td>
<td>1.998</td>
<td>3.9749</td>
<td>6.5553</td>
</tr>
<tr>
<td>4</td>
<td>3.998</td>
<td>2.9883</td>
<td>6.8463</td>
</tr>
<tr>
<td>5</td>
<td>5.998</td>
<td>1.7871</td>
<td>7.6049</td>
</tr>
<tr>
<td>6</td>
<td>7.001</td>
<td>1.6730</td>
<td>8.1210</td>
</tr>
<tr>
<td>7</td>
<td>9.007</td>
<td>3.3120</td>
<td>9.6230</td>
</tr>
<tr>
<td>8</td>
<td>11.997</td>
<td>3.0829</td>
<td>9.3349</td>
</tr>
<tr>
<td>9</td>
<td>13.000</td>
<td>1.5520</td>
<td>7.5400</td>
</tr>
<tr>
<td>10</td>
<td>15.000</td>
<td>1.0060</td>
<td>6.6640</td>
</tr>
<tr>
<td>11</td>
<td>17.000</td>
<td>6.1180</td>
<td>8.3860</td>
</tr>
<tr>
<td>12</td>
<td>18.003</td>
<td>9.1510</td>
<td>3.2970</td>
</tr>
<tr>
<td>13</td>
<td>19.000</td>
<td>9.7460</td>
<td>15.2430</td>
</tr>
<tr>
<td>14</td>
<td>22.003</td>
<td>-1.7530</td>
<td>-20.8910</td>
</tr>
<tr>
<td>15</td>
<td>24.003</td>
<td>-3.2221</td>
<td>-10.0240</td>
</tr>
<tr>
<td>16</td>
<td>24.999</td>
<td>-36.9260</td>
<td>-67.8430</td>
</tr>
<tr>
<td>17</td>
<td>26.002</td>
<td>-31.7600</td>
<td>-57.7130</td>
</tr>
<tr>
<td>18</td>
<td>28.002</td>
<td>-21.3470</td>
<td>-46.3160</td>
</tr>
<tr>
<td>19</td>
<td>30.002</td>
<td>-13.4120</td>
<td>-34.0390</td>
</tr>
<tr>
<td>20</td>
<td>32.002</td>
<td>-15.4380</td>
<td>-34.0390</td>
</tr>
</tbody>
</table>

The pointwise results are summarized in the above table. Clearly, when the temperature is low, the point estimate of the rental difference is significantly positive since the robust confidence intervals do not cover 0. In contrast, when the temperature is above 7, it is no longer significant. This implies that the difference in the number of rentals between working days and other periods is less pronounced when the weather is warm. Again, we can use the command `lsprobust.plot()` to plot point estimates, confidence intervals and uniform band simultaneously:

```r
> lsprobust.plot(diff, CS = "all", xlabel = "Temperature", ylabel = "Number of Rentals", + legendGroups = "Difference between Working and Other Days") + + theme(text = element_text(size = 17), legend.position = c(0.36, + 0.2))
> ggsave("output/diff2.pdf", width = 6.8, height = 5.5)
```

In addition, some basic options for the command `lsprobust()` may be passed on to the command `lsplincom()`. For example, the following code generates a smoother fit of the rental difference by setting m=3:

```r
> diff <- lsplincom(y, x, data$workingday, R = c(-1, 1), band = T, + cb.method = "pl", m = 3)
> lsprobust.plot(diff, CS = "all", xlabel = "Temperature", ylabel = "Number of Rentals") + + theme(text = element_text(size = 17), legend.position = "none")
> ggsave("output/diff3.pdf", width = 6.8, height = 5.5)
```

The results are shown in Figure 5. The confidence band for the difference is constructed based on the plug-in distributional approximation computed previously. It leads to an even stronger conclusion: the entire difference as a function of temperature is significantly positive uniformly over a range of low temperatures since the confidence band is above zero when the temperature is low.
Figure 5: Estimated difference in the number of rentals between working and nonworking days. The y-axis plots the difference in the number of rentals, and the x-axis plots the temperature. The solid curves show the point estimates, error bars show the robust confidence intervals, and the shaded regions show the robust confidence bands. Results in 5a are based on a linear basis, and those in 5b are based on a quadratic basis. The uniformly valid confidence bands used here provide an assessment of the difference between the groups overall, compared to the pointwise results in Figure 4.

Summary

We gave an introduction to the software package `lspartition`, which offers estimation and robust inference procedures (both pointwise and uniform) for partitioning-based least squares regression. In particular, splines, wavelets, and piecewise polynomials are implemented. The main underlying methodologies were illustrated empirically using real data. Finally, installation details, scripts replicating the numerical results reported herein, links to software repositories, and other companion information, can be found in the package’s website:

https://nppackages.github.io/lspartition/.

Bibliography


Matias D. Cattaneo  
Department of Operation Research and Financial Engineering  
Princeton University  
Princeton, NJ 08544  
cattaneo@princeton.edu

Max H. Farrell  
Booth School of Business  
University of Chicago  
Chicago, IL 60637  
max.farrell@chicagobooth.edu

Yingjie Feng  
Department of Politics  
Princeton University  
Princeton, NJ 08544  
yingjief@princeton.edu
Skew-t Expected Information Matrix Evaluation and Use for Standard Error Calculations

by R. Douglas Martin, Chindhanai Uthaisaad and Daniel Z. Xia

Abstract Skew-t distributions derived from skew-normal distributions, as developed by Azzalini and several co-workers, are popular because of their theoretical foundation and the availability of computational methods in the R package \texttt{sn}. One difficulty with this skew-t family is that the elements of the expected information matrix do not have closed form analytic formulas. Thus, we developed a numerical integration method of computing the expected information matrix in the R package \texttt{skewtInfo}. The accuracy of our expected information matrix calculation method was confirmed by comparing the result with that obtained using an observed information matrix for a very large sample size. A Monte Carlo study to evaluate the accuracy of the standard errors obtained with our expected information matrix calculation method, for the case of three realistic skew-t parameter vectors, indicates that use of the expected information matrix results in standard errors as accurate as, and sometimes a little more accurate than, use of an observed information matrix.

Introduction

The primary motivation for this paper was our desire to use a flexible family of skew-t distributions for quantitative finance research, and in particular for empirical studies of the tail fatness and skewness of returns distributions of various classes of assets, e.g., stock returns and hedge fund returns. That said, we hasten to add that skew-t distributions have extensive application uses in the broad areas of physical and social sciences, and in engineering and computer science. We believe that the main results of this paper will be of interest to this broader audience as well as to those interested in finance applications.

We chose to focus on the family of skew-t distributions based on skew-normal distribution, henceforth “skew-t” distributions, for two main reasons. The first reason is the existence of extensive research results on the family, as reflected for example in Azzalini and Capitanio (2003) and Azzalini and Capitanio (2014) (AZC) monograph. The second reason is that research using skew-t distributions based on the skew-normal distribution is greatly facilitated by the existence of the comprehensive and well-supported R package \texttt{sn}.

The expected information matrix for skew-t distributions is a key quantity for a number of purposes, including: (a) computing approximate standard errors of skew-t parameter estimates from the asymptotic covariance matrix evaluated at maximum-likelihood estimates (MLEs); (b) evaluating the large-sample inefficiencies of skew-t MLEs when the distribution is normal; (c) comparing the large-sample performances of non-parametric versus skew-t MLE expected shortfall estimators. See for example Martin and Zhang (2017) for (b) and (c) in the case of symmetric-t distributions, where a closed-form expression exists for the expected information matrix.

Unfortunately, there does not exist a closed form analytic matrix expression for the skew-t information matrix. Therefore, we have developed a numerical integration method for computing the elements of a skew-t information matrix at specified skew-t parameter values, and have implemented the resulting algorithm in the R function \texttt{stInfoMat()} in the R package \texttt{skewtInfo}. Users can install the package with RStudio via the following commands:

\begin{verbatim}
install.packages("devtools")
develtools::install_github("chindhanai/skewtInfo")
\end{verbatim}

Although our particular interest in using the \texttt{skewtInfo} package is focused on applications in quantitative finance, we anticipate that it will be of interest to researchers across many fields of study, e.g., physical and social sciences and engineering and computer science, where skew-t distribution have important applications.

For the purpose of analyzing the resulting expected information matrix (henceforth simply “the information matrix”) and evaluating its use for computing standard errors, we used the \texttt{sn} package maximum penalized-likelihood estimator (MPLE) function \texttt{st.mple()} to compute three sets of skew-t parameter values for three sets of returns, labeled D1, D2, D3 obtained from a single stock with ticker D on three different time segments between December 1991 and September 2015.
The remainder of this paper is organized as follows. The "Skew-t Distribution" section introduces
the skew-t distribution family, and provides the relationship between the skew-t distribution direct
parameters "dp" and the usual moment-based "cp" parameters, i.e., the mean, standard deviation,
skewness and excess kurtosis. The "Maximum Penalized Likelihood Estimation Method" section
briefly describes the sn package MPLE method used to fit skew-t distributions, and uses the method to
compute skew-t parameters for each of the three sets of returns D1, D2, D3, compares the parameters
to those obtained with symmetric-t distribution maximum-likelihood estimates (MLEs), and computes
the values of the cp parameters from the skew-t dp parameters. The "Skew-t Information Matrix"
section discusses our numerical method for computing the information matrix, as implemented in
the package skewtInfo, and describes how we verified by examples the accuracy of our method. That
section also examines the ill-conditioning of the information matrix for the three sets of skew-t
MLEs, in comparison with the use of an observed information matrix, and in comparison with Monte Carlo
"true" standard errors, which are used as true skew-t parameters the MPLEs for stock returns D1, D2,
D3. As a preliminary to this Monte Carlo, the first part of the "Standard Errors of Skew-t Parameter
MLEs" section discusses a problem that arises when using skew-t MPLEs to compute them for a
large number of replicates, and shows how to deal with the problem. Finally, the last section discusses
related future research that needs to be done.

Skew-t distributions

It is well-known that a random variable \(X\) with a standard t-distribution, i.e., with location parameter
of value zero, scale parameter of value one, and \(\nu\) degrees of freedom has the stochastic representation

\[
X = \frac{Z}{\sqrt{\chi^2/\nu}}
\]  

(1)

where \(Z\) follows a standard normal distribution and the independent random variable \(\chi^2\) follows a chi-
squared distribution with \(\nu\) degrees of freedom. The following results and related details concerning
skew-t random variables and their distributions are provided in Azzalini and Capitanio (2014).

A standard skew-t random variable \(Y_{0,1,a,v} \sim ST(0, 1, a, v)\) with slant (skewness) parameter \(a\)
and degrees of freedom parameter \(v\), can be obtained by replacing \(Z\) with an independent standard
skew-normal random variable \(Z_{0,1,a} \sim SN(0, 1, a)\), whose probability density function is

\[
f_{SN}(x; a) = 2 \phi(x) \cdot \Phi(ax)
\]  

(2)

where \(\phi(x)\) is the standard normal density function and \(\Phi(x)\) is the standard normal (cumulative)
distribution function. The resulting standard skew-t random variable

\[
Y_{0,1,a,v} = \frac{Z_{0,1,a}}{\sqrt{\chi^2/\nu}}
\]  

(3)

has the standard skew-t probability density

\[
f_{ST,ald}(x; a, v) = 2 t_v(x) \cdot T_{v+1} \left( ax \cdot \sqrt{\frac{v+1}{v+x^2}} \right)
\]  

(4)

where \(t_v(x)\) is the density function of a standard t-distribution random variable with \(v\) degrees of
freedom, and \(T_v(x)\) is the corresponding cumulative distribution function.

A general skew-t random variable \(Y_{\xi,\omega,a,v}\) with location parameter \(\xi\) and scale parameter \(\omega\), whose
distribution we denote as \(ST(\xi, \omega, a, v)\), has the stochastic representation

\[
Y_{\xi, \omega, a, v} = \xi + \omega Y_{0,1,a,v}
\]  

(5)

with the transformations

\[
z = \frac{y - \xi}{\omega}, \quad \tau(z) = \sqrt{\frac{v+1}{v+z^2}}
\]  

(6)

A general skew-t probability density function of \(Y_{\xi, \omega, a, v}\) is then given by

\[
f_{ST}(y; \xi, \omega, a, v) = \frac{2}{\omega} \cdot t(z) \cdot T_{v+1} (az\tau(z))
\]  

(7)
with the transformations
\[ \delta_x = \frac{\alpha}{\sqrt{1+\alpha^2}}, \quad b_\nu = \sqrt{\frac{\pi}{\Gamma(\nu/2)}} \]
one would find that the variance of the standardized skew-t random variable \( z \) is:
\[ \sigma_z^2 = \text{var}(z) = \frac{\nu}{\nu-2} - \left( b_\nu \delta_x \right)^2 \]

Note that as the slant parameter \( \alpha \) ranges over \((-\infty, +\infty)\), the parameter \( \delta_x \) ranges over the interval \((-1, 1)\), and when there is no skewness, i.e., when \( \alpha = \delta_x = 0 \), the variance of \( z \) is that of a standard symmetric-t random variable.

It will sometimes be of interest to transform a skew-t distribution parameter vector \((\xi, \omega, \alpha, \nu)\) to the usual mean, standard deviation, skewness, and excess kurtosis vector \((\mu, \sigma, \gamma, \kappa)\).

This can be done with the following transformations, provided on pages 103-104 of AZC:
\[
\begin{align*}
\mu &= \xi + \omega b_\nu \delta_x \\
\sigma^2 &= \omega^2 \sigma_z^2 \quad \text{for } \nu > 2 \\
\gamma &= \frac{b_\nu \delta_x}{\sigma_z^2} \left( \frac{3}{\nu-3} - \frac{3}{\nu-2} + 2 (b_\nu \delta_x)^2 \right) \quad \text{for } \nu > 3 \\
\kappa &= \frac{1}{\sigma_z^2} \left[ \frac{3}{\nu-2} \frac{\nu}{\nu-4} - \frac{4 (b_\nu \delta_x)^2 (\nu-3 \delta_x^2)}{\nu-3} + \frac{6 \nu (b_\nu \delta_x)^2}{\nu-2} - 3 (b_\nu \delta_x)^4 \right] - 3 \quad \text{for } \nu > 4.
\end{align*}
\]

The maximum penalized likelihood estimation method

The log-likelihood for a single observation \( y \) with density \( f(y; \theta) \) is
\[ \ell(\theta; y) = \log f(y; \theta) \]
where \( \theta \) is a parameter vector. With \( \theta = (\xi, \omega, \alpha, \nu) \) and \( z = (y - \xi)/\omega \), it follows from (7) that for a skew-t distribution the log-likelihood function for a single observation \( y \) is
\[ \ell_{ST}(\theta; y) = \text{constant} - \frac{1}{2} \log \nu + \log \Gamma\left(\frac{1}{2}(\nu+1)\right) - \log \Gamma\left(\frac{\nu}{2}\right) \]
\[ - \frac{1}{2} (\nu+1) \log \left(1 + \frac{z^2}{\nu}\right) + \log T_{\nu+1}(\alpha z \tau) \]
(11)

For \( n \) independently and identically distributed skew-t observations \( y = (y_1, y_2, \ldots, y_n) \) the log-likelihood is
\[ \ell_n(\theta) = \ell_n(\theta; y) = \sum_{i=1}^{n} \ell_{ST}(\theta; y_i) \]
(12)

Let \( \hat{\theta}_n = (\hat{\xi}_n, \hat{\omega}_n, \hat{\delta}_n, \hat{\nu}_n) \) denote a maximum-likelihood estimate (MLE) for sample size \( n \). Azzalini and Arellano-Valle (2013) showed that with non-negligible probability the MLE \( \hat{\delta}_n \) of the skewness parameter \( \alpha \) diverges. Consequently, they proposed a maximum penalized likelihood estimator (MPLE) to solve this problem, based on a penalized likelihood of the form
\[ \ell_n(\theta) = \ell_n(\theta) - Q(\theta) \]
(13)
where \( Q(\theta) = Q(\alpha, \nu) \) is a non-negative quantity that penalizes the divergence of \( \alpha \) by virtue of tending to infinity as \( \alpha \to \pm \infty \) for each fixed \( \nu \). Details may be found in Section 3 of Azzalini and Arellano-Valle (2013).

Application to stock returns data

We illustrate the use of the \texttt{sn} package function \texttt{st.mle()} to compute skew-t distribution MPLE fits to three disjoint time segments of the returns of a large-cap stock with ticker D and company name The Dominion Resources, INC\(^2\). The code shown below creates the plot of monthly returns of this stock from December 1991 to September 2015 in Figure 1.

\(^1\)The skewness and excess kurtosis are given by \( \gamma = E \left( \left( \frac{Y - \mu}{\sigma} \right)^3 \right) \) and \( \kappa = E \left( \left( \frac{Y - \mu}{\sigma} \right)^4 \right) - 3 \), respectively.

\(^2\)See https://www.sec.gov/Archives/edgar/data/715957/000119312505038784/d10k.htm#toc55889_2 for more information.
The entire time history of the D returns is longer than one would normally use for portfolio construction, and we note that this time interval contains several major market crises, e.g., the Russian default in 1998, the dot-com bubble collapse in 2000, and the financial markets crises of 2007-2008. Thus we will fit skew-t distributions, and also symmetric-t distributions, to the stock with ticker D for the sub-intervals 1991-1996, 1998-2009, and 2010-2015. The first interval, consisting of 61 months, contains no market crises, and the same is true of the third interval that consists of 72 months. The second interval, consisting of 156 months, contains all three crises. For convenience, we refer to the stock D returns on the first, second and third time intervals as D1, D2, and D3. Based on this fact that D2 contains three market crises, along with the visual character of the D2 returns, we expected that we would obtain a skew-t fit with negative slant parameter and rather small degrees of freedom for D2. But for D1 and D3 we were not sure what to expect.

The code below extracts the three segments of returns D1, D2 and D3, then uses st.mple() with the optional argument symmetr = T to obtain symmetric-t MLE values $\hat{\theta}_{Di, \text{sym}}$, $i = 1, 2, 3$, and finally uses st.mple() without that optional argument to compute skew-t MPLE estimates $\hat{\theta}_{Di}, i = 1, 2, 3$.

```r
# Define returns in each time period
Dreturns <- as.numeric(Dreturns[,2])
returns1 <- Dreturns[1:61]
returns2 <- Dreturns[62:217]
returns3 <- Dreturns[218:289]

# Symmetric-t fit
tFitD1 <- st.mple(y = returns1, symmetr = TRUE)$dp
tFitD2 <- st.mple(y = returns2, symmetr = TRUE)$dp
tFitD3 <- st.mple(y = returns3, symmetr = TRUE)$dp
tFit <- rbind(tFitD1, tFitD2, tFitD3)

# Skew-t fit
stFitD1 <- st.mple(y = returns1, penalty = "Qpenalty")$dp
stFitD2 <- st.mple(y = returns2, penalty = "Qpenalty")$dp
stFitD3 <- st.mple(y = returns3, penalty = "Qpenalty")$dp
stFit <- rbind(stFitD1, stFitD2, stFitD3)
```

The results are shown in Table 1.
We also have the scalar score functions with degrees-of-freedom estimates of 4.21 and 4.42 respectively, indicating very normally distributed D1 returns, but negatively skewed by about the same amount as the D2 returns. Fat-tailed, but a little less so than in the second period, and negatively skewed, but less so than for D1, as indicated by skew-t slant parameter estimate of −3.73. In the second period 1998-2009, that contains three financial crises, both \( \hat{\theta}_{D2, \text{sym}} \) and \( \hat{\theta}_{D2} \) imply that the D2 returns are quite fat-tailed with degrees-of-freedom estimates of 4.21 and 4.42 respectively, \( \hat{\theta}_{D2} \) indicates that the returns are negatively skewed, but less so than for D1, as indicated by skew-t slant parameter estimate of −1.06. Finally, for the third interval both \( \hat{\theta}_{D3, \text{sym}} \) and \( \hat{\theta}_{D3} \) imply that the D3 returns distribution is fat-tailed, but a little less so than in the second period, and \( \hat{\theta}_{D3} \) indicate that the D3 returns distribution is negatively skewed by about the same amount as the D2 returns.

### Skew-t information matrix

There are two equivalent forms of an information matrix, one of which involves the products that are derivatives with respect to each parameter of the log-likelihood for a single observation, and the other of which involves the partial derivatives of the score functions (second partial derivatives of the log-likelihood for a single observation) with respect to each parameter. In our case here, where \( \theta = (\xi, \omega, \alpha, v) \) and the skew-t log-likelihood for a single observation is

\[
\ell_{ST}(\theta; y) = \ell_{ST}(\xi, \omega, \alpha, v; z), \quad z = (y - \xi) / \omega
\]

We also have the scalar score functions

\[
\begin{align*}
S_\xi(y) &= \frac{\partial \ell_{ST}(\xi, \omega, \alpha, v; z)}{\partial \xi} \\
S_\omega(y) &= \frac{\partial \ell_{ST}(\xi, \omega, \alpha, v; z)}{\partial \omega} \\
S_\alpha(y) &= \frac{\partial \ell_{ST}(\xi, \omega, \alpha, v; z)}{\partial \alpha} \\
S_v(y) &= \frac{\partial \ell_{ST}(\xi, \omega, \alpha, v; z)}{\partial v}
\end{align*}
\]

and the score vector is the gradient of the log-likelihood:

\[
S(y; \xi, \omega, \alpha, v) = \nabla \ell_{ST}(\theta; y) = \left[ S_\xi(y) S_\omega(y) S_\alpha(y) S_v(y) \right]^T
\]

The two equivalent forms of the information matrix are:

\[
I(\theta) = \begin{bmatrix}
E[S_\xi(y)S_\xi(y)] & E[S_\xi(y)S_\omega(y)] & E[S_\xi(y)S_\alpha(y)] & E[S_\xi(y)S_v(y)] \\
E[S_\omega(y)S_\xi(y)] & E[S_\omega(y)S_\omega(y)] & E[S_\omega(y)S_\alpha(y)] & E[S_\omega(y)S_v(y)] \\
E[S_\alpha(y)S_\xi(y)] & E[S_\alpha(y)S_\omega(y)] & E[S_\alpha(y)S_\alpha(y)] & E[S_\alpha(y)S_v(y)] \\
E[S_v(y)S_\xi(y)] & E[S_v(y)S_\omega(y)] & E[S_v(y)S_\alpha(y)] & E[S_v(y)S_v(y)]
\end{bmatrix}
\]

\[
= -\begin{bmatrix}
E[S_{\xi\xi}(y)] & E[S_{\xi\omega}(y)] & E[S_{\xi\alpha}(y)] & E[S_{\xi v}(y)] \\
E[S_{\omega\xi}(y)] & E[S_{\omega\omega}(y)] & E[S_{\omega\alpha}(y)] & E[S_{\omega v}(y)] \\
E[S_{\alpha\xi}(y)] & E[S_{\alpha\omega}(y)] & E[S_{\alpha\alpha}(y)] & E[S_{\alpha v}(y)] \\
E[S_{v\xi}(y)] & E[S_{v\omega}(y)] & E[S_{v\alpha}(y)] & E[S_{v v}(y)]
\end{bmatrix}
\]

The importance of the information matrix \( I(\theta) \) is that \( \mathbf{V}(\theta) = I(\theta)^{-1} \) is the asymptotic covariance.

---

3See for example Casella and Berger (2001).

---

**Table 1: Symmetric-t and skew-t parameter estimates for D1, D2 and D3.**

<table>
<thead>
<tr>
<th></th>
<th>Symmetric-t</th>
<th>Skew-t</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>location</td>
<td>scale</td>
</tr>
<tr>
<td>D1 returns</td>
<td>0.006</td>
<td>0.035</td>
</tr>
<tr>
<td>D2 returns</td>
<td>0.013</td>
<td>0.044</td>
</tr>
<tr>
<td>D3 returns</td>
<td>0.014</td>
<td>0.031</td>
</tr>
</tbody>
</table>
matrix of a consistent and asymptotically normal MPLE $\hat{\theta}_n = (\hat{\xi}_n, \hat{\omega}_n, \hat{\alpha}_n, \hat{\nu}_n)$. Consequently $V(\hat{\theta}_n)$ can be used to compute approximate standard errors in the usual way, i.e., by extracting the diagonal elements of $V(\hat{\theta}_n)$, taking the square root of the result and then dividing by $\sqrt{n}$. In the remainder of this section, we focus on a numerical integration method of evaluating $I(\theta)$, verifying the accuracy of the method, and analyzing the character of the resulting $I(\theta)$ for the MPLE parameters for the stock returns D1, D2, and D3.

**Numerical evaluation of the skew-t information matrix**

The skew-t information matrix in (17)-(18) does not admit of closed-form analytic expressions for its elements, and one must resort to numerical integration. It turns out that information matrix form (17) is considerably simpler than that of (18) for purposes of numerical integration, and so we focus on the former. For this purpose we need the following analytic expressions for the score functions (15) derived by DiCiccio and Monti (2011). Let

$$w(z) = \frac{t_{v+1}(az \cdot \tau(z))}{T_{v+1}(az \cdot \tau(z))} \quad \psi(x) = \frac{\partial}{\partial x} \log(\Gamma(x))$$

where $\psi(x)$ is the digamma function. Then the score functions are

$$S_\xi(y) = \frac{z \cdot \tau(z)^2}{\omega} - \frac{av \cdot \tau(z)}{\omega(v + z^2)} \cdot w(z)$$

$$S_\omega(y) = -\frac{1}{\omega} + \frac{z^2 \cdot \tau(z)^2}{\omega} - \frac{az \cdot \tau(z)}{\omega(v + z^2)} \cdot w(z)$$

$$S_\alpha(y) = z \cdot w(z) \cdot \tau(z)$$

$$S_\nu(y) = \varphi(\nu) - \log \left( 1 + \frac{z^2}{\nu} \right) + \frac{z^2 \cdot \tau(z)^2}{\nu} + \frac{az(2z - 1)}{(v + z^2)^2 \tau(z)} \cdot w(z) + \frac{\xi(z)}{T_{v+1}(az \cdot \tau(z))}$$

where

$$\varphi(\nu) = \frac{1}{2} \left\{ \varphi \left( \frac{\nu}{2} + 1 \right) - \varphi \left( \frac{\nu}{2} \right) \right\} - \frac{2v + 1}{v(v + 1)}$$

$$\xi(z) = \int_{-\infty}^{\nu \tau(z)} \left( \frac{(v + 1)u^2}{(v + 1)(v + 1 + u^2)} - \log \left( 1 + \frac{u^2}{v + 1} \right) \right) t(u; v + 1) \, du$$

We have developed the R package `skewtInfo` at https://github.com/chindhanai/skewtInfo for numerically evaluating Equation (17). The main function in this package is `stInfoMat()`, which has an argument type that controls the type of information matrix to be computed, with the choice type = “expected” resulting in computing an expected information matrix, and the choice type = “observed” resulting in computing an observed information matrix. We note that the computation of an observed information matrix for skew-t distributions is already available using the functions st.infoObsv() in the `sn` package. But for the sake of stand-alone completeness of the `skewtInfo` package we included the capability to compute an observed information matrix.

The `stInfoMat()` function, with about 400 lines of R code, uses numerical quadrature as implemented in the function `integrate()` from the R base package, with the choice of tolerance of $10^{-9}$. The `integrate()` function maps the infinite range of parameters onto finite sub-intervals, and then uses globally adaptive interval subdivision in conjunction with Wynn’s Epsilon algorithm extrapolation, with the basic step being Gauss-Kronrod quadrature. See Piessens et al. (1983) and the documentation of the function `integrate()` for more information.

Note that the expectations in the bottom row of the information matrix involve a double integration due to the presence of $\xi(z)$ in the last term of $S_\xi(y)$, and hence the presence of $\xi(z)$ in the pairwise products of (22) with (19)-(21). For example, as a part of computing the expected value of the score functions product $S_\alpha \cdot S_\nu$, we need to evaluate

$$E \left( \frac{z^2 \xi(z)}{T_{v+1}(az \tau(z))} \right) = \int \left( \frac{z^2 \xi(z)}{T_{v+1}(az \tau(z))} \right) f_{ST}(y) \, dy$$

$$= \int_{R} \left( \int_{-\infty}^{\nu \tau(z)} \left( \frac{(v + 1)u^2}{(v + 1)(v + 1 + u^2)} - \log \left( 1 + \frac{u^2}{v + 1} \right) \right) t(u; v + 1) \, du \right) \frac{z^2 \xi(z)}{T_{v+1}(az \tau(z))} \, dy$$

To compute such an integral in R, we do the following

1. Define the innermost integrand of (24) as a function of $u$ and $v$

\[\text{We suppress the variable $z$ dependency for $w$, $\tau$ and $\xi$ for a more compact expression of the integrals.}\]
2. Integrate the inner integrand with respect to \( u \) and define this inner integral as a function of \( z \).
3. Define the outer integrand to be the product of integral from Step 2 and the function
\[
\frac{z \cdot w(z) \cdot \tau(z) \cdot f_{ST}(y)}{T_{z+1}(az \cdot \tau(z))}
\]
4. Integrate the vectorized integrand in Step 3 with respect to \( y \) by using \( \text{sapply()} \) on the \( y \) range over the real line.

Usage example

The function \( \text{stInfoMat()} \) has three arguments:

1. \( y \), a vector of skew-t random variables used to compute an observed information matrix. Its default is \( \text{NULL} \).
2. \( dp \), the skew-t direct parameter used to compute an expected information matrix, or an observed information matrix.
3. \( \text{type} = \text{c("expected","observed")}, \) the type of output information matrix, with default "expected".

The object returned by a call to \( \text{stInfoMat()} \) is a list having four objects:

1. \( dp \), the skew-t direct parameter used to compute an expected information matrix or an observed information matrix.
2. \( \text{stInfoMat} \), an expected information matrix when \( \text{type} = \text{"expected"}, \) and an observed information matrix when \( \text{type} = \text{"observed"}. \)
3. \( \text{type} \), the type of output information matrix.
4. \( \text{SEMat} \), the asymptotic standard errors of the skew-t parameters when \( \text{type} = \text{"expected"}, \) and the element-wise standard error of the observed information matrix in the case of \( \text{type} = \text{"observed"}. \)

Here the asymptotic standard errors of the skew-t MPLEs are the square roots of the diagonal elements of the inverse of the expected information matrix, and the element-wise standard errors of the observed information matrix are the standard errors of the element-wise sample means defined in equation (27) in the next subsection.

The following code illustrates the use of \( \text{stInfoMat()} \) to compute the expected information matrix \( I(\theta_{D2}) \) for the case of the skew-t parameter values \( \theta_{D2} = (0.0461, 0.0536, -1.06, 4.42) \), previously obtained as MPLE for D2 returns.

\[
\expInfo_{D2} \leftarrow \text{stInfoMat}(dp = \text{stFitD2}, \text{type} = \text{"expected"})$\text{stInfoMat}$
\]

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>392.994</td>
<td>-133.197</td>
<td>6.697</td>
<td>0.111</td>
</tr>
<tr>
<td>-133.197</td>
<td>459.792</td>
<td>2.931</td>
<td>-1.005</td>
</tr>
<tr>
<td>6.697</td>
<td>2.931</td>
<td>0.205</td>
<td>-0.007</td>
</tr>
<tr>
<td>0.111</td>
<td>-1.005</td>
<td>-0.007</td>
<td>0.005</td>
</tr>
</tbody>
</table>

Expected information matrix accuracy verification

Recall the expression for a skew-t log-likelihood \( \ell_n(\theta) \) in (12) for a sample of size \( n \), where \( \ell_{ST}(\theta; y_i) \) is the log-likelihood for a single observation, and the gradient of the log-likelihood is given by (16). It follows that the Hessian of \( \ell_n(\theta) \) is

\[
\nabla^2 \ell_n(\theta; y) = \begin{bmatrix}
\sum_{i=1}^{n} S_{\xi\xi}(y_i) & \sum_{i=1}^{n} S_{\xi\alpha}(y_i) & \sum_{i=1}^{n} S_{\xi\nu}(y_i) \\
\sum_{i=1}^{n} S_{\xi\alpha}(y_i) & \sum_{i=1}^{n} S_{\alpha\alpha}(y_i) & \sum_{i=1}^{n} S_{\alpha\nu}(y_i) \\
\sum_{i=1}^{n} S_{\xi\nu}(y_i) & \sum_{i=1}^{n} S_{\nu\alpha}(y_i) & \sum_{i=1}^{n} S_{\nu\nu}(y_i)
\end{bmatrix}
\]

(25)

where the expressions for the second partial derivatives above were derived by DiCiccio and Monti (2011). The observed information matrix \( I_{\text{obs},n}(\theta) \) for a sample of size \( n \) is:

\[
I_{\text{obs},n}(\theta) = -\nabla^2 \ell_n(\theta; y)
\]

(26)
It is a general result that the observed information matrix converges in probability to the expected information matrix:

\[ -\frac{1}{n} \nabla^2 \ell_n(\theta; y) \to^P I(\theta) \] (27)

We shall confirm the accuracy of our numerical evaluation of \( I(\theta) \) by evaluating the observed information matrix on the left-hand side of the above convergence result for very large \( n \).

We could use either \texttt{stInfoMat()} with type = “observed” or the \texttt{sn} package function \texttt{st.infoUv()} to compute the observed information matrix. These two functions give virtually identical results, and since our \texttt{skewTInfo} package contains \texttt{stInfoMat()}, we use that function to compute both observed and expected information matrices for the case of the skew-t parameter vector \( \theta_{D2} = (0.0461, 0.0536, -1.06, 4.42) \) with \( n = 100,000 \), and compare the results. Table 2 shows the following results: the absolute differences (AD) and relative differences (RD) between the expected and observed information matrix, along with the standard errors (SE) of the observed information matrix estimates, and the t-statistics (AD/SE) for testing whether or not the observed information matrix values are equal to those of the expected information matrix.

<table>
<thead>
<tr>
<th>Diagonal</th>
<th>Expected</th>
<th>Observed</th>
<th>AD</th>
<th>RD</th>
<th>SE</th>
<th>tstat.AD</th>
</tr>
</thead>
<tbody>
<tr>
<td>I_{11}</td>
<td>392.994</td>
<td>393.048</td>
<td>0.054</td>
<td>0.000</td>
<td>0.825</td>
<td>0.065</td>
</tr>
<tr>
<td>I_{22}</td>
<td>459.792</td>
<td>459.621</td>
<td>-0.170</td>
<td>0.000</td>
<td>2.249</td>
<td>-0.076</td>
</tr>
<tr>
<td>I_{33}</td>
<td>0.205</td>
<td>0.206</td>
<td>0.000</td>
<td>0.002</td>
<td>0.000</td>
<td>0.812</td>
</tr>
<tr>
<td>I_{44}</td>
<td>0.005</td>
<td>0.005</td>
<td>0.000</td>
<td>0.002</td>
<td>0.000</td>
<td>0.334</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Off.diagonal</th>
<th>Expected</th>
<th>Observed</th>
<th>AD</th>
<th>RD</th>
<th>SE</th>
<th>tstat.AD</th>
</tr>
</thead>
<tbody>
<tr>
<td>I_{12}</td>
<td>-133.197</td>
<td>-136.135</td>
<td>-2.938</td>
<td>0.022</td>
<td>1.582</td>
<td>-1.857</td>
</tr>
<tr>
<td>I_{13}</td>
<td>6.697</td>
<td>6.643</td>
<td>-0.054</td>
<td>-0.008</td>
<td>0.030</td>
<td>-1.807</td>
</tr>
<tr>
<td>I_{14}</td>
<td>2.931</td>
<td>2.904</td>
<td>-0.027</td>
<td>-0.009</td>
<td>0.020</td>
<td>-1.386</td>
</tr>
<tr>
<td>I_{23}</td>
<td>0.111</td>
<td>0.111</td>
<td>0.000</td>
<td>0.000</td>
<td>0.003</td>
<td>-0.017</td>
</tr>
<tr>
<td>I_{24}</td>
<td>-1.005</td>
<td>-1.004</td>
<td>-0.002</td>
<td>0.002</td>
<td>0.008</td>
<td>0.213</td>
</tr>
<tr>
<td>I_{34}</td>
<td>-0.007</td>
<td>-0.007</td>
<td>0.000</td>
<td>-0.007</td>
<td>0.000</td>
<td>0.485</td>
</tr>
</tbody>
</table>

Table 2: Absolute differences (AD), relative differences (RD), standard errors (SE), and t-statistics (tstat.AD) of diagonal elements (Top) and off-diagonal elements (Bottom) of the expected information matrix and the observed information matrix based on \( n = 100,000 \) for the skew-t parameter \( \theta_{D2} \).

The accuracy of the results in this table indicate that the observed information matrix with a very large sample size \( n \) is sufficiently close to the expected information matrix for us to conclude that our expected information matrix computation is sufficiently accurate.

**Information matrices condition numbers and parameters correlations**

One obtains an MPLE asymptotic covariance matrix by computing the inverse of an expected information matrix, and the stability of such inversion depends on the condition number of the latter matrix.\(^5\) In order to get a feeling for the extent of ill-conditioning of expected information matrices for the three skew-t MPLEs of the previous section, we computed the eigenvalues and condition numbers for the corresponding expected information matrices, and show the results in Table 3.

We see that the expected information matrix for \( \theta_{D1} \) is very badly ill-conditioned, and the expected information matrices of \( \theta_{D2} \) and \( \theta_{D3} \) are also quite ill-conditioned, but less so than the information matrix for \( \theta_{D1} \). Nonetheless all three of these information matrices are positive definite and their inverses exist.

\(^5\)The condition number of a positive definite real matrix \( A \) is given by

\[ \kappa(A) = \frac{\lambda_{\text{max}}}{\lambda_{\text{min}}} = \frac{\lambda^{-1}_{\text{max}}}{\lambda^{-1}_{\text{min}}} = \kappa(A^{-1}) \]

where \( \lambda_{\text{min}} \) and \( \lambda_{\text{max}} \) are the smallest eigenvalue and the largest eigenvalue of \( A \).
Table 3: Eigenvalues and condition numbers corresponding to the information matrices in the three time periods.

Asymptotic correlations of the four parameter MPLEs

The correlation matrix obtained from such an MPLE asymptotic covariance matrix indicates how correlated the skew-t MPLEs will be in large samples. In order to see what these correlations look like, we display below the MPLE asymptotic correlation matrices for the returns D1, D2 and D3.

The six pairwise asymptotic correlations are rather similar across the three skew-t parameter values as follows. The correlations between location and scale parameters tend to be quite positive (ranging from 0.734 to 0.885), the correlations between location are slant parameters are strongly negative (ranging from -0.954 to -0.802), and the correlations between location and DoF tend to be small and positive (ranging from 0.394 to 0.443). The scale and slant parameters are substantially negatively correlated (ranging from -0.874 to -0.706), and the scale and DoF parameters are rather positively correlated (ranging from 0.641 to 0.769). Finally, the slant and DoF parameters are somewhat negatively correlated (ranging from -0.411 to -0.359).

Standard errors of skew-t parameter MPLEs

Since our expected information matrix computation is highly accurate, we will use the inverse of the expected information matrix to compute standard errors in the usual way, i.e., by taking the square root of the variance estimates obtained as the diagonal elements of the estimated asymptotic covariance matrix (the inverse of the estimated expected information matrix) divided by n:

\[
SE_\theta = (SE_\xi, SE_\omega, SE_\alpha, SE_\nu) = \sqrt{\text{diag}(I^{-1}(\hat{\theta}_n))}
\]

We evaluate the performance of these standard errors in the following two ways: (a) we compare them with the ‘true’ standard errors obtained via Monte Carlo, and (b) we compare them with the standard errors based on an observed information matrix computed in the sn package. These results will reveal whether or not there are any meaningful differences in the accuracy of the standard errors obtained using the expected information matrix versus using the observed information matrix.
Before we make such comparisons, the next subsection first discusses a basic problem that occurs in carrying out Monte Carlo studies with many replicates of `st.mple()` fits, and how to deal with the problem.

### A Monte Carlo skew-t fitting difficulty and a solution

When carrying out our Monte Carlo studies involving generating a large number of random sample replicates from a skew-t distribution, and fitting a skew-t distribution with `st.mple()` to each replicate, we would often run into a problem that `st.mple()` sometimes results in a huge value for the degrees of freedom estimate and simultaneously an essentially zero estimate for the slant parameter. This problem is associated with a well-known singularity problem with a skewed normal distribution when the slant (skewness) parameter is zero, and a near singularity when the slant parameter is close to zero. The problem occurs when a randomly generated replicate of a skew-t random variable is close to normality, which happens the more so for small sample sizes.

We illustrate the behavior using $M = 1,000$ replicates of samples of size $n = 50, 100, 200, 400$ from a skew-t distribution with parameter value $\theta_{D2} = (0.0461, 0.0536, -1.06, 4.42)$ that we obtained as an `st.mple()` estimate in Table 1. The code below implements this for the case $n = 50$, with the replicates stored in the matrix `retReps`, and the fits stored in the matrix `skewtFits`.

```r
set.seed(123)
# Skew-t parameters corresponding to D2 returns
dp <- stFitD2
M <- 1000
n <- 50
retReps <- matrix(NA, ncol = n, nrow = M)
skewtFits <- matrix(NA, ncol = 4, nrow = M)
for (m in 1:M) {
  retReps[m, ] <- rst(n = n, dp = dp)
}
for (m in 1:M) {
  skewtFits[m, ] <- st.mple(y = retReps[m,], penalty = "Qpenalty")$dp
}
```

Now for each sample size 50, 100, 200, and 400, we identify a sub-matrix `skewtFitsHugeDof` of `skewtFits` for which the degrees of freedom estimate is larger than each of thresholds 20, 40, 100, 500 and 1,000, and print the number of rows of `skewtFitsHugeDof` for which this occurs for each of those thresholds, along with the parameter estimates of the first such row. The code below accomplishes this for the threshold value 1,000.

```r
skewtFitsHugeDof <- skewtFits[(skewtFits[, 4] > 1000), ]
nrow(skewtFitsHugeDof)
## [1] 123
```

Table 4 displays the number and percent of replicates for which the `st.mple()` fit for the parameters choice $\theta_{D2} = (0.046, 0.054, -1.064, 4.416)$, has degrees of freedom larger than the threshold of 20, 40, 100, 500 and 1,000, for each of the sample sizes 50, 100, 200, 400. The percentages of replicates whose DoF estimates are larger than the above thresholds indicate that the fits are quite unreliable for $n = 50$ and $n = 100$, but are much more reliable for sample size 200, where the percent of huge DoF estimates is less than 1% for all thresholds greater than 40, and at the sample size 400 it is only 0.3% for a threshold of 20, and 0.1% for larger thresholds.

In order to carry out our Monte Carlo studies with 1,000 replicates, whose results we discuss in the remainder of this subsection, we need to obtain that many replicates after deleting any replicates for which the DoF estimate is sufficiently large. For purposes of our study, we will use a threshold of 20, noting that any replicate with a DoF estimate larger than 20 is rather close to normality, and if

---

6We note that following "How to sample from the SN and related distributions when we want to fix skewness and other cumulants", we always use the DP parameterization when generating random samples from a skew-t distribution.

7On page 63 of Azzalini and Capitanio (2014) the authors suggest that for fitting their skew distributions the usual assumption that "sample sizes less than 30" should be considered "small sample", should be replaced with "sample sizes less than 50". But they also say that setting such a threshold is difficult because it depends on the slant parameter $\alpha$. For the case of the $\alpha$ values for the three skew-t parameter vectors we focus on here, it seems like the threshold may be more like 200.
significant skewness is indicated by the slant parameter estimate then one should fit a skew-normal
distribution. Thus we carry out our Monte Carlo study to obtain 1,000 clean replicates, with no DoF
estimate larger than 20, by first generating 1,300 replicates to clean, which will leave more than 1,000
replicates remaining for all cases considered, and take the first 1,000 of those remaining replicates for
Monte Carlo calculations.

Our main goal here is to do a simple Monte Carlo comparison of the MPLE values with true
parameter values, where we take the true parameter values to be the three sets of values $\theta_{Di}$, $i = 1, 2, 3$
in Table 1 for the skew-t distribution. So we compute the sample mean values $MC_{Di,n}$ of the MPLEs
across the 1,000 replicates, along with their Monte Carlo standard errors, and their percent relative
biases:

$$RB_{Di,n} = 100 \times \frac{MC_{Di,n} - \theta_{Di}}{\theta_{Di}}, \ i = 1, 2, 3.$$ 

However, before we show the results of deleting MPLEs whose DoF estimate is larger than 20,
we first show the results obtained without doing such deleting. These results, shown in Table 8 for
sample sizes $n = 100, 200, 400$, with standard errors of the MC values in parentheses, reveal that for all
three parameter sets considered, the average values of the DoF estimates are hugely positively biased.
This is because a fraction of the st.mp1e() estimates of the DoF are huge outliers that very adversely
distort the DoF Monte Carlo average value.\footnote{We deleted the standard error estimates for the DoF average values for display reasons, and that they are pretty irrelevant.}

Table 12 then shows the results of deleting any MPLEs whose DoF estimate is greater than 20. In
Table 9a where the true DoF parameter has a value close to 32 and a rejection threshold of 20 is used,
it is not surprising that the DoF values are very negatively biased. However in Tables 10a and 11a,
where the true DoF values are about 4.4 and 6.6 respectively, the DoF values are positively biased, and
the relative bias is considerably smaller than in Table 9a where the true DoF parameter value is close
to 32.

\begin{table}[h]
\begin{center}
\begin{tabular}{|c|c|c|c|c|}
\hline
 & 50 & 100 & 200 & 400 \\
\hline
Fit DoF > 20 & 170 (17\%) & 80 (8\%) & 16 (1.6\%) & 3 (0.3\%) \\
Fit DoF > 40 & 148 (14.8\%) & 58 (5.8\%) & 10 (1\%) & 1 (0.1\%) \\
Fit DoF > 100 & 136 (13.6\%) & 45 (4.5\%) & 6 (0.6\%) & 1 (0.1\%) \\
Fit DoF > 500 & 126 (12.6\%) & 39 (3.9\%) & 5 (0.5\%) & 1 (0.1\%) \\
Fit DoF > 1000 & 123 (12.3\%) & 36 (3.6\%) & 5 (0.5\%) & 1 (0.1\%) \\
\hline
\end{tabular}
\end{center}
\caption{Numbers and percentages of replicates for $\theta_{D2}$ whose MPLE fit has DoF greater than 20, 40, 100, 500, 1000.}
\end{table}
<table>
<thead>
<tr>
<th>Theta</th>
<th>MC (4e-04)</th>
<th>RB%</th>
<th>MC (3e-04)</th>
<th>RB%</th>
<th>MC (2e-04)</th>
<th>RB%</th>
<th>MC (1e-04)</th>
<th>RB%</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.046</td>
<td>0.04 (5e-04)</td>
<td>-13.2</td>
<td>0.043 (4e-04)</td>
<td>-6.7</td>
<td>0.045 (3e-04)</td>
<td>-2.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.054</td>
<td>0.053 (3e-04)</td>
<td>-1.1</td>
<td>0.053 (2e-04)</td>
<td>-1.1</td>
<td>0.054 (2e-04)</td>
<td>0.8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-1.064</td>
<td>-0.974 (0.0227)</td>
<td>-8.5</td>
<td>-1.015 (0.0158)</td>
<td>-4.6</td>
<td>-1.066 (0.0115)</td>
<td>0.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.416</td>
<td>1.77e+05</td>
<td>4.0e+06</td>
<td>4.27e+03</td>
<td>1.0e+05</td>
<td>4.82e+00</td>
<td>9.0e+00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 8: Means MC and percent relative biases RB% of the Monte Carlo skew-t MLEs corresponding to (a) $\theta_{D_1}$, (b) $\theta_{D_2}$, (c) $\theta_{D_3}$ with $M = 1000$. The four rows of values in each table are for the location, scale, slant and DoF parameters, respectively, and MC standard errors are shown in parentheses.
Further comparison of the Monte Carlo average values and relative biases of the location, scale and slant parameter estimates in Tables 8 and 12 reveal that for the stock returns data under consideration: (a) These parameter estimates are not much affected by whether or not MPLEs are rejected because their DoF estimate is larger than 20; (b) These parameter estimates all have relative bias percentages that decrease with increasing sample size, with small single digit relative bias percentages at sample size 400 for stocks D2 and D3. Emphasize that this is a very narrow conclusion based on the returns subsets of a particular stock, and a much more extensive study is needed of large cross-sections of stock to explore this issue thoroughly.

### Standard errors of skew-t MPLEs

A reason for studying the accuracy of the standard errors obtained from our skew-t expected information matrices is that in general one does not know whether or not the use of an expected information matrix yields standard errors that are more accurate, less accurate, or as accurate as those obtained using an observed information matrix. The statistical literature concerning this question includes,

---

**Table 12:** Means MC and percent relative biases RB% of the Monte Carlo skew-t MPLEs corresponding to (a) $\theta_{D_1}$, (b) $\theta_{D_2}$, (c) $\theta_{D_3}$ with a DoF threshold of 20 for eliminating an estimate with $M = 1000$. The four rows of values in each table are for the location, scale, slant and DoF parameters, respectively, and MC standard errors are shown in parentheses.
among other, an early paper by Efron and Hinkley (1978) that argues in favor of the observed information, and the more recent papers by Cao and Spall (2010) and Cao (2013) arguing that the use of the expected information matrix is preferred.

For our Monte Carlo study of the relative performance of standard errors obtained using our expected information matrix versus an observed information matrix, we focus on the two skew-t parameter vectors $\theta_{D2,n}$ and $\theta_{D3,n}$ of the "Maximum Penalized Likelihood Estimation Method" section, and skip $\theta_{D1,n}$ since its DoF parameter value calls for using a skew-normal or a normal distribution model. Noting that the Monte Carlo standard error of a standard error estimate under normality, relative to the standard deviation of the observations, is approximately $1/\sqrt{2\cdot(M-1)}$ where $M$ is the number of Monte Carlo replicates, we use $M = 4000$ to obtain a standard error of about 1%. The detailed steps of our Monte Carlo are as follows.

For $i = 2, 3$ and for each of the sample sizes $n = 50, 100, 200, 400$ and $800$, we do the following:

1. Generate a first random sample of size $n$ of skew-t random random variables with parameter $\theta_{Di,n}$.
2. Use the random sample to fit a skew-t distribution with the $\text{sn}$ package function $\text{st.mle()}$, thereby obtaining the set of skew-t parameter estimates $\hat{\xi}_{n,1}, \hat{\omega}_{n,1}, \hat{\alpha}_{n,1}, \hat{\nu}_{n,1}$.
3. Redo Step 1 and 2 for 5000 times so that we have $(\hat{\xi}_{n,m}, \hat{\omega}_{n,m}, \hat{\alpha}_{n,m}, \hat{\nu}_{n,m}), m = 1, 2, \ldots, 5000$.
4. Remove the parameter estimate vectors for which the DoF estimate $\hat{\nu}_{n,m}$ is larger than 20, and take the first 4000 of the resulting skew-t parameter estimate vectors.
5. Compute the Monte Carlo standard error $SE_{MC}$ of each of the four parameters as their sample standard deviations for the 4000 replicates.

Table 13 shows the $SE_{MC}$ results in the columns labeled MC for each of $\theta_{D2,n}$ and $\theta_{D3,n}$. The columns labeled OBS contain the standard errors $SE_{obs}$ obtained using the observed information matrix as computed by $\text{st.infoUv()}$ in $\text{sn}$, and the columns labeled EXP contain the standard errors $SE_{exp}$ as computed with the function $\text{stInfomat()}$ in our new package $\text{skewtInfo}$.

For the sake of reproducibility, we use $\text{set.seed(123)}$ for the simulation results in this section.
Table 13: Standard errors of skew-t MLEs for $\theta_{D2}$ and $\theta_{D3}$ with $M = 4000$.

The results in Table 13 are displayed graphically in the two Lattice plots of Figure 2. Each panel of the plots contains the three standard errors $SE_{MC}$, $SE_{obs}$, $SE_{exp}$ for one of the parameters, versus sample size, corresponding to columns in Table 13.

<table>
<thead>
<tr>
<th></th>
<th>D2</th>
<th>D3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MC</td>
<td>OBS</td>
</tr>
<tr>
<td>n = 50</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.0201</td>
<td>0.0214</td>
</tr>
<tr>
<td></td>
<td>0.0118</td>
<td>0.0175</td>
</tr>
<tr>
<td></td>
<td>1.7353</td>
<td>0.9236</td>
</tr>
<tr>
<td></td>
<td>3.3251</td>
<td>3.1274</td>
</tr>
<tr>
<td>n = 100</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.0157</td>
<td>0.0151</td>
</tr>
<tr>
<td></td>
<td>0.0096</td>
<td>0.0124</td>
</tr>
<tr>
<td></td>
<td>0.6744</td>
<td>0.6531</td>
</tr>
<tr>
<td></td>
<td>2.9386</td>
<td>2.2114</td>
</tr>
<tr>
<td>n = 200</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.0112</td>
<td>0.0107</td>
</tr>
<tr>
<td></td>
<td>0.0075</td>
<td>0.0087</td>
</tr>
<tr>
<td></td>
<td>0.4794</td>
<td>0.4618</td>
</tr>
<tr>
<td></td>
<td>2.1900</td>
<td>1.5637</td>
</tr>
<tr>
<td>n = 400</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.0079</td>
<td>0.0076</td>
</tr>
<tr>
<td></td>
<td>0.0059</td>
<td>0.0062</td>
</tr>
<tr>
<td></td>
<td>0.3379</td>
<td>0.3265</td>
</tr>
<tr>
<td></td>
<td>1.4120</td>
<td>1.1057</td>
</tr>
<tr>
<td>n = 800</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.0056</td>
<td>0.0054</td>
</tr>
<tr>
<td></td>
<td>0.0042</td>
<td>0.0044</td>
</tr>
<tr>
<td></td>
<td>0.2405</td>
<td>0.2309</td>
</tr>
<tr>
<td></td>
<td>0.8146</td>
<td>0.7818</td>
</tr>
</tbody>
</table>
The figure results indicate that there is often little difference between the standard errors computed with the expected information matrix versus those of the observed information matrix, and when there are significant differences, such as for the scale and DoF estimates for D3 at small sample sizes, the expected information matrix typically results in somewhat more accurate standard errors than the observed information matrix. While sample size 100 seems to be adequate for obtaining accurate standard errors for the location and slant parameter MPLEs, it appears that one needs sample size at least 200 to obtain accurate standard errors for scale parameter MPLEs. Furthermore, while the expected information matrix standard error of the DoF skew-t MPLE for D3 is quite accurate for sample sizes 100 and larger, this is not the case for D2, which indicates that quite large sample sizes are likely needed in order to obtain accurate standard errors for DoF skew-t MPLEs.

Concluding comments

Since there is no analytic formula for the Azzalini skew-t expected information matrix, we developed a numerical integration method of evaluation and implemented it in the function `stInfoMat()` in
We made an initial exploration of the properties of the resulting expected information matrix for three skew-t parameter vectors obtained using the `sn` package function `st.mple()` for three sets of monthly returns D1, D2, D3 of a single stock with ticker D, on three disjoint time intervals from December 1991 to September 2015. We confirmed the accuracy of our expected information matrix evaluation method for one of these skew-t parameter vectors by using the well-known fact that an observed information matrix converges in probability to the negative of the expected information matrix, and showing that for $n = 100,000$ the observed information matrix values differ by a negligible amount from those of our expected information matrix. We examined the condition number of the expected information matrices for those three skew-t parameter vectors and found that although the matrices are positive definite, they are quite poorly conditioned to various extents. The asymptotic correlation patterns of the three skew-t parameter MLEs was examined, and we found that they are fairly consistent in sign and magnitude across those three parameter vectors, as described at the end of the "Information Matrices Condition Numbers and Parameters Correlation" subsection.

For the two of the three skew-t MLE parameter vectors with single digit DoF values, we carried out a Monte Carlo study of the standard errors obtained with the expected information matrix, as compared with the observed information matrix, and comparing both with Monte Carlo reference standard errors. The results indicate that accurate standard errors for location and slant parameters can be obtained using both expected and observed information matrices for sample sizes of at least 100, but not for smaller sample sizes. For the scale and DoF parameters, it appears that sample sizes as large as 200 or 400 may be needed to obtain accurate standard errors, and in some cases the expected information matrix gives better results.

The Reviewer’s suggestion to consider the methods for model reduction presented by DiCiccio and Monti (2018) is quite interesting. However, we feel that it is rather outside the focus and scope of the current paper, and could be the basis of a separate study, that would also include the hypothesis that the distribution is a symmetric-t distribution. In fact, it would be great to see this capability in the `sn` package at some point in the future.
Bibliography


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 (Vadrenko, 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 (Isagris 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 (Feliciäsmo et al., 2014) conducts statistical analysis on 3d vectors. It includes estimation of parameters and some spherical statistical tests. 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 astronomy 1. 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 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

1see the list in https://asaip.psu.edu/forums/software-forum/459833927
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 packages\(^2\) 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 SMICA\(^4\). These CMB maps are provided at 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.


\(^3\)hosted at the link http://irsa.ipac.caltech.edu

\(^4\)https://wiki.cosmos.esa.int/planckpla2015/index.php/Astrophysical_component_separation
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 $K_{\text{cmb}}$ 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 "HPDataFrames" is a type of "data.frame" designed to carry data located on the unit sphere. Unlike "CMBDataFrame", "HPDataFrames" 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.

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 Nside = 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 Nside = 1024, so it contains \(N_{pix} = 12 \times 1024^2 = 12582912\) pixels, each having its own intensity I, polarisation Q, U, temperature mask value \(T_{mask} \in \{0, 1\}\) and polarisation mask value \(P_{mask} \in \{0, 1\}\).

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/
After downloading the map with `downloadCMBMap` and applying the function `CMBDataFrame`, we obtain an object of class "CMBDataFrame".

```r
data <- library(rcosmo)
data$filename <- "CMB_map_smica1024.fits"
data$downloadCMBMap(foreground = "smica", nside = 1024, filename)
data$sky <- CMBDataFrame(filename)
data$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.

```r
data$set.seed(0); s <- 2e6;
data$cmb_sample <- CMBDataFrame(filename, sample.size = s, include.m = T, include.p = T)
data$cmb_sample
```

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 `CMBDataFrame`. `rcosmo` still substantially outperforms the last version of `FITSio`. In the following example we used the CMB map with ‘SMICA’ foreground separation and \( N_{side} = 2048 \) (having approximately 50 million intensity samples) to test it on a modern laptop.

```r
filename1 <- "CMB_map_smica2048.fits"
data$downloadCMBMap(foreground = "smica", nside = 2048, filename1)
data$system.time(sky <- CMBDataFrame(filename1))
data$system.time(fits <- FITSio::readFITS(filename1))
data$1.36 0.29 1.73
```

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, `mmap` is a highly optimised interface to ‘POSIX mmap’ and Windows ‘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 \texttt{rcosmo}, integration of \texttt{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.

```r
> v <- c(1,2,4,7,11)
> sky <- CMBDataFrame(filename, spix = v, include.p = T, include.m = T, coords = "spherical")
> sky
```

A \texttt{CMBDataFrame} # A tibble: 5 x 7

<table>
<thead>
<tr>
<th>theta</th>
<th>phi</th>
<th>I</th>
<th>Q</th>
<th>U</th>
<th>TMASK</th>
<th>PMASK</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;dbl&gt;</td>
<td>&lt;dbl&gt;</td>
<td>&lt;dbl&gt;</td>
<td>&lt;dbl&gt;</td>
<td>&lt;dbl&gt;</td>
<td>&lt;int&gt;</td>
<td>&lt;int&gt;</td>
</tr>
<tr>
<td>1.57</td>
<td>0.785</td>
<td>-0.0000920</td>
<td>6.47e-8</td>
<td>-0.000000657</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1.57</td>
<td>0.786</td>
<td>-0.0000804</td>
<td>-9.19e-9</td>
<td>-0.000000694</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1.57</td>
<td>0.785</td>
<td>-0.0000771</td>
<td>5.45e-9</td>
<td>-0.000000718</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1.57</td>
<td>0.786</td>
<td>-0.0000663</td>
<td>-5.81e-8</td>
<td>-0.000000751</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1.57</td>
<td>0.783</td>
<td>-0.0000836</td>
<td>7.60e-8</td>
<td>-0.000000697</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

```r
> 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 \texttt{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 \texttt{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 \texttt{rcosmo}, we can first generate a \texttt{CMBDataFrame} at some low resolution (e.g, \texttt{Nside = 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 (Cabaretta and Roukema, 2007).

```r
> 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 = "Var", ylab = "Var", zlab = "Var")
> 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 \texttt{CMBDataFrame} and display the outputs in Figure ??).

```r
> 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)
```
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 $N_{\text{side}}$ parameter, and satisfies $N_{\text{pix}} = 12N_{\text{side}}^2$, where we use $N_{\text{pix}}$ to denote the total number of pixels.

At a given $N_{\text{side}}$, 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 2. The ring scheme is demonstrated in Figure 3.

---

**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).

**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.
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.

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.

```r
> 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 \(k = 1,2,\ldots,5\), using the internal function ancestor.

```r
> 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.

```r
p <- 1; j1 <- 1; j2 <- 5
P <- pixelWindow(j1 = j1, j2 = j2, pix.j1 = p)
displayPixels(spix = P, j = j2, plot.j = j2)
```

The group of rcosmo 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.

```r
> library(microbenchmark)
> geoDistSearch <- function(target, nside, spix) {
  xyz <- pix2coords(nside = nside, spix = spix)
  dists <- geoDist(xyz, target)
  ```
Figure 4: All pixel centers (at resolution 5), within pixel 1 (at resolution 1)

```r
return(xyz[which.min(dists),])
```

```r
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]
```

> summary(mb)$median[1]/summary(mb)$median[2]

```r
[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 `Nside = 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 \(12 + 4 \log_2(N_{side})\) pixels rather than \(12N_{side}^2\).

When \(N_{side} = 2048\), only \(12 + 4 \log_2(2048) = 56\) pixels must be checked from over 50 million.

### Subsetting and combining spherical regions

The `rcosmo` 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 `CMBWindow` objects that correspond to the Dragon and Scorpion constellations. Files of constellation boundaries\(^8\) 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.

```r
> download.file("https://www.iau.org/static/public/constellations/txt/dra.txt",
  "bound1.txt")
> x1 <- readLines("bound1.txt")
```

\(^8\)available at [https://www.iau.org/public/themes/constellations/](https://www.iau.org/public/themes/constellations/)
> 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.

![The Dragon](image1)
![The Scorpion](image2)

**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))

![CMB intensity data extracted from an CMBDataFrame object by the window function.](image3)

**Figure 6:** CMB intensity data extracted from an CMBDataFrame object by the window function.

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.
> 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 `CMBWindow` 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.

```r
> 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.

```r
> 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.

![Triangulation of a spherical polygon.](image)

**Figure 7**: Triangulation of a spherical polygon.

```r
> 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,
• 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 CMBDataFrame.

```R
> 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.

```R
> summary(sky.annulus)

Número de 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

-----------------------------------------------
Intensidade quartis
Min. 1st Qu. Median  Mean 3rd Qu. Max.

-----------------------------------------------

The function entropyCMB returns an estimated entropy for specified column intensities and CMBWindow.

```R
> entropyCMB(cmbdf = df, win = d.exterior, intensities = "I")
[1] 2.13523
```
The first Minkowski functional $f_{mf}$ returns an area of the spherical region, where the intensities are above of the specified threshold level $\alpha$.

```r
> fmf(cmbdf = sky.annulus, alpha = 0, intensities = "I")
[1] 1.054269
```

$f_{Ren}$ computes values of the sample fractal scaling exponent on the grid of $N$ uniformly spaced points in the interval $[q_{\text{min}}, q_{\text{max}}]$. 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 straight line for the data in $\text{CMBWindow} \ sky.\text{annulus}$. Thus, there is not enough evidence for substantial multifractality of the random field based on the data in this $\text{CMBWindow}$. More details can be found in the paper (Leonenko et al., 2020).

```r
> 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].](image)

The function `plotAngDis` helps to visualise the marginal distributions of temperature intensities versus $\theta$ and $\phi$ angles. It produces scatterplots and barplots of the corresponding means computed over bins, see Figure 9.

```r
> df1 <- sampleCMB(df, sample.size = 100000)
> cmbdf.win <- window(df1, new.window = polygon2)
> plotAngDis(cmbdf.win, intensities = "I")
```

![Figure 9: Distributions of temperature versus $\theta$ and $\phi$ angles for the Scorpion constellation region.](image)
Multivariate statistics for data from different CMBWindows.

There are several rcosmo functions for comparison of data from two or more CMBWindows. For example, the function `qqplotWin` is a modification of the standard `qqplot` to produce 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.

```r
> qqplotWin(df1, polygon1, polygon2)
```

![Figure 10: QQ plot of observations in the Dragon vs Scorpion constellation regions.](image)

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 show 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.

```r
> lw <- list(polygon1, polygon2)
> qstat(df1, lw)
[1] 0.01089514
```

Investigating spatial dependencies

This section presents some of rcosmo tools for the analysis of spatial dependencies in spherical data.

As the geodesic and Euclidean distances are different, covariance functions on \(\mathbb{R}^3\) cannot 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). rcosmo 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 separation 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 \(\sigma^2\) and the range \(\psi\) 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 \(h = \pi/4\).

```r
> covmodelCMB(pi/4, cov.pars = c(1, pi), kappa = 4, cov.model = "askey")
[1] 0.3164062
```

The command `plotcovmodelCMB` is designed to produce quick plots of theoretical covariance functions. The result for the Askey covariance function is shown in Figure 11.

```r
> plotcovmodelCMB("askey", phi = pi/4, to = pi/2, kappa = 4)
```

If a random field 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 \(\pi\) (by default equals \(\pi\)).
Figure 11: Plot of the Askey covariance function for the parameters $\sigma^2 = 1$, $\psi = \pi/4$, and $\kappa = 4$.

```r
> df1 <- sampleCMB(df, sample.size = 100000)
> Cov <- covCMB(df1, max.dist = 0.03, num.bins = 10)
> Cov$v
[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

```r
> plot(Cov)
```

Figure 12: Plot of the empirical covariance function for $\text{max.dist} = 0.03$.

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.

```r
> 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)
```

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 Archive\(^9\). 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.

\(^9\)hosted at the link [https://pla.esac.esa.int/pla/](https://pla.esac.esa.int/pla/)
Figure 13: Plots of the empirical and fitted variograms.

```r
> 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 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 preprocessed 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 Database\textsuperscript{10}. First, cities latitudes and longitudes in degrees are converted to spherical coordinates ($\theta, \phi$) in radians. Then, we create and visualize HPDataFrame at the resolution nside = 1024, see Figure 15.

```r
> 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")
```

\textsuperscript{10}hosted at the link \url{https://simplemaps.com/data/world-cities}
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.

### 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


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
aolenko@latrobe.edu.au
Appendix: Statistical model

Consider a sphere in the three-dimensional Euclidean space $S^2 = \{x \in \mathbb{R}^3 : ||x|| = 1\} \subset \mathbb{R}^3$.

A spherical random field on a probability space $(\Omega, \mathcal{F}, P)$, 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 $\bar{T} = \{\bar{T}(x), x \in S^2\}$, is a stochastic function defined on the sphere $S^2$.

The field $\bar{T}(x)$ is called isotropic on the sphere $S^2$ if its mean $\mathbb{E} T(\theta, \varphi) = c = \text{constant}$ and the covariance function $\mathbb{E} T(\theta, \varphi)T(\theta', \varphi')$ depends only on the angular distance $\Theta = \Theta_{\Omega\bar{Q}}$ between the points $P = (\theta, \varphi)$ and $Q = (\theta', \varphi')$ on $S^2$.

A real-valued second-order mean-square continuous spherical random field $T$ can be expanded in the series
\[ T(\theta, \varphi) = \sum_{l=0}^{\infty} \sum_{m=-l}^{l} a_{lm} Y_{lm}(\theta, \varphi), \]
where $\{Y_{lm}(\theta, \varphi)\}$ represents the complex spherical harmonics and $a_{lm}$ are random variables.

If a random field is isotropic then
\[ \mathbb{E}a_{lm}a_{l'm'} = \delta_{ll'}\delta_{mm'}C_l, \quad -l \leq m \leq l, \quad -l' \leq m' \leq l', \quad l \geq 0. \]

Thus, $\mathbb{E}|a_{lm}|^2 = C_l$, $m = 0, \pm 1, \ldots, \pm l$. The series $\{C_1, C_2, \ldots, C_l, \ldots\}$ is called the angular power spectrum of the isotropic random field $T(\theta, \varphi)$.

The covariance function of the isotropic fields $T$ has the following representation
\[ \Gamma(\cos \Theta) = \mathbb{E} T(\theta, \varphi)T(\theta', \varphi') = \frac{1}{4\pi} \sum_{l=0}^{\infty} (2l + 1) C_l P_l(\cos \Theta), \]
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
\[ \gamma(h) = \Gamma(0) - \Gamma(h). \]

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 \texttt{cosmo}. 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
\[ \hat{\gamma}(h) = \frac{1}{2N_h} \sum_{(x_1, x_2) \in N_h} (T(x_1) - T(x_2))^2, \]
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 et al., 2013; Loy et al., 2017; Wickham et al., 2015; Buja et al., 2009; Loy et al., 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.

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
CONTRIBUTED RESEARCH ARTICLES

Figure 1: A flowchart of a typical analysis that uses matahari and tidycode to analyze and classify R 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.
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.

```r
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.

```r
library(matahari)
dance_start()
1 + 2
"here is some text"
sum(1:10)
dance_stop()
dance_tbl()
```

```r
dance_tbl()
#> # A tibble: 6 x 6
#> expr value path contents selection dt
#> <list> <list> <list> <list> <list> <dttm>
#> 1 <language... <S3: sessionIn... <lgl [1... <lgl [1... <lgl [1]> 2018-09-11 22:22:12
#> 6 <language... <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)`.

```r
dance_tbl()[["expr"]]
```

```r
#> [[1]]
#> sessionInfo()
#> [[2]]
```
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
```

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]>
#> 3 <language> <dbl [1]> <NULL> <chr [1]> <chr [0]> <chr [0]>
#> 4 <language> <NULL> <S3: simpleError> <NULL> <NULL> <NULL>
#> 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

```r
code_string <- '# 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]>
#> 3 <language> <dbl [1]> <NULL> <chr [1]> <chr [0]> <chr [0]>
#> 4 <language> <NULL> <S3: simpleError> <NULL> <NULL> <NULL>
#> 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.

```r
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.

```r
tidycode_example("example_analysis.R")
```

Running the function without any file specified will supply a vector of all available file names.

```r
tidycode_example()
#> [1] "example_analysis.R" "example_plot.R"
```

We can use these example files in the `read_rfiles()` function.

```r
df <- read_rfiles(tidycode_example(c("example_analysis.R", "example_plot.R"))

df
```

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 `m` and the input column is `expr`.

```r
u <- unnest_calls(df, expr)

u
```

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 library <list [1]>
#> 2 /Library/Frameworks/R.framework/Versions/3.5/R- 2 library <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.

```r
data <- 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.

```r
data <- 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.

```r
data <- 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 > data cleaning
#> 6 data cleaning
#> 7 ( data cleaning
#> 8 ^ modeling
#> 9 library setup
#> 10 library setup
#> # ... with 10 more rows
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

```r
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 it’s arguments.

```r
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.

```r
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).

```r
classification_tbl %>%
  group_by(id, classification) %>%
  summarise(n = n()) %>%
  mutate(pct = n / sum(n)) %>%
  group_by(classification) %>%
  summarise(`Average percent` = mean(pct) * 100 %>%
    arrange(`Average percent`)
```

We can also examine the most common functions in each classification.

```r
func_counts <- classification_tbl %>%
  count(func, classification, sort = TRUE) %>%
```
Table 1: Average percent of functions spent on each task.

<table>
<thead>
<tr>
<th>classification</th>
<th>Average percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>data cleaning</td>
<td>36.40</td>
</tr>
<tr>
<td>visualization</td>
<td>23.17</td>
</tr>
<tr>
<td>exploratory</td>
<td>21.32</td>
</tr>
<tr>
<td>setup</td>
<td>18.87</td>
</tr>
<tr>
<td>modeling</td>
<td>17.69</td>
</tr>
<tr>
<td>import</td>
<td>8.58</td>
</tr>
<tr>
<td>communication</td>
<td>5.14</td>
</tr>
<tr>
<td>evaluation</td>
<td>3.62</td>
</tr>
<tr>
<td>export</td>
<td>0.82</td>
</tr>
</tbody>
</table>

We could then examine a word cloud of the functions used, colored by the classification. We can do this using the wordcloud library.

```r
library(wordcloud)

classification_tbl %>%
count(func, classification) %>%
with(
  wordcloud(func, n,
    colors = brewer.pal(9, "Set1")[factor(.x$classification)],
    random.order = FALSE,
    ordered.colors = TRUE
  )
)
```

Additionally, we could examine the variability in the types of functions used between groups.
Figure 2: Functions that contribute to data cleaning, exploratory analysis, modeling and visualization classifications in p-hack-athon 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.

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.

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.
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".
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.

```r
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.

```r
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.

```r
func_counts <- pkg_data %>%
  unnest_calls(expr) %>%
  count(pkg, func, sort = TRUE)
```

```r
dplyr = 1634
datatable = 1590
```

Using this data frame, we can visualize which functions are most commonly called in each package.

```r
Using this data frame, we can visualize which functions are most commonly called in each package.
```

```r
top_funcs <- func_counts %>%
  group_by(pkg) %>%
  top_n(10) %>%
```
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.

```r
ungroup() %>%
arrange(pkg, n) %>%
multiply(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, matabari, 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.

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 Spyridon 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 *gganimate* 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 *gganimate* (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:
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:

- \(X\), the data, an \(n \times p\) numeric matrix to be projected to 2D.
- \(B = (B_1, B_2)\), 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”.
- \(e\), a 1D basis vector of length \(p\), with 1 in the \(k\)-th position and 0 elsewhere.
- \(M\) is a \(p \times 3\) matrix, defining the 3D subspace where data rotation occurs and is called the manipulation space.
- \(R\), the 3D rotation matrix, for performing unconstrained 3D rotations within the manip space, \(M\).
- \(\theta\), the angle of in-projection rotation, for example, on the reference axes; \(c_\theta, s_\theta\) are its cosine and sine.
- \(\phi\), the angle of out-of-projection rotation, into the manip space; \(c_\phi, s_\phi\) are its cosine and sine. The initial value for animation purposes is \(\phi_1\).
- \(U\), the axis of rotation for out-of-projection rotation orthogonal to \(e\).
- \(Y\), the resulting projection of the data through the manip space, \(M\), and rotation matrix, \(R\).

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 \((B)\) 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, \(Y_{[n, 2]} = X_{[n, p]} B_{[p, 2]}\). 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, $e$, 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, $M$.

$$e_k \leftarrow 1$$

$$e_{[p, 1]} = e - \langle e, B_1 \rangle B_1 - \langle e, B_2 \rangle B_2$$

$$M_{[p, 3]} = (B_1, B_2, e^*)$$

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 $\theta$, the angle describing rotation within the projection plane, and compute a sequence for $\phi$, defining movement out of the plane. This will change $\phi$ from the initial value, $\phi_1$, the angle between $e$ and its shadow in $B$, 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 $\phi_1$.

Rotations in 3D can be defined by the axes they pivot on. Rotation within the projection, $\theta$, is rotation around the $Z$ axis. Out-of-projection rotation, $\phi$, is the rotation around an axis on the $XY$ plane, $U$, orthogonal to $e$. Given these axes, the rotation matrix, $R$ 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.

\[
R_{3, 3} = I_3 + s_\theta U + (1 - c_\theta)U^2
\]

\[
= \begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1 \\
\end{bmatrix} + \begin{bmatrix}
0 & 0 & c_\phi s_\theta \\
0 & 0 & s_\phi s_\theta \\
-c_\phi s_\theta & -s_\phi s_\theta & 0 \\
\end{bmatrix} + \begin{bmatrix}
-c_\theta (1 - c_\phi) & s_\phi s_\theta & 0 \\
-c_\phi s_\theta (1 - c_\phi) & -c_\phi s_\theta & 0 \\
0 & 0 & c_\phi - 1 \\
\end{bmatrix}
\]

where

\[
U = (u_x, u_y, u_z) = (s_\theta, -c_\theta, 0)
\]

\[
= \begin{bmatrix}
0 & -u_z & u_y \\
u_z & 0 & -u_x \\
-u_y & u_x & 0 \\
\end{bmatrix} = \begin{bmatrix}
0 & 0 & -c_\theta \\
0 & 0 & -s_\theta \\
c_\theta & s_\theta & 0 \\
\end{bmatrix}
\]

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 \( \phi \) to vary between 0 and \( \pi/2 \), call the steps \( \phi_i \).
CONTRIBUTED RESEARCH ARTICLES

(1) \( \phi = 0.78 \)  
(2) \( \phi = 0 \)  
(3) \( \phi = 1.57 \)  
(4) \( \phi = 0.78 \)

Figure 3: Snapshots of a radial manual tour manipulating aede2: (1) original projection, (2) full contribution, (3) zero contribution, (4) back to original.

## 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.

\[
Y_{[n, 3]}^{(i)} = X_{[n, p]} M_{[p, 3]} R_{[3,3]}^{(i)}
\]

where \( R_{[3,3]}^{(i)} \) is the incremental rotation matrix, using \( \phi_i \). To make the data plot, use the first two columns of \( Y \). Show the projected data for each frame in sequence to form an animation.

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
### Table 1: Summary of available functions.

<table>
<thead>
<tr>
<th>Type</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>construction</td>
<td>create_manip_space</td>
<td>forms the 3D space of rotation</td>
</tr>
<tr>
<td>construction</td>
<td>rotate_manip_space</td>
<td>performs 3D rotation</td>
</tr>
<tr>
<td>construction</td>
<td>manual_tour</td>
<td>generates sequence of 2D frames</td>
</tr>
<tr>
<td>render</td>
<td>array2df</td>
<td>turn the tour path array into long form, for plotting</td>
</tr>
<tr>
<td>render</td>
<td>render</td>
<td>render long form as a ggplot2 object for animation</td>
</tr>
<tr>
<td>render</td>
<td>render_plotly</td>
<td>render the animation as a plotly object (default)</td>
</tr>
<tr>
<td>render</td>
<td>render_gganimate</td>
<td>render the animation as a gganimate object</td>
</tr>
<tr>
<td>animation</td>
<td>play_tour_path</td>
<td>composite function animating the specified tour path</td>
</tr>
<tr>
<td>animation</td>
<td>play_manual_tour</td>
<td>composite function animating the specified manual tour</td>
</tr>
<tr>
<td>specialty</td>
<td>print_manip_space</td>
<td>table of the rotated basis and manip space</td>
</tr>
<tr>
<td>specialty</td>
<td>oblique_frame</td>
<td>display the reference axes of a given basis</td>
</tr>
<tr>
<td>specialty</td>
<td>view_manip_space</td>
<td>illustrative display of any manip space</td>
</tr>
</tbody>
</table>

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](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:

```r
## Install from CRAN
install.package("spinifex")
## Load into session
library("spinifex")
```

Also see the shiny app for understanding and the vignette for basic usage:

```r
## Shiny app for visualizing basic application
run_app("intro")
## View the code vignette
vignette("spinifex_vignette")
```

The development version can be installed from GitHub:

```r
## Optionally install latest development 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.

```r
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:

```r
## 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:

```r
## 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 spinifex. 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.

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.
Figure 5: Snapshots of a radial manual tour of PC4 within the jet cluster, with color indicating experiment type: ATLASnew (green) and ATLASold (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.
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


Nicholas Spyrisou
Monash University
Faculty of Information Technology
nicholas.spyrisou@monash.edu

Dianne Cook
Monash University
Department of Econometrics and Business Statistics
dicook@monash.edu
**ari: The Automated R Instructor**
*by Sean Kross, Jeffrey T. Leek, John Muschelli*

**Abstract** We present the ar-i 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 ar-i 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 ar-i, 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 ar-i, 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 ar-i 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 ar-i, 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 ar-i package with reproducible use case examples and the video outputs with different voices in multiple languages.

The ar-i 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; ar-i 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. ar-i 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 ar-i.

We have developed a workflow with ar-i 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, ar-i 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 ar-iExtra package, which contains functions that connect ar-i 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 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
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.

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):

```r
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/SFhvM9gI0kE.

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

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:

```r
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:

```r
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:

```r
# 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 “ggplot2” 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.

```r
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/wrtqcpxn685_zk570bnx9_rr0000gr/T/ /Rtmpo6a9u/filede623636195.png... done!
Converting page 2 to /var/folders/1s/wrtqcpxn685_zk570bnx9_rr0000gr/T/ /Rtmpo6a9u/filede62326b98ef.png... done!

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

$images
[1] "/private/var/folders/1s/wrtqcpxn685_zk570bnx9_rr0000gr/T/ Rtmpo6a9u/filede623658cc5_files/slide_1.png"
[2] "/private/var/folders/1s/wrtqcpxn685_zk570bnx9_rr0000gr/T/ Rtmpo6a9u/filede623658cc5_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."

```
"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:

```r
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](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:

```r
gs_doc <- ariExtra::gs_to_ari("14gd2DioCVKRNPpFflriryGG7D3S8p9aZ")
\end{Sinput}
\begin{Soutput}
Converting page 1 to
/var/folders/zw/l4fv__6n4tnbk3xb31dnbt5m0000gn/T/\RtmpphWBAj/filebd69651ed561.png... done!
Converting page 2 to
/var/folders/zw/l4fv__6n4tnbk3xb31dnbt5m0000gn/T/\RtmpphWBAj/filebd69651b4b724.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 mscstts 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


T. J. Leeper. `aws.polly`: Client for AWS Polly, 2017. R package version 0.1.5. [p258]

T. J. Leeper. `aws.signature`: Amazon Web Services Request Signatures, 2019. R package version 0.5.1. [p260]


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 uses good functions 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 the bivariate and vine copula models (Vatter and Chavez-Demoulin, 2015). For the right-censoring situation, the CopulaSurv 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 Sundararjo 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

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

where $u$ and $v$ are two uniformly distributed margins; $\phi_{\eta}$ is the generator function, which is a continuous, strictly decreasing and convex function; $\phi_{\eta}^{-1}$ is the inverse of $\phi_{\eta}$. One generator function uniquely
determines an Archimedean copula. The copula parameter $\eta$ has a one-to-one correspondence with the popular dependence measure Kendall’s $\tau$. Another property of the copula is the tail dependence (i.e., $\tau_L$ and $\tau_U$ 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 $\eta$. 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

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

(1)

The two dependence parameters ($\alpha$ and $\kappa$) are explicitly connected to Kendall’s $\tau$ with $\tau = 1 - 2\alpha/\left(2\alpha + 1\right)$, and they account for the correlation between $u$ and $v$ at upper and lower tails. In particular, when $\kappa = 1$, Copula2 becomes the Clayton copula, and when $\kappa \to \infty$, 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.

<table>
<thead>
<tr>
<th>Family</th>
<th>Parameter Space</th>
<th>Generator $g_k(t), t \in (0, \infty)$</th>
<th>Generator Inverse $d_k^{-1}(s), s \in (0, 1)$</th>
<th>$\tau_L$</th>
<th>$\tau_U$</th>
<th>Kendall’s $\tau$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clayton</td>
<td>$\eta &gt; 0$</td>
<td>$\exp\left(-t^{1/\eta}\right)$</td>
<td>$\frac{1}{\eta} \left(1 - \exp\left(-t^{1/\eta}\right)\right)$</td>
<td>0</td>
<td>$\frac{1}{\eta}$</td>
<td>$\frac{1}{\eta}$</td>
</tr>
<tr>
<td>Gumbel</td>
<td>$\eta &gt; 0$</td>
<td>$-\eta \log(1 + e^{\eta t})$</td>
<td>$\log\left(1 + e^{\eta t}\right)$</td>
<td>0</td>
<td>$\frac{1}{\eta}$</td>
<td>$\frac{1}{\eta}$</td>
</tr>
<tr>
<td>Frank</td>
<td>$\eta &gt; 0$</td>
<td>$\left(1 - \eta e^{\eta t}\right)^{1/\eta}$</td>
<td>$\left(1 - e^{\eta t}\right)^{1/\eta}$</td>
<td>0</td>
<td>$1 - \frac{1}{\eta}$</td>
<td>$1 - \frac{1}{\eta}$</td>
</tr>
<tr>
<td>AMH</td>
<td>$\eta &lt; 0$</td>
<td>$\frac{1}{\eta} \left(\log(1 + e^{\eta t})\right)$</td>
<td>$\log(1 + e^{\eta t})$</td>
<td>0</td>
<td>$\frac{1}{\eta}$</td>
<td>$\frac{1}{\eta}$</td>
</tr>
<tr>
<td>Joe</td>
<td>$\eta &gt; 0$</td>
<td>$\frac{1}{\eta} \left(\log(1 + e^{\eta t})\right)$</td>
<td>$\left(1 - \eta e^{\eta t}\right)^{1/\eta}$</td>
<td>0</td>
<td>$\frac{1}{\eta}$</td>
<td>$\frac{1}{\eta}$</td>
</tr>
<tr>
<td>Copula2</td>
<td>$\alpha, \kappa &gt; 0$</td>
<td>$\left(1 - \frac{1}{\kappa} \log(1 + e^{\eta t})\right)$</td>
<td>$\left(1 - e^{\eta t}\right)^{1/\kappa}$</td>
<td>0</td>
<td>$\frac{1}{\kappa}$</td>
<td>$\frac{1}{\kappa}$</td>
</tr>
</tbody>
</table>

$\tau_L$ and $\tau_U$ are the lower and upper tail dependence measures.

$D_k(t)$ is the Debye function written as $D_k(t) = \frac{1}{kB_k} \int_0^t t^{k-1} \ln(t) dt$.

<table>
<thead>
<tr>
<th>Clayton ($\tau = 0.5$)</th>
<th>Gumbel ($\tau = 0.5$)</th>
<th>Frank ($\tau = 0.5$)</th>
<th>AMH ($\tau = 0.3$)</th>
<th>Joe ($\tau = 0.5$)</th>
<th>Copula2 ($\tau = 0.5$)</th>
</tr>
</thead>
</table>

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

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 semiparametric 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).

<table>
<thead>
<tr>
<th>Type</th>
<th>Models</th>
<th>Survival Distributions $S(t)$</th>
<th>Corresponding R Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parametric</td>
<td>Weibull</td>
<td>$\exp{-\left(\frac{1}{\lambda}\right)t^{\beta}}$</td>
<td>rc_par_copula, ic_par_copula</td>
</tr>
<tr>
<td></td>
<td>Compertrz</td>
<td>$\exp{-\frac{1}{\lambda}(e^{t\beta}-1)}$</td>
<td>rc_par_copula, ic_par_copula</td>
</tr>
<tr>
<td></td>
<td>Loglogistic</td>
<td>${1 + (t/\lambda)^{\beta}}^{-1}$</td>
<td>rc_spCox_copula</td>
</tr>
<tr>
<td>Semiparametric</td>
<td>Cox</td>
<td>$\exp{-\Lambda(t)e^{\beta}}$</td>
<td>rc_spCox_copula</td>
</tr>
<tr>
<td></td>
<td>Transformation</td>
<td>$\exp{-G(\Lambda(t)e^{\beta})}$</td>
<td>ic_spTran_copula</td>
</tr>
</tbody>
</table>

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 (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 $\tau$ 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 $S_i(t_j) = Pr(T_j > t_j), j = 1, 2$, 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 $i = 1 \cdots n$, 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 $T_{ij}$, $\Delta_{ij}$ 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 \( C_y \) that connects two marginal survival functions into the joint survival function: \( S(t_1, t_2) = C_y \{ S_1(t_1), S_2(t_2) \} \), \( t_1, t_2 \geq 0 \). Here, the function \( C_y \) is called a copula and its parameter \( \eta \) 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 \( C_y(u, v) \) as \( c_y(u, v) = \partial^2 C_y(u, v)/\partial u \partial v \). Let \( f(t_1, t_2) = \partial^2 S(t_1, t_2)/\partial t_1 \partial t_2 = c_y \{ S_1(t_1), S_2(t_2) \} f_1(t_1) f_2(t_2) \) denote the corresponding density function of \( S(t_1, t_2) \). Denote by \( \theta = (\beta_1^T, \beta_2^T, \eta, S_{01}, S_{02})^T \) all the unknown parameters in \( S(t_1, t_2) \), where \( \beta_j \) 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

\[
L_n(\theta | D) = \prod_{i=1}^n f(y_{1i}, y_{2i}|Z_{1i}, Z_{2i})^{\delta_{1i} \delta_{2i}} \times \left[ - \frac{\partial S(y_{1i}, y_{2i}|Z_{1i}, Z_{2i})}{\partial y_{1i}} \right]^{\delta_{1i}(1-\delta_{2i})} \times \left[ S(y_{1i}, y_{2i}|Z_{1i}, Z_{2i}) \right]^{(1-\delta_{1i})(1-\delta_{2i})}
\]

\[
= \prod_{i=1}^n \left[ - \frac{\partial C_y \{ S_1(y_{1i}|Z_{1i}), S_2(y_{2i}|Z_{2i}) \} f_1(y_{1i}) f_2(y_{2i})}{\partial y_{1i}} \right]^{\delta_{1i}(1-\delta_{2i})} \times \left[ \frac{\partial C_y \{ S_1(y_{1i}|Z_{1i}), S_2(y_{2i}|Z_{2i}) \}}{\partial y_{2i}} \right]^{\delta_{2i}(1-\delta_{1i})} \times C_y \{ S_1(y_{1i}|Z_{1i}), S_2(y_{2i}|Z_{2i}) \}^{(1-\delta_{1i})(1-\delta_{2i})},
\]

where \((\delta_{1i}, \delta_{2i}) \in \{ (0, 0), (0, 1), (1, 0), (1, 1) \}\).

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

\[
L_n(\theta | D) = \prod_{i=1}^n \left[ Pr(L_{i1} < T_{i1} \leq R_{i1}, L_{i2} < T_{i2} \leq R_{i2}|Z_{1i}, Z_{2i}) \right]
\]

\[
= \prod_{i=1}^n \left[ Pr(T_{i1} > L_{i1}, L_{i2} > L_{i2}|Z_{1i}, Z_{2i}) - Pr(T_{i1} > L_{i1}, T_{i2} > R_{i2}|Z_{1i}, Z_{2i}) 
- Pr(T_{i1} > R_{i1}, T_{i2} > L_{i2}|Z_{1i}, Z_{2i}) + Pr(T_{i1} > R_{i1}, T_{i2} > R_{i2}|Z_{1i}, Z_{2i}) \right]
\]

\[
= \prod_{i=1}^n \left[ C_y \{ S_1(L_{i1}|Z_{1i}), S_2(L_{i2}|Z_{2i}) \} - C_y \{ S_1(L_{i1}|Z_{1i}), S_2(R_{i2}|Z_{2i}) \}
- C_y \{ S_1(R_{i1}|Z_{1i}), S_2(L_{i2}|Z_{2i}) \} + C_y \{ S_1(R_{i1}|Z_{1i}), S_2(R_{i2}|Z_{2i}) \} \right].
\]

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 \( R_{1i} \) and \( R_{2i} \) are \( \infty \)) or two (if one \( R_{ij} \) is \( \infty \)) 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

\[
S_j(t_j|Z_i) = \exp \{ -(t_j/\lambda_j)^{\beta_j} e^{Z_i^T\beta_j} \}, \quad j = 1, 2,
\]
where \( \lambda_i \) and \( k_j \) are the scale and shape parameters of the baseline Weibull distribution, and \( \beta_j \) are the covariate effects. The model follows the PH assumption. In this case, the parameter set \( \theta \) becomes \( \{\beta^T, \eta, \lambda_1, k_1, \lambda_2, k_2\}^T \). 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

\[
S_j(t_j|Z_j) = \exp\{-\Lambda_j(t_j) e^{Z_j^T \beta_j}\}, \quad j = 1, 2,
\]

in which the Breslow baseline cumulative hazard function \( \Lambda_j(t) \) is given by

\[
\Lambda_j(t) = \sum_{i=1}^n \frac{I(Y_{ij} \leq t) \delta_{ij}}{\sum_{k \in R_{ij}} \exp{Z_k^T \beta_j}},
\]

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:

\[
S_j(t_j|Z_j) = \exp\{-G_j(\Lambda_j(t_j) e^{Z_j^T \beta_j})\}, \quad j = 1, 2.
\]

\( \Lambda_j(\cdot) \) is an unknown and non-decreasing function of \( t \), which is not necessarily the baseline cumulative hazards function. In CopulaCenR, we approximate \( \Lambda_j \) in a sieve space constructed by Bernstein polynomials. A Bernstein basis polynomial with degree \( m \) is expressed as:

\[
B_k(t, m, l, u) = \binom{m}{k} \frac{t - l}{u - l}^k (1 - \frac{t - l}{u - l})^{m-k}, \quad k = 0, ..., m,
\]

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):

\[
G_j(x) = \begin{cases} 
\frac{(1+x)^r-1}{r}, & 0 < r \leq 2, \\
\log(1+(r-2)x), & r > 2.
\end{cases}
\]

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 \( \exp\{-\Lambda_j(t) e^{Z_j^T \beta_j}\} \), which is essentially a PH model. Likewise, when \( G_j(x) = \log(1 + x) \) at \( r = 3 \), \( S_j(t|Z) \) becomes \( \{1 + \Lambda_j(t) e^{Z_j^T \beta_j}\}^{-1} \), 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 \( \theta \). For simplicity, we use the general notation \( \theta = (\beta_1^T, \beta_2^T, \eta, S_{01}, S_{02})^T \) 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 \( \theta_0 \):

   (a) \((\widehat{\beta}_j^{(1)}, \widehat{S}_b^{(1)}) = \arg \max_{(\beta_j, S_b)} l_m(\beta_j, S_b), \) where \( l_m \) denotes the log-likelihood for the marginal model, \( j = 1, 2 \);
We now separate the arguments \( \beta \) and \( \eta \) into two parts: \( \beta_E \) and \( \beta_{ng} \), where \( \beta_E \) is the parameter set of interest for hypothesis testing and \( \beta_{ng} \) denotes the rest of the regression coefficients. In certain cases, \( \beta_E \) can be the entire regression parameter \( \beta \). 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 \( \chi^2 \) with \( \text{dim}(\beta_E) \) degrees of freedom. In particular, the generalized score test is usually faster than the other two tests for large-scale testing 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.

### Likelihood-based tests for covariate effects

We now separate \( \beta \) into two parts: \( \beta_E \) and \( \beta_{ng} \), where \( \beta_E \) is the parameter set of interest for hypothesis testing and \( \beta_{ng} \) denotes the rest of the regression coefficients. In certain cases, \( \beta_E \) can be the entire regression parameter \( \beta \). 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 \( \chi^2 \) with \( \text{dim}(\beta_E) \) degrees of freedom. In particular, the generalized score test is usually faster than the other two tests for large-scale testing 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 \( \lambda \) (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, \( x_1 \) and \( x_2 \) 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.

```r
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)

<table>
<thead>
<tr>
<th>id</th>
<th>ind</th>
<th>var1</th>
<th>var2</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>6.130533</td>
<td>8.062168</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>6.154606</td>
<td>7.472649</td>
<td></td>
</tr>
</tbody>
</table>
```
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`).

```r
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

<table>
<thead>
<tr>
<th>estimate</th>
<th>SE</th>
<th>stat</th>
<th>pvalue</th>
</tr>
</thead>
<tbody>
<tr>
<td>lambda</td>
<td>90.6440318</td>
<td>47.2360</td>
<td>13.1887218</td>
</tr>
<tr>
<td>k</td>
<td>0.8062766</td>
<td>0.0586207</td>
<td>189.1758</td>
</tr>
<tr>
<td>treat1</td>
<td>0.5714498</td>
<td>0.1997080</td>
<td></td>
</tr>
<tr>
<td>treat2</td>
<td>0.0052997</td>
<td>0.1739106</td>
<td></td>
</tr>
<tr>
<td>eta</td>
<td>0.6205855</td>
<td>0.2610638</td>
<td></td>
</tr>
</tbody>
</table>

---
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 $-k \log(\lambda) + \text{treat1} = -4.20$ and similarly the argon treatment effect is $-k \log(\lambda) + \text{treat2} = -3.63$, 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`.

```r
AIC(clayton_wb)
1689.442
BIC(clayton_wb)
1705.858
```

After the model is fitted, Kendall’s $\tau$ can be estimated through the function `tau_copula`.

```r
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.

```r
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 0.4655284

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 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)))

head(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
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., \( r = 3 \)).

```r
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**

<table>
<thead>
<tr>
<th>estimate</th>
<th>SE</th>
<th>stat</th>
<th>pvalue</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENROLLAGE</td>
<td>0.042610</td>
<td>0.012271</td>
<td>12.057</td>
</tr>
<tr>
<td>rs2284665</td>
<td>0.397712</td>
<td>0.091180</td>
<td>19.026</td>
</tr>
<tr>
<td>SevScaleBL</td>
<td>0.930508</td>
<td>0.058714</td>
<td>251.137</td>
</tr>
<tr>
<td>alpha</td>
<td>0.074037</td>
<td>0.026808</td>
<td>18.562</td>
</tr>
<tr>
<td>kappa</td>
<td>0.974037</td>
<td>0.226081</td>
<td>18.562</td>
</tr>
</tbody>
</table>

---

**Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 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 \( 1.29 \times 10^{-5} \), 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.

```r
AIC(copula2_sp)
4226.356

BIC(copula2_sp)
4266.353
```

Also, the estimated Kendall’s \( \tau \) is 0.38, suggesting moderate dependence in AMD progression between two eyes.

```r
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")

<table>
<thead>
<tr>
<th>stat</th>
<th>pvalue</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.943163e+01</td>
<td>1.042661e-05</td>
</tr>
</tbody>
</table>

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)

<table>
<thead>
<tr>
<th>stat</th>
<th>pvalue</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.543119588</td>
<td>0.002007003</td>
</tr>
</tbody>
</table>

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))

<table>
<thead>
<tr>
<th>id</th>
<th>ind</th>
<th>SevScaleBL</th>
<th>ENROLLAGE</th>
<th>rs2284665</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
<td>60</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>60</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
<td>60</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>60</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>3</td>
<td>60</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
<td>60</td>
<td>2</td>
</tr>
</tbody>
</table>

plot(x = copula2_sp, class = "joint", newdata = newdata2)

![Figure 3: Estimated joint progression-free probability contours for subjects with different genotypes of rs2284665 (age 60 and severity score 3 in both eyes).](image)

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 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


**Tao Sun**  
School of Statistics  
Renmin University of China

The R Journal Vol. 12/1, June 2020  
ISSN 2073-4859
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.

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

\[ F(x) = \sum_{i=1}^{n} w_i F_i(x|B_i), \quad f(x) = \sum_{i=1}^{n} \frac{w_i}{F_i(B_i)} 1_{B_i}(x) f_i(x), \]

and

\[ F(x) = \sum_{i=1}^{n} w_i f_i(x), \quad f(x) = \sum_{i=1}^{n} w_i f_i(x), \]

respectively, where \( w_1, w_2, \ldots, w_n \) are positive weights that sum up to one, \( B_1, B_2, \ldots, B_n \) are Borel sets giving a disjoint partition of the support, \( 1_{B_i} \) is the indicator function of a set \( B_i \), and \( F_1, F_2, \ldots, F_n \) are the probability distributions over \( \mathbb{R} \) with \( F_i(B_i) > 0 \) for all \( i = 1, 2, \ldots, n \). 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 lognormal 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), Relns (Reynkens and Verbelen, 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 \( B_1, B_2, \ldots, B_n \). 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 \( i = 2, \ldots, n \),

\[
\lambda_i = \begin{cases} 
F_i(\beta_{i-1}) & \text{if } \beta_{i-1} \notin B_i, \\
F_i(\beta_{i-1}) & \text{otherwise},
\end{cases}
\]

where as before \( F_i(\beta_{i-1}) \) is the left-hand limit \( \mathbb{P}(X_i < x) \), and for all \( i = 1, 2, \ldots, n-1 \),

\[
\rho_i = \begin{cases} 
F_i(\beta_i) & \text{if } \beta_i \in B_i, \\
F_i(\beta_i) & \text{otherwise},
\end{cases}
\]

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 \( p_i = \sum_{j=1}^{i-1} w_j \), the composite distribution satisfies

\[
F(x) = p_{i-1} + w \frac{F_i(x) - \lambda_i}{F_i(B_i)} = p_{i-1} + w \frac{F_i(x) - \lambda_i}{\rho_i - \lambda_i}, \quad \forall x \in B_i
\]

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

\[
F(x) \geq p \iff F_i(x) \geq \lambda_i + \frac{p - p_i}{w_i} (\rho_i - \lambda_i),
\]

The R Journal Vol. 12/1, June 2020

ISSN 2073-4859
which implies
\[
F^{-1}(p) = F^{-1}_i \left( \lambda_i + \frac{p - P_i}{w_i} (\rho_i - \lambda_i) \right), \quad \text{for } p_{i-1} < p \leq p_i.
\]

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:

```r
N <- normdist(mean = 1, sd = 3)
N
```

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

```r
d(N, c(1, 2, 3))
p(N, c(1, 2, 3))
q(N, c(0.1, 0.2, 0.3))
r(N, 3)
```

Additional important functions provided by `mistr` are `plim()` and `qlim()`, which implement, respectively, the already mentioned left-hand limit of the cumulative distribution function \( F(x^-) = P(X < x) \) and its pseudoinverse

\[
Q(p+) = \inf \{ x \in \mathbb{R} : p < P(X \leq x) \}.
\]

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

\[
q = P(-X \leq x) = 1 - P(X < -x) = -Q_{-X}(1 - q+) = x.
\]

```r
B <- binomdist(size = 12, prob = 0.3)
plim(B, c(-3, 0, 3, 12))
qlim(B, plim(B, c(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.

```r
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.

```r
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.

```r
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.

```r
Binom_trafo <- -3 * log(B + 4)
q(Binom_trafo, c(0.05, 0.5, 0.95))
#> [1] -6.907755 -6.238325 -4.828314
```
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)`

![CDF and PDF plots](image1.png)

**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))`

![CDF and PDF plots](image2.png)

**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

$$X_{(np)} \sim \text{AN} \left(F^{-1}(p), \frac{p(1-p)}{n \left[f_{F}^{-1}(p)\right]^2}\right),$$

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).

For alternative bounds see Almeida et al. (2018).

```r
QQplotgg(Norm_trafo, r(Norm_trafo, 1000), conf = 0.99, ylab = NULL, xlab = NULL)
```

![Q-Q plot](image)

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.

```r
mixdist(c("norm", "unif"), list(c(2, 2), c(1, 5)), weights = c(0.5, 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:

```r
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 \( q_{\text{lim}} \) and \texttt{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

\[
\min_{i \in \{1, \ldots, n\}} Q_i(p) \leq Q(p) \leq \max_{i \in \{1, \ldots, n\}} Q_i(p),
\]

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.

```r
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")
```

![CDF plot of the -DM object with calculated quantiles annotated with white dots.](image)

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 \texttt{actuar} package (Dutang et al., 2008). This approach is implemented in the corresponding method of the \texttt{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 \texttt{mixdist}, e.g. \texttt{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 \texttt{r()} and \texttt{autoplot()} follows.

```
M_trans <- -2 * (M)^(1/3)
r(M_trans, 4)
```
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.

```r
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
```

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.

```r
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.

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.

```r
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.
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.

\[
C_3 \leftarrow \text{compdist}(M_{\text{trans}} - 3, \text{C}_{\text{trans}}, \text{weights} = c(0.5, 0.5), \text{breakpoints} = -4.5)
\]

\[
C_3_{\text{trans}} \leftarrow -2 \times C_3 + 2
\]

Thus, the object \(C_3_{\text{trans}}\) 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 \text{plim()}\ and \text{qlim()}\. The functions \text{d()}\ and \text{p()}\ can be again best represented graphically, where both distributions can easily be recognized from previous chapters.

\[
\text{autoplot}(C_3_{\text{trans}}, \text{xlim1} = c(-10, 5))
\]

\[
\text{autoplot}(C_{\text{trans}}, \text{xlim1} = c(-10, 5))
\]

**Figure 9:** CDF (left) and PDF (right) plot of the \(C_{\text{trans}}\) object using \text{autoplot()}\.

**Figure 10:** CDF (left) and PDF (right) plot of the \(C_3_{\text{trans}}\) 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 misr 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:

$$f(x) = \begin{cases} 
  w_1 \frac{f_p(x)}{1 - \Phi_{\beta_1}} & \text{if } -\infty < x < \beta_1, \\
  w_2 \frac{f_n(x)}{\Phi_{\beta_2} - \Phi_{\beta_1}} & \text{if } \beta_1 \leq x < \beta_2, \\
  w_3 \frac{f_p(x)}{1 - \Phi_{\beta_2}} & \text{if } \beta_2 \leq x < \infty,
\end{cases}$$

where $f_p(x) = f_{-p}(-x)$ and $F_p(x) = 1 - (K/x)^{a}$ 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 $\gamma$ and $a$. This means that the conditional distribution $f_p(x|K,a)/(1 - F_p(\beta_2|K,a)) = f_p(x|\beta_2,a)$ if $\beta_2 > K$. On the other hand, if $\beta_2 < K$ the distribution
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

$$
w_1 = \frac{\sum_{i=1}^n 1[x_i < \beta_1]}{n},
$$

$$
w_2 = \frac{\sum_{i=1}^n 1[\beta_1 \leq x_i < \beta_2]}{n},
$$

$$
w_3 = \frac{\sum_{i=1}^n 1[\beta_2 \leq x_i]}{n},
$$

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 \texttt{PNP_fit()} which takes the data and a named vector of starting values with names \texttt{break1}, \texttt{break2}, \texttt{mean}, and \texttt{sd} and returns a list of class \texttt{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 \texttt{mle2()} function from \texttt{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.

\[
\text{PNP_model <- PNP_fit(stocks$SAP)}
\]

\[
\text{PNP_model}
\]

\[
\text{#> Fitted composite Pareto-Normal-Pareto distribution:}
\]

\[
\text{#> Breakpoints: -0.019304 0.020518}
\]

\[
\text{#> Weights: 0.092443 0.82135 0.086207}
\]

\[
\text{#> Parameters:}
\]

\[
\text{#> scale1 shapel mean sd scale2 shape2}
\]

\[
\text{#> 0.019304 1.773598 0.000961 0.012950 0.020518 2.198590}
\]

\[
\text{#> Log-likelihood: 7400.117, Average log-likelihood: 2.7146}
\]

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 \texttt{parameters()}, weights using \texttt{weights()}, and breakpoints using \texttt{breakpoints()}. The \texttt{distribution()} function can be used to extract the distribution with fitted parameters that can be used for evaluation.

Finally, again \texttt{plot()} and \texttt{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 \texttt{plot} calls.

\[
\text{plot(PNP_model, ylab1 = "", ylab2 = "")}
\]
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)

\[ F_{\text{GPD}}(x) = \begin{cases} 
1 - \left(1 + \frac{x - \theta}{\gamma}\right)^{-1/\xi} & \text{if } \xi \neq 0, \\
1 - \exp\left(-\frac{x - \theta}{\gamma}\right) & \text{if } \xi = 0.
\end{cases} \]

This means that the PDF of this model can be written as:

\[ f(x) = \begin{cases} 
w_1 \frac{f_{\text{GPD}}(x)}{F_{\text{GPD}}(\beta_1)} & \text{if } -\infty < x < \beta_1, \\
w_2 \frac{f_{\text{N}}(x) - F_{\text{N}}(\beta_1)}{F_{\text{N}}(\beta_2) - F_{\text{N}}(\beta_1)} & \text{if } \beta_1 \leq x < \beta_2, \\
w_3 \frac{f_{\text{GPD}}(x)}{1 - F_{\text{GPD}}(\beta_2)} & \text{if } \beta_2 \leq x < \infty.
\end{cases} \]

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 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()`.

```r
GNG_model <- GNG_fit(stocks$SAP)
```

The R Journal Vol. 12/1, June 2020 ISSN 2073-4859
#> 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)

![Graph](image)

**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.

```r
risk(GNG_model, c(0.02, 0.05, 0.07, 0.1, 0.2, 0.3))
```

<table>
<thead>
<tr>
<th>level</th>
<th>VaR</th>
<th>ES</th>
<th>Exp</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.02</td>
<td>0.042341368</td>
<td>0.06220519</td>
</tr>
<tr>
<td>2</td>
<td>0.05</td>
<td>0.027889909</td>
<td>0.04520065</td>
</tr>
<tr>
<td>3</td>
<td>0.07</td>
<td>0.023062791</td>
<td>0.03952074</td>
</tr>
</tbody>
</table>
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**


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 inflexion point in the item characteristic curve) whereas non-uniform DIF also comprises a difference in item discrimination (slope at the inflexion 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.Rproject.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 Martiníková (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 Martiníková, 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 (Martiníková 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 Martiníková, 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

\[
P(Y_{ip} = 1|X_{ip}, G_p) = c_{iG_p} + (d_{iG_p} - c_{iG_p}) \frac{e^{a_{iG_p}(X_{ip} - b_{iG_p})}}{1 + e^{a_{iG_p}(X_{ip} - b_{iG_p})}},
\]

where \(X_{ip}\) 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 \(a_{iG_p} = a_i + a_{DIF} G_p\), where \(a_{DIF}\) is a difference between the focal and the reference group in discrimination (analogously for other parameters). In other words, \(a_0 = a_i\) for the reference group and \(a_1 = a_i + a_{DIF}\) 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 Martiníková (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 \(X_{ip}\) is represented by the total test
The logistic regression procedure which estimates the probability of the correct answer can be extended to the item parameters in terms of power and rejection rates (Drabinová and Martínková, 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:

$$\text{RSS}(a_{ig}, b_{ig}, c_{ig}, d_{ig}) = \sum_{p=1}^{n} \left\{ y_{ip} - c_{ig} - \frac{(d_{ig} - c_{ig})}{1 + e^{-a_{ig}(x_p - b_{ig})}} \right\}^2,$$

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 $i$

$$L(a_{ig}, b_{ig}, c_{ig}, d_{ig}) = \prod_{p=1}^{n} \left\{ c_{ig} + \frac{d_{ig} - c_{ig}}{1 + e^{-a_{ig}(x_p - b_{ig})}} \right\} y_{ip} \left\{ 1 - c_{ig} - \frac{d_{ig} - c_{ig}}{1 + e^{-a_{ig}(x_p - b_{ig})}} \right\}^{1-y_{ip}},$$

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., $Y \in \{0, 1, \ldots, K\}$, the cumulative probability is

$$P(Y_{ip} \geq k| X_p, G_p) = \frac{e^{a_{ip}(x_p - b_{kg})}}{1 + e^{a_{ip}(x_p - b_{kg})}}.$$

for $k = 1, \ldots, K$, where $P(Y_{ip} \geq 0| X_p, G_p) = 1$ and $b_{kg} = b_k + b_{iDIF} G_p$. The parameter $b_{iDIF}$ can be interpreted as the difference in difficulty of item $i$ between the reference and the focal group. The parameter $b_k$ 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 $k = 0, \ldots, K - 1$ 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

$$\log \frac{P(Y_{ip} = k| X_p, G_p)}{P(Y_{ip} = k-1| X_p, G_p)} = a_{ig}(x_p - b_{kg}),$$

where $k = 1, \ldots, K$ and $b_{kg} = b_k + b_{iDIF} G_p$. The category probability takes the following form:

$$P(Y_{ip} = k| X_p, G_p) = \frac{\sum_{j=0}^{k} e^{a_{ig}(x_p - b_{gj})}}{\sum_{j=0}^{K} e^{a_{ig}(x_p - b_{gj})}},$$

where $k = 1, \ldots, K$ and parameters for $k = 0$ are set to zero, i.e., $a_{ig}(x_p - b_{kg}) = 0$ and hence $P(Y_{ip} = 0| X_p, G_p) = \frac{1}{\sum_{j=0}^{K} e^{a_{ig}(x_p - b_{gj})}}$. 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), DIF detection can be performed with the multinomial model. Considering that $k = 0, \ldots, K$ 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
\[
P(Y_{ip} = k | X_p, G_p) = \frac{e^{a_{ikG_p}(X_p - b_{ikG_p})}}{\sum_{k=0}^{K} e^{a_{ikG_p}(X_p - b_{ikG_p})}},
\]
while for the correct answer \( k = 0 \) the parameters are set to zero, i.e., \( a_{0kG} = b_{0kG} = 0 \) and thus
\[
P(Y_{ip} = 0 | X_p, G_p) = \frac{1}{\sum_{k=0}^{K} e^{a_{ikG_p}(X_p - b_{ikG_p})}}.
\]
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).

#### Table 1: DIF and DDF detection methods available in the difNLR package.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>difNLR()</td>
<td>Performs DIF detection procedure for dichotomous data based on a nonlinear</td>
</tr>
<tr>
<td></td>
<td>regression model (generalized logistic regression) and either likelihood-ratio</td>
</tr>
<tr>
<td></td>
<td>or F test of the submodel.</td>
</tr>
<tr>
<td>difORD()</td>
<td>Performs DIF detection procedure for ordinal data based on either an adjacent</td>
</tr>
<tr>
<td></td>
<td>category logit model or a cumulative logit model and likelihood ratio test</td>
</tr>
<tr>
<td></td>
<td>of the submodel.</td>
</tr>
<tr>
<td>ddfMLR()</td>
<td>Performs DDF detection procedure for nominal data based on a multinomial</td>
</tr>
<tr>
<td></td>
<td>log-linear regression model and likelihood ratio test of the submodel.</td>
</tr>
</tbody>
</table>

#### 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

```r
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.

```r
# discrimination
a <- matrix(rep(c(1.00, 1.12, 1.45, 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.

```r
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.

```r
set.seed(42)
df <- genNLR(N = 1000, a = a, b = b, c = c, d = d)
head(df[, c(1:5, 16)])
```

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).

We are now able to perform the basic DIF detection with a 4PL model for all the items on a generated example dataset DataDIF.
### Table 2: Predefined models for the model argument in the difNLR() function.

<table>
<thead>
<tr>
<th>Model annotation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;4PL&quot;</td>
<td>4PL model</td>
</tr>
<tr>
<td>&quot;4PLcdg&quot;, &quot;4PLc&quot;</td>
<td>4PL model with an inattention parameter set equal for the two groups</td>
</tr>
<tr>
<td>&quot;4PLcgd&quot;, &quot;4PLd&quot;</td>
<td>4PL model with a guessing parameter set equal for the two groups</td>
</tr>
<tr>
<td>&quot;4PLcgdg&quot;</td>
<td>4PL model with a guessing and an inattention parameters set equal for the two groups</td>
</tr>
<tr>
<td>&quot;3PLd&quot;</td>
<td>3PL model with an inattention parameter and $c = 0$</td>
</tr>
<tr>
<td>&quot;3PLce&quot;, &quot;3PL&quot;</td>
<td>3PL model with a guessing parameter and $d = 1$</td>
</tr>
<tr>
<td>&quot;3PLdg&quot;</td>
<td>3PL model with an inattention parameter set equal for the two groups</td>
</tr>
<tr>
<td>&quot;3PLcg&quot;</td>
<td>3PL model with a guessing parameter set equal for the two groups</td>
</tr>
<tr>
<td>&quot;2PL&quot;</td>
<td>Logistic regression model, i.e. $c = 0$ and $d = 1$</td>
</tr>
<tr>
<td>&quot;1PL&quot;</td>
<td>1PL model with a discrimination parameter fixed on value of 1 for the two groups</td>
</tr>
<tr>
<td>&quot;Rasch&quot;</td>
<td>1PL model with a discrimination parameter set equal for the two groups</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Chisq-value</th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.2044</td>
<td>0.1844</td>
</tr>
<tr>
<td>0.2802</td>
<td>0.9911</td>
</tr>
<tr>
<td>2.7038</td>
<td>0.6086</td>
</tr>
<tr>
<td>5.8271</td>
<td>0.2124</td>
</tr>
<tr>
<td>48.0052</td>
<td>0.0000 ***</td>
</tr>
<tr>
<td>7.2060</td>
<td>0.1254</td>
</tr>
<tr>
<td>3.2390</td>
<td>0.5187</td>
</tr>
<tr>
<td>16.8991</td>
<td>0.0020 **</td>
</tr>
<tr>
<td>2.1595</td>
<td>0.7064</td>
</tr>
<tr>
<td>4.6866</td>
<td>0.3210</td>
</tr>
<tr>
<td>69.5328</td>
<td>0.0000 ***</td>
</tr>
<tr>
<td>8.1931</td>
<td>0.0848</td>
</tr>
<tr>
<td>2.5850</td>
<td>0.6295</td>
</tr>
<tr>
<td>2.9478</td>
<td>0.5666</td>
</tr>
<tr>
<td>20.6589</td>
<td>0.0004 ***</td>
</tr>
</tbody>
</table>

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...
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.

The \texttt{difNLR()} function provides a visual representation of the item characteristic curves using the \texttt{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 \texttt{plot()} and by specifying items to be plotted. We show here characteristic curves for DIF items only (Figure 1).

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{characteristic_curves.png}
\caption{Characteristic curves of DIF items.}
\end{figure}
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.

```r
# item-specific model
fit2 <- difNLR(DataDIF, groupDIF, focal.name = 1, model = model, type = "all")
fit2$DIFitems
[1] 5 8 11 15
# item-specific type
type <- rep("all", 15)
fit3 <- difNLR(DataDIF, groupDIF, focal.name = 1, model = model, type = type)
fit3$DIFItems
[1] 5
# item-specific constraints
constraints <- rep(NA, 15)
fit4 <- difNLR(DataDIF, groupDIF, focal.name = 1, model = model, type = constraints)
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).

```r
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
Figure 2: Information criteria for item models.

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.

\[
\text{logLik(fit3, item = 8)}
\]

'log Lik.' -312.7227 (df=7)

\[
\text{logLik(fit4, item = 8)}
\]

'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.

\[
predict(fit1, item = 5, group = c(0, 1), match = 0)
\]

<table>
<thead>
<tr>
<th>item</th>
<th>match</th>
<th>group</th>
<th>prob</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item5</td>
<td>0</td>
<td>0</td>
<td>0.7851739</td>
</tr>
<tr>
<td>Item5</td>
<td>0</td>
<td>1</td>
<td>0.5624883</td>
</tr>
</tbody>
</table>

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 ordinaly 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, ..., 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.

```r
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.

```r
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.

```r
DataORD <- genNLR(N = 1000, itemtype = "ordinal", a = a, b = b)
```

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.

```r
(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

<table>
<thead>
<tr>
<th>Item</th>
<th>group</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item1</td>
<td>Min. :0.0</td>
</tr>
<tr>
<td>Item2</td>
<td>1st Qu.:0.0</td>
</tr>
<tr>
<td>Item3</td>
<td>Median :0.5</td>
</tr>
<tr>
<td>Item4</td>
<td>Mean :0.5</td>
</tr>
<tr>
<td>Item5</td>
<td>3rd Qu.:1.0</td>
</tr>
<tr>
<td></td>
<td>Max. :1.0</td>
</tr>
</tbody>
</table>

Likelihood-ratio Chi-square statistics

<table>
<thead>
<tr>
<th>Item</th>
<th>Chisq-value</th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item1</td>
<td>7.4263</td>
<td>0.0244 *</td>
</tr>
<tr>
<td>Item2</td>
<td>13.4267</td>
<td>0.0012 **</td>
</tr>
<tr>
<td>Item3</td>
<td>0.6805</td>
<td>0.7116</td>
</tr>
<tr>
<td>Item4</td>
<td>5.6662</td>
<td>0.0588</td>
</tr>
</tbody>
</table>

The R Journal Vol. 12/1, June 2020 ISSN 2073-4859
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")

Figure 3: Cumulative probabilities and characteristic curves of item 1 with a cumulative logit model.

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

<table>
<thead>
<tr>
<th></th>
<th>Chisq-value</th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item1</td>
<td>8.9024</td>
<td>0.0117  *</td>
</tr>
<tr>
<td>Item2</td>
<td>12.9198</td>
<td>0.0016  **</td>
</tr>
<tr>
<td>Item3</td>
<td>1.0313</td>
<td>0.5971</td>
</tr>
<tr>
<td>Item4</td>
<td>4.3545</td>
<td>0.1134</td>
</tr>
<tr>
<td>Item5</td>
<td>2.3809</td>
<td>0.3041</td>
</tr>
</tbody>
</table>

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).

```r
plot(fit6, item = fit6$DIFitems)
```

**Figure 4:** Characteristic curves of DIF items with an adjacent category logit model.

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., \( b_{0Gp} + b_{1Gp} X_p \) instead of \( a_{IGp} (X_p - b_{ikGp}) \). The DIF detection is the same as with IRT parametrization, the only difference can be found in parameter estimates:

```r
fit6a <- difORD(DataORD, group = 6, focal.name = 1, model = "adjacent", parametrization = "classic")
# coefficients with IRT parametrization
round(coef(fit6)[[1]], 3)
```

<table>
<thead>
<tr>
<th>b1</th>
<th>b2</th>
<th>b3</th>
<th>b4</th>
<th>a</th>
<th>bDIF1</th>
<th>bDIF2</th>
<th>bDIF3</th>
<th>bDIF4</th>
<th>aDIF</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.013</td>
<td>0.603</td>
<td>1.500</td>
<td>2.500</td>
<td>1.776</td>
<td>-0.042</td>
<td>-0.121</td>
<td>-0.240</td>
<td>-0.374</td>
<td>0.273</td>
</tr>
</tbody>
</table>

```r
# coefficients with classical intercept-slope parametrization
round(coef(fit6a)[[1]], 3)
```

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-0.023</td>
<td>-1.070</td>
<td>-2.664</td>
</tr>
</tbody>
</table>

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:

```r
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).

```r
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.

```r
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.

```r
DataDDF <- genNLR(N = 1000, itemtype = "nominal", a = a, b = b)
head(DataDDF)
```

<table>
<thead>
<tr>
<th>Item1</th>
<th>Item2</th>
<th>Item3</th>
<th>Item4</th>
<th>Item5</th>
<th>Item6</th>
<th>Item7</th>
<th>Item8</th>
<th>Item9</th>
<th>Item10</th>
<th>group</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>B</td>
<td>C</td>
<td>A</td>
<td>C</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>C</td>
<td>B</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>C</td>
<td>C</td>
<td>B</td>
<td>C</td>
<td>B</td>
<td>C</td>
<td>B</td>
<td>D</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A</td>
<td>C</td>
<td>A</td>
<td>C</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>B</td>
<td>C</td>
<td>B</td>
<td>C</td>
<td>A</td>
<td>C</td>
<td>A</td>
<td>A</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

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).

```r
(fit7 <- ddfMLR(DataDDF, group = "group", focal.name = 1, key = rep("A", 10)))
```

Likelihood-ratio chi-square statistics

```r
Detection of both types of Differential Distractor Functioning using multinomial log-linear regression model
```

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
```

---

The R Journal Vol. 12/1, June 2020 ISSN 2073-4859
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"))
```

*Figure 5: Characteristic curves of DDF items with a multinomial model.*

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 and 6 as functioning differently. Now we can set all items excluding these two as the anchors.

```r
fit8b <- difNLR(DataDIF[, 1:6], groupDIF, focal.name = 1, match = "score",
                model = "4PL", type = "all", anchor = 1:4)
```

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.

```r
fit9 <- difNLR(DataDIF[, 1:6], groupDIF, focal.name = 1, match = "score",
               model = "4PL", type = "all", purify = TRUE)
```

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.

```r
fit10 <- difNLR(DataDIF[, 1:6], groupDIF, focal.name = 1, match = "score",
                  model = "4PL", type = "all", p.adjust.method = "holm")
```

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.

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 \(a_{\text{Dif}}, b_{\text{Dif}}, c_{\text{Dif}},\) and \(d_{\text{Dif}}\). 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.

```r
# sampled data
set.seed(42)
sam <- sample(1:1000, 420)
# using re-calculation of starting values
fit12a <- difNLR(DataDIF[, ], groupDIF[,], focal.name = 1, model = "4PL",
                 type = "all")
Starting values were calculated based on bootstrapped samples.

# turn off option of re-calculating starting values
fit12b <- difNLR(DataDIF[, ], groupDIF[,], 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.

```r
fit13 <- difNLR(DataDIF[, ], groupDIF[,], 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.

```r
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.

```r
fit14$diffPur
   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
```

The R Journal Vol. 12/1, June 2020 ISSN 2073-4859
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.

```r
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
     3453   AS     0       0       0       0       0       0       1
     3456   AS     0       0       0       0       1       1       0
     3458   AS     0       0       0       0       0       0       1
     3464   AS     0       0       1       0       0       0       1
     3474   AS     0       1       0       0       1       1       0
     3491   AS     0       0       1       0       0       1       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.

```r
fitex1 <- difNLR(Data = LtL6_gr6, group = "track", focal.name = "AS", model = "3PL", match = zscore6)
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 = zscore6)
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)

Figure 6: Characteristic curves of item 6H with DIF caused by differences in guessing between tracks in Grade 6.

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 dfdMLR() 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)

head(LtL6_gr9)

track Item6A_9 Item6B_9 Item6C_9 Item6D_9 Item6E_9 Item6F_9 Item6G_9 Item6H_9
3453 AS 1 0 0 1 0 1 0 1
3456 AS 1 1 1 1 1 1 1 1
3458 AS 1 1 0 1 1 1 0 0
3464 AS 0 0 0 0 1 0 0 1
3474 AS 1 1 0 1 0 1 0 0
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"
)

<table>
<thead>
<tr>
<th>item match group</th>
<th>prob lwr.conf upr.conf</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item6A_9 -1 0 0 0.6785773 0.6188773 0.7382773</td>
<td></td>
</tr>
<tr>
<td>Item6A_9 0 0 0.7773050 0.7269066 0.8277034</td>
<td></td>
</tr>
<tr>
<td>Item6A_9 1 0 0.8781427 0.8114574 0.9448280</td>
<td></td>
</tr>
<tr>
<td>Item6A_9 -1 1 0.7802955 0.7187001 0.8418909</td>
<td></td>
</tr>
<tr>
<td>Item6A_9 0 1 0.8431045 0.7869886 0.8992204</td>
<td></td>
</tr>
<tr>
<td>Item6A_9 1 1 0.9290780 0.8549489 1.0032071</td>
<td></td>
</tr>
</tbody>
</table>

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.

```r
# 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])
```

```r
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.

```r
fitex4 <- ddfMLR(Data = LtL6_change, group = "track", focal.name = "AS", key = rep("11", 8), match = zscore6)
```

```r
[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).

```r
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.

```r
# ordinal data for change between Grade 6 and 9
LtL6_change_ord <- data.frame(track = LtL6_change$track,
  ...)
```
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.

```r
fitex5 <- difORD(Data = LtL6_change_ord, group = "track", focal.name = "AS", model = "adjacent", match = zscore6)
```

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).

```r
plot(fit5, item = fit5$DIFitems)
```

**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.

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
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 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.
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 diffMLR() 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


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
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 distancesmost 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 identification 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 \( A = \{A_1, \ldots, A_k\} \) 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 \( R : A \rightarrow \mathcal{P}_n \) that attributes a rank \( R_t \in \{1, \ldots, n\} \) to each item, according to some criterion. We here denote a generic complete ranking by \( R = (R_{1}, \ldots, R_{n}) \), where \( R_t \) is the rank assigned to item \( A_t \). Given a pair of items \( \{A_i, A_k\} \), we denote a pairwise preference between them as \( (A_i < A_k) \), 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 \( R \in \mathcal{P}_n \), 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:

\[
(A_t \prec A_{t'}) \iff R_t < R_{t'}, \quad t_1, t_2 = 1, \ldots, n, \quad t_1 \neq t_2.
\]

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 \( r \in \mathcal{P}_n \) as follows

\[
P(\mathbf{r}|\alpha, \rho) = \frac{1}{Z_n(\alpha)} \exp \left[ -\frac{\alpha}{n} d(\mathbf{r}, \rho) \right] 1_{\mathcal{P}_n}(\mathbf{r})
\]

(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 \( r \in \mathcal{P}_n \) and zero otherwise.

In the complete data case, \( N \) assessors have provided complete rankings of the \( n \) items in \( A \) according to some criterion, yielding the permutation \( R_j = (R_{1j}, \ldots, R_{nj}) \) for assessor \( j, j = 1, \ldots, N \). The likelihood of the \( N \) observed rankings \( R_1, \ldots, R_N \), assumed conditionally independent given \( \alpha \) and \( \rho \), is

\[
P(\mathbf{R}_1, \ldots, \mathbf{R}_N|\alpha, \rho) = \frac{1}{Z_n(\alpha)^N} \exp \left[ -\frac{\alpha}{n} \sum_{j=1}^{N} d(\mathbf{R}_j, \rho) \right] \prod_{j=1}^{N} 1_{\mathcal{P}_n}(\mathbf{R}_j).
\]

(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 \( \alpha \)

\[
\pi(\alpha | \lambda) = \frac{\lambda \exp(-\lambda \alpha)}{1 - \exp(-\lambda \sigma_{\text{max}})} 1_{[0,\sigma_{\text{max}}]}(\alpha),
\]

(3)

where \( \lambda \) is a rate parameter, small enough to ensure good prior dispersion, and \( \sigma_{\text{max}} \) is a cutoff, large enough to cover reasonable \( \alpha \) values. A uniform prior \( \pi(\rho) \) on \( \mathcal{P}_n \) was assumed for \( \rho \). It follows that the posterior distribution for \( \alpha \) and \( \rho \) is

\[
P(\alpha, \rho | \mathbf{R}_1, \ldots, \mathbf{R}_N) \propto \frac{1_{[0,\sigma_{\text{max}}]}(\rho)}{Z_n(\alpha)} \exp \left[-\frac{1}{n} \sum_{j=1}^{N} \left( \mathbf{R}_{\cdot j} - \mathbf{R}_{\cdot j}^\rho \right) - \lambda \alpha \right] 1_{[0,\sigma_{\text{max}}]}(\alpha).
\]

(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 \( \rho \) 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 \( A_j \subseteq A \), which were given ranks from 1 to \( n_j = |A_j| \). Now \( R_{ij} \in \{1, \ldots, n_j\} \) if \( A_i \in A_j \), while for \( A_i \notin A_j \), \( R_{ij} \) is unknown, except for the constraint \( R_{ij} > n_j \). The augmented data vectors \( \tilde{R}_{1j}, \ldots, \tilde{R}_{N} \) are introduced in the model to include the missing ranks, which are estimated as latent parameters. Let \( S_j = \{ \tilde{R}_j \in \mathcal{P}_n : R_{ij} = R_{ij} \text{ if } A_i \in A_j, j = 1, \ldots, N \} \) 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 \( \{n_j + 1, \ldots, n\} \). The goal is to sample from the posterior distribution

\[
P(\alpha, \rho | \mathbf{R}_1, \ldots, \mathbf{R}_N) = \sum_{\mathbf{R}_1 \in S_1} \cdots \sum_{\mathbf{R}_N \in S_N} P(\alpha, \rho, \mathbf{R}_1, \ldots, \mathbf{R}_N | \mathbf{R}_1, \ldots, \mathbf{R}_N).
\]

(5)

The augmentation scheme amounts to alternating between sampling \( \alpha \) and \( \rho \) given the current values of the augmented ranks using the posterior given in (4), and sampling the augmented ranks given the current values of \( \alpha \) and \( \rho \). For the latter task, once \( \alpha, \rho \) and the observed ranks \( \mathbf{R}_1, \ldots, \mathbf{R}_N \) are fixed, one can see that \( \mathbf{R}_1, \ldots, \mathbf{R}_N \) are conditionally independent, and that each \( \tilde{R}_j \) only depends on the corresponding \( R_j \). As a consequence, the update of new augmented vectors is performed independently, for each \( j = 1, \ldots, N \).

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 \( A_j \subseteq A \) consisting of \( n_j \leq n \) items. Note that the only difference from the previous formulation is that each latent rank vector \( \tilde{R}_j \) takes values in the set

\[
S_j = \left\{ \tilde{R}_j \in P_n : \left( \tilde{R}_{ij} < \tilde{R}_{ij} \right) \land \left( A_{ij}, A_{ij} \in A_j \right) \Rightarrow \tilde{R}_{ij} < \tilde{R}_{ij} \right\}.
\]

Also in this case the prior for \( \tilde{R}_j \) is assumed uniform on \( S_j \).

In the case of pairwise comparison data, let us call \( B_j \) the set of all pairwise preferences stated by assessor \( j \), and let \( A_j \) be the set of items appearing at least once in \( B_j \). Note that the items in \( A_j \) 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 \( B_j \) are transitive, i.e., mutually compatible, and define by \( tc(B_j) \) the transitive closure of \( B_j \), which contains all pairwise orderings of the elements in \( A_j \) induced by \( B_j \). The model formulation remains the same as in the case of partial rankings, with the prior for the augmented data vectors \( \tilde{R}_j, \ldots, \tilde{R}_N \) being uniform on the set \( S_j \) 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 \( x \not\prec z \). 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 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 de 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 \( \tilde{R} \). A mistake is defined as an inconsistency between one of the assessor’s pairwise comparisons and \( \tilde{R} \). Suppose each assessor \( j = 1, \ldots, N \) has assessed \( M_j \) pairwise comparisons, collected in the set \( B_j \), and assume the existence of latent ranking vectors \( \tilde{R}_j, j = 1, \ldots, N \). Differently from Section 2.4, since \( B_j \) is allowed to contain non-transitive pairwise preferences, the transitive closure of \( B_j \) is not defined, and the posterior density (5) cannot be evaluated. In this case, the posterior takes the form,

\[
P(\alpha, \rho | B_1, \ldots, B_N) = \sum_{\tilde{R}_1 \in P_n} \cdots \sum_{\tilde{R}_N \in P_n} P(\alpha, \rho, \tilde{R}_1, \ldots, \tilde{R}_N | B_1, \ldots, B_N) = \sum_{\tilde{R}_1 \in P_n} \cdots \sum_{\tilde{R}_N \in P_n} \sum_{\tilde{R}_1 \in P_n} \cdots \sum_{\tilde{R}_N \in P_n} P(\alpha, \rho | \tilde{R}_1, \ldots, \tilde{R}_N) P(\tilde{R}_1, \ldots, \tilde{R}_N | B_1, \ldots, B_N)
\]

where the term \( P(\tilde{R}_1, \ldots, \tilde{R}_N | B_1, \ldots, B_N) \) models the presence of mistakes in the data, while in the case of transitive pairwise comparisons it was implicitly assumed equal to 1 if each augmented ranking \( \tilde{R}_j \) was compatible with the partial information contained in \( B_j \), 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:

\[
P(\tilde{R}_1, \ldots, \tilde{R}_N|\theta, B_1, \ldots, B_N) \propto \theta^M (1 - \theta)^{\sum_i M_i - M}, \quad \theta \in [0, 0.5]
\]

where \( M \) counts the number of times the observed preferences contradict what is implied by the ranking \( \tilde{R}_j \), \( M_j \) is the number of pairwise comparisons reported by assessor \( j \), and the parameter \( \theta \) is the probability of making a mistake in a single pairwise preference. \( \theta \) is a priori assigned a truncated Beta distribution on the interval \([0, 0.5]\) with given hyper-parameters \( \kappa_1 \) and \( \kappa_2 \), conjugate to the Bernoulli model (7). The Logistic model is a generalization of (7) where, instead of assigning a constant value \( \theta \) 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 \( \theta \), and the augmentation scheme for \( \tilde{R}_j \), which is slightly different. We
Clusterings

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 MM.s. 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 \( z_1, \ldots, z_N \in \{1, \ldots, C\} \) assign each assessor to one of \( C \) clusters, and let the rankings within each cluster \( c \in \{1, \ldots, C\} \) be described by an MM with parameters \( \alpha_c \) and \( \rho_c \). The likelihood of the observed rankings \( R_1, \ldots, R_N \) is given by

\[
P \left( \{R_c, \alpha_c\}_{c=1,\ldots,C}, \{z_j\}_{j=1,\ldots,N} \right) = \prod_{j=1}^{N} \frac{1}{Z_\alpha \left( \alpha_{z_j} \right)} \exp \left[ -\frac{\alpha_{z_j}}{\bar{\alpha}} d \left( R_j, \rho_{z_j} \right) \right],
\]

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 posterior density is then given by

\[
P \left( \{\rho_c, \alpha_c\}_{c=1,\ldots,C}, \{z_j\}_{j=1,\ldots,N} \right) \propto \prod_{c=1}^{C} e^{-3\lambda_c \tau_c} - \frac{\lambda_c}{\bar{\lambda}} d (\{R_c\}, \{\rho_c\}).
\]

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 \( \rho \) and (ii) updating \( \alpha \) (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 \( \rho \) 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 \( \alpha \) in step (ii), a log-normal density is used as proposal, and its variance \( \sigma^2_{\alpha} \) 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 \( \{R_j\}_{j=1,\ldots,N} \). In the case of partial rankings, for updating the augmented data \( R_j, j = 1, \ldots, N \) we use a uniform proposal on the set of rankings compatible with the partial data, \( S_j \). 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 \( \tilde{R}_j \) within the constraints given by \( \tilde{S}_j \) (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 a Gibbs step for the update of \( \theta \). The MCMC algorithm for sampling from the mixture model posterior (8) (Algorithm 2 of Vitelli et al., 2018) alternates between updating \( \{\rho_c, \alpha_c\}_{c=1,\ldots,C} \) in an M-H step, and \( \{\tau_c, z\}_{c=1,j=1}^{CN} \) 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 $\alpha$ 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 $\alpha$ 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 $Z_{\text{lim}}(\cdot)$, 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 $\alpha = 0.1, 0.2, \ldots, 20$. 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 $\alpha$. The IPFP algorithm was used to estimate $\log(Z_{\text{lim}}(\alpha))$, 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 $\alpha$. As expected, the bias of the asymptotic estimate decreases when the number of items increases, as this brings it closer to the asymptotic limit.

![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.](image-url)
Sampling from the Bayesian Mallows Model

To obtain random samples from the MM with Cayley, Hamming, Kendall, or Ulam distance and fixed \( a \) and \( \rho \), we suggest using the PerMallows (Irurozki 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 \( a \) and \( \rho \) 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.

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 \( a \) and \( \rho \) using the function 1nm 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 \( a \) together with its variance. The MLE of \( \rho \), 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 \( a \) and \( \rho \), 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 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 \( a \), and in the latent ranking of the potatoes, as described by \( \rho \). 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.

```r
bmm_test <- compute_mallows(potato_visual)
assess_convergence(bmm_test)
```
By default, assess_convergence returns a trace plot for \( \alpha \), shown in Figure 2a. The algorithm seems to be mixing well after around 500 iterations. Next, we study the convergence of \( \rho \). To avoid overly complex plots, we pick potatoes 1–5 by specifying this in the items argument.

```r
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 \( \alpha \) with `plot.BayesMallows`.

```r
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 \( \alpha \) 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.

```r
compute_posterior_intervals(bmm_visual, decimals = 1L)
```

# A tibble: 1 x 6
#  parameter mean median conf_level  hpdi  central_interval
# 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 \( \rho \).

```r
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 $\alpha$ 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 $\alpha$ only every `alpha_jump`-th iteration. To update $\alpha$ every 10th update of $\rho$, we do

```r
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 $\alpha$ 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

```r
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.

```r
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.

```r
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.

```r
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

<table>
<thead>
<tr>
<th>assessor</th>
<th>bottom_item</th>
<th>top_item</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>15</td>
</tr>
</tbody>
</table>

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:

```r
# All implied orderings for assessor 1 involving item 2
filter(beach_tc, assessor == 1, bottom_item == 2 | top_item == 2)
```

<table>
<thead>
<tr>
<th>assessor</th>
<th>bottom_item</th>
<th>top_item</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2 15</td>
</tr>
</tbody>
</table>

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.

```r
bmm_test <- compute_mallows(rankings = beach_init_rank,
                              preferences = beach_tc, save_aug = TRUE)
```

Running `assess_convergence` for \( \alpha \) and \( \rho \) 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.

```r
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:

```r
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.

```r
assess_convergence(bmm_test, parameter = "Rtilde",
                    items = 13:15, assessors = 1:8)
```

The resulting plot is shown in Figure 6, indicating good mixing.

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.

```r
bmm_beaches <- compute_mallows(rankings = beach_init_rank, preferences = beach_tc,
                                nmc = 102000, save_aug = TRUE)
bmm_beaches$burnin <- 2000
```

The posterior distributions of \( \alpha \) and \( \rho \) can be studied as shown in the previous sections. Posterior intervals for the latent rankings of each beach are obtained with `compute_posterior_intervals`:
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.

```
compute_posterior_intervals(bmm_beaches, parameter = "rho")
```

```
# A tibble: 15 x 7
  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      9 95 % [   8,10] [8,10]        
8  Beach 8 rho 12     12 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     13 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 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
```
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.

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 probabilities. By default, it sets \(k = 3\), so a heatmap of the probability of being ranked top-3 is obtained with the call:

\[
\text{plot_top_k}(\text{bmm_beaches})
\]

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 dataframe 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:

\[
\text{predict_top_k}(\text{bmm_beaches}) \%>\%
\text{filter(prob > 0.9, assessor %in% 1:5)}
\]
# 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 `cl` 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.

```r
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 `a` with the command

```r
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 \( \tau_c \), \( c = 1, \ldots, C \) have converged properly, by setting `parameter = "cluster_probs"`

```r
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.
Figure 8: Trace plot of $\alpha$ and $\tau_c$ for the sushi dataset with 1, 4, 7, and 10 mixture components, respectively. Both trace plots indicate good mixing after a few thousand iterations.

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.

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 $\rho_c$ ($c = 1, \ldots, C$) are highly peaked, we save some memory by only saving every 10th value of $\rho$ by setting `rho_thinning = 10`.

```r
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 =
We can plot the posterior distributions of $\alpha$ and $\rho$ 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:

```r
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:

```r
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 `tidyr` (Wickham and Henry, 2018) to create one column for each cluster. The result is shown in Table 1.

```r
library("tidyr")
compute_consensus(bmm) %>%
  select(-cumprob) %>%
  spread(key = cluster, value = item)
```

<table>
<thead>
<tr>
<th>Cluster 1</th>
<th>Cluster 2</th>
<th>Cluster 3</th>
<th>Cluster 4</th>
<th>Cluster 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>shrimp</td>
<td>fatty tuna</td>
<td>fatty tuna</td>
<td>fatty tuna</td>
<td>fatty tuna</td>
</tr>
<tr>
<td>sea eel</td>
<td>sea urchin</td>
<td>sea eel</td>
<td>tuna</td>
<td>sea urchin</td>
</tr>
<tr>
<td>egg</td>
<td>salmon roe</td>
<td>tuna</td>
<td>shrimp</td>
<td>shrimp</td>
</tr>
<tr>
<td>squid</td>
<td>sea eel</td>
<td>shrimp</td>
<td>tuna roll</td>
<td>tuna</td>
</tr>
<tr>
<td>salmon roe</td>
<td>tuna</td>
<td>tuna roll</td>
<td>squid</td>
<td>salmon roe</td>
</tr>
<tr>
<td>fatty tuna</td>
<td>shrimp</td>
<td>squid</td>
<td>salmon roe</td>
<td>squid</td>
</tr>
<tr>
<td>tuna</td>
<td>tuna roll</td>
<td>egg</td>
<td>tuna roll</td>
<td>sea urchin</td>
</tr>
<tr>
<td>tuna roll</td>
<td>squid</td>
<td>cucumber roll</td>
<td>cucumber roll</td>
<td>sea eel</td>
</tr>
<tr>
<td>cucumber roll</td>
<td>egg</td>
<td>salmon roe</td>
<td>sea eel</td>
<td>egg</td>
</tr>
<tr>
<td>sea urchin</td>
<td>cucumber roll</td>
<td>sea urchin</td>
<td>sea urchin</td>
<td>cucumber roll</td>
</tr>
</tbody>
</table>

**Table 1:** CP consensus for each of the clusters found for sushi data.
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 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 $\rho$; 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**


Øystein Sørensen  
Center for Lifespan Changes in Brain and Cognition  
Department of Psychology  
University of Oslo  
ORCID: 0000-0003-0724-3542  
ystein.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  
quinghualmath.uio.no

Valeria Vitelli  
Oslo Centre for Biostatistics and Epidemiology  
Department of Biostatistics  
University of Oslo  
ORCID: 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 \textit{model-agnostic approaches} are generally used to measure each predictor’s importance. Enter \texttt{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 \textit{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 \textit{interpretable machine learning} (IML).\footnote{Although “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.”} For an in-depth discussion on IML, see Molnar (2019b). In this paper, we focus on \textit{global methods} for quantifying the importance\footnote{In this context “importance” can be defined in a number of different ways. In general, we can describe it as the \textit{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).} 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 \textit{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.\footnote{We 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.}

VI scores and VIPs can be constructed for general ML models using a number of available packages. The \texttt{iml} package (Molnar, 2019a) provides the \texttt{FeatureImp()} function which computes feature importance for general prediction models using the permutation approach (discussed later). It is written in \texttt{R6} (Chang, 2019) and allows the user to specify a generic loss function or select one from a pre-defined list (e.g., \texttt{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 \texttt{iml :: FeatureImp()} function can also be run in parallel using any parallel backend supported by the \texttt{foreach} package (Revolution Analytics and Weston).

The \texttt{ingredients} package (Biecek et al., 2019a) also provides permutation-based VI scores through the \texttt{feature_importance()} function. (Note that this function recently replaced the now deprecated \texttt{DALEX} function \texttt{variable_importance()} (Biecek, 2019).) Similar to \texttt{iml :: FeatureImp()}, this function allows the user to specify a loss function and how the importance scores are computed (e.g., using the \texttt{loss} argument for mean squared error) and takes as input a fitted model object. The VI scores can be used to identify which features are driving the model’s predictions and to understand the relative importance of each feature. The VIPs are then used to visualize the importance of each feature in relation to the predicted outcome. They are particularly useful in identifying features that have a “meaningful” impact on the predicted outcome.

The vip \texttt{R} package is designed to make it easy to compute and visualize variable importance scores and plots for a variety of ML models. It provides a number of functions for calculating VIs and plotting them, including \texttt{vip::feature_importance()} and \texttt{vip::plot.vip()}, which can be used to generate both global and local variable importance plots. The package also includes functions for generating \textit{sparklines}, which are small line charts that can be used to visualize the importance of individual features in relation to the predicted outcome. These features make it easy to quickly understand the relative importance of each feature in the model.

The \texttt{vip} package is designed to be flexible and easy to use, making it accessible to a wide range of users. It can be used with a variety of ML models, including \texttt{randomForest} and \texttt{gbm}, and it can be used to calculate VIs for both regression and classification problems. It also includes functions for generating \textit{variable importance plots} (VIPs), which can be used to visualize the relative importance of each feature in the model.

To illustrate the use of the \texttt{vip} package, we will use it to calculate the variable importance scores and generate variable importance plots for a \texttt{randomForest} model fitted to the \texttt{iris} dataset.

```r
library(vip)
library(randomForest)

# Fit a random forest model to the iris dataset
set.seed(123)
iris_rf <- randomForest(Species ~ ., data = iris, importance = TRUE)

# Calculate variable importance scores
vip_scores <- feature_importance(iris_rf)

# Generate variable importance plots
plot(vip_scores)
```

The resulting variable importance plots are shown in Figure \ref{fig:variable-importance}. The plots show the relative importance of each feature in the model, with the x-axis representing the feature and the y-axis representing the variable importance score. The top plot shows the global variable importance scores, while the bottom plot shows the local variable importance scores. The plots are generated using the \texttt{plot.vip()} function, which accepts a \texttt{feature_importance} object as input.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{variable-importance.png}
\caption{Variable importance plots generated using the \texttt{vip} package. The top plot shows the global variable importance scores, while the bottom plot shows the local variable importance scores.}
\end{figure}

The variable importance plots generated using the \texttt{vip} package are easy to read and interpret, making it a useful tool for understanding the relative importance of each feature in a ML model. The \texttt{vip} package is designed to be flexible and easy to use, making it accessible to a wide range of users.

In conclusion, the \texttt{vip} package provides a powerful and easy-to-use tool for calculating variable importance scores and generating variable importance plots for a variety of ML models. It is a useful tool for understanding the relative importance of each feature in a ML model and for communicating the results of the model to others.

\begin{thebibliography}{10}
\bibitem{Molnar2019} Molnar, C. (2019). \textit{Interpretable machine learning: A guide for making black box models explainable}.\footnote{We 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.}
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).

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).

### Constructing VIPs in R

We’ll illustrate major concepts using the Friedman 1 benchmark problem described in Friedman (1991) and Breiman (1996):

\[
Y_i = 10 \sin (\pi X_{1i} X_{2i}) + 20 (X_{3i} - 0.5)^2 + 10X_{4i} + 5X_{5i} + \epsilon_i, \quad i = 1, 2, \ldots, n, \tag{1}
\]

These packages were discovered using pkgsearch’s ps() function (Csárdi and Salmon, 2019) with the key phrases "variable importance" and "feature importance".
where \( \epsilon_i \sim N(0, \sigma^2) \). 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.

```r
trn <- vip::gen_friedman(500, sigma = 1, seed = 101) # simulate training data
tibble::as_tibble(trn) # inspect output
```

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 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.

The R Journal Vol. 12/1, June 2020 ISSN 2073-4859
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.)

```r
# 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).

```r
# 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)
```

![Figure 1: Model-specific VIPs for the three different tree-based models fit to the simulated Friedman data.](image-url)
As we would expect, all three methods rank the variables $x_1$–$x_5$ 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 \texttt{vip} can help...one function to rule them all! Once \texttt{vip} is loaded, we can use \texttt{vi()} to extract a tibble of VI scores.

```r
# 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. 
#> 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 \texttt{vi()} function always returns a tibble\(^6\) with two columns: \texttt{Variable} and \texttt{Importance} (the exceptions are coefficient-based models which also include a \texttt{Sign} column giving the sign of the corresponding coefficient, and permutation importance involving multiple Monte Carlo simulations,\(^5\))

\(^5\) In order to avoid deprecation warnings due to recent updates to \texttt{tibble} and \texttt{ggplot2}, the code examples in this article are based on the latest development versions of both \texttt{vip} (version 0.2.2.9000) and \texttt{pdp} (version 0.7.0.9000); the URL to the development version of each package is available on its associated CRAN landing page.\(^6\) Technically, it's a tibble with an additional “\texttt{vi}” class.
but more on that later). Also, by default, \texttt{vi()} always orders the VI scores from highest to lowest; this, among other options, can be controlled by the user (see \texttt{?vip::vi} for details). Plotting VI scores with \texttt{vip()} is just as straightforward. For example, the following code can be used to reproduce Figure 1.

```r
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 \texttt{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 \texttt{vip()} and change a few useful options (the resulting plot is displayed in Figure 2). Note that we can also call \texttt{vip()} directly on a "vi" object if it's already been constructed.

```r
# 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.](image)

**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 (\texttt{trn}) allowing for all main effects and two-way interactions, then use the \texttt{step()} function to perform backward elimination. The resulting VIP is displayed in Figure 3.

```r
# Fit a LM
linmod <- lm(y ~ .^2, data = trn)
backward <- step(linmod, direction = "backward", trace = 0)

# Extract VI scores
(vi_backward <- vi(backward))
```

<table>
<thead>
<tr>
<th>Variable</th>
<th>Importance</th>
<th>Sign</th>
</tr>
</thead>
<tbody>
<tr>
<td>x4</td>
<td>14.2</td>
<td>POS</td>
</tr>
<tr>
<td>x2</td>
<td>7.31</td>
<td>POS</td>
</tr>
</tbody>
</table>

\(^7\)Since this approach is biased towards large-scale features it is important to properly standardize the predictors (before fitting the model) or the estimated coefficients.
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):

```r
# 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
```

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.
# 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 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).

**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
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. 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):

A similar approach is taken in the vivo package (Kozak and Biecek, 2019).
# 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 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 x1–x5 and x6–x10; 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 x1 and x2 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.

# 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)

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):
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).

Figure 9: ICE-based feature importance for the PPR and NN models fit to the simulated Friedman data.

```r
# 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)
}
```
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 2 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 $X_1, X_2, \ldots, X_j$ be the features of interest and let $M_{\text{orig}}$ 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 $i = 1, 2, \ldots, j$:
   (a) Permute the values of feature $X_i$ in the training data.
   (b) Recompute the performance metric on the permuted data $M_{\text{perm}}$.
   (c) Record the difference from baseline using $\text{imp} (X_i) = M_{\text{perm}} - M_{\text{orig}}$.
2. Return the VI scores $\text{imp} (X_1), \text{imp} (X_2), \ldots, \text{imp} (X_j)$.

**Algorithm 2:** A simple algorithm for constructing permutation-based VI scores.

Algorithm 2 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 $M_{\text{perm}}/M_{\text{orig}}$ 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.

```r
# 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 2 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.)

```r
# 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:

```r
list_metrics()
```

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>accuracy</td>
<td>Classification accuracy</td>
</tr>
<tr>
<td>2</td>
<td>error</td>
<td>Misclassification error</td>
</tr>
<tr>
<td>3</td>
<td>auc</td>
<td>Area under (ROC) curve</td>
</tr>
<tr>
<td>4</td>
<td>logloss</td>
<td>Log loss</td>
</tr>
<tr>
<td>5</td>
<td>mauc</td>
<td>Multiclass area under (ROC) curve</td>
</tr>
<tr>
<td>6</td>
<td>mae</td>
<td>Mean absolute error</td>
</tr>
<tr>
<td>7</td>
<td>mse</td>
<td>Mean squared error</td>
</tr>
<tr>
<td>8</td>
<td>r2</td>
<td>R squared</td>
</tr>
<tr>
<td>9</td>
<td>rsquared</td>
<td>R squared</td>
</tr>
<tr>
<td>10</td>
<td>rmse</td>
<td>Root mean squared error</td>
</tr>
<tr>
<td>11</td>
<td>sse</td>
<td>Sum of squared errors</td>
</tr>
</tbody>
</table>
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):

\[
MAE = \frac{1}{n} \sum_{i=1}^{n} |Y_i - \hat{f}(X_i)|, \tag{2}
\]

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).

```r
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)
```r
set.seed(2321) # for reproducibility
pfun <- function(object, newdata) predict(object, newdata = newdata)
nv(n, 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 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)
```r
set.seed(2327) # for reproducibility
vi(n, 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")
```

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.
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.

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.

# 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")

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 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).

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 increasing popular to assess global importance by aggregating local VI measures; in particular, Shapley explanations (Strumbelj 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 possess (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 17.¹⁰ (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.](image)

### 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

---

¹⁰Note that the exact = TRUE option is only available if you have fastshap version 0.0.4 or later

¹¹It’s been argued that approximate Shapley values share the same drawback, however, Janzing et al. (2019) makes a compelling case against those arguments.

¹²A basic R implementation is available at https://github.com/bgreenwell/rstratx.
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).

```r
# 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
```

<table>
<thead>
<tr>
<th>Variable</th>
<th>Importance</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>x4</td>
<td>3801.341</td>
<td><img src="image" alt="Sparkline" /></td>
</tr>
<tr>
<td>x2</td>
<td>2456.534</td>
<td><img src="image" alt="Sparkline" /></td>
</tr>
<tr>
<td>x1</td>
<td>2048.462</td>
<td><img src="image" alt="Sparkline" /></td>
</tr>
<tr>
<td>x5</td>
<td>1492.825</td>
<td><img src="image" alt="Sparkline" /></td>
</tr>
<tr>
<td>x3</td>
<td>952.217</td>
<td><img src="image" alt="Sparkline" /></td>
</tr>
<tr>
<td>x8</td>
<td>380.200</td>
<td><img src="image" alt="Sparkline" /></td>
</tr>
<tr>
<td>x10</td>
<td>368.846</td>
<td><img src="image" alt="Sparkline" /></td>
</tr>
<tr>
<td>x6</td>
<td>354.177</td>
<td><img src="image" alt="Sparkline" /></td>
</tr>
<tr>
<td>x7</td>
<td>346.030</td>
<td><img src="image" alt="Sparkline" /></td>
</tr>
<tr>
<td>x9</td>
<td>338.066</td>
<td><img src="image" alt="Sparkline" /></td>
</tr>
</tbody>
</table>

*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

---

**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).
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.

```r
# 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
```

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).

```r
# 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.

```r
# Setup parallel backend
library(doParallel) # load the parallel backend
c1 <- makeCluster(5) # use 5 workers
registerDoParallel(cl) # register the parallel backend
```
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.

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


P. Biecek. DALEX: Descriptive mAchine Learning EXplanations, 2019. URL https://CRAN.R-project.org/package=DALEX. R package version 0.4.9. [p343]


F. Doshi-Velez and B. Kim. Towards a rigorous science of interpretable machine learning, 2017. [p343]


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
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; Paternaude 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 Schlemmer 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 languages, 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 Grolemund, 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 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).

1 For 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.

2 See 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
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
  + }
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 -
       .......$ : 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) $\langle f, a_1, \ldots, a_n \rangle$, $n \geq 0$, which represents a call to $f$ with arguments $a_1, \ldots, a_n$ (which we are typically used to denote as $f(a_1, \ldots, a_n)$).

The above AST can be written in the Polish (prefix) notation as

$\langle '<-', d, \langle '+'*, \langle '-'-, x, y \rangle, \langle '-'-, x, y \rangle \rangle \rangle$

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

$'<'-(d, \langle '+'*\langle '-'-(x, y), \langle '-'-(x, y) \rangle \rangle\rangle$

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
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
```
Figure 1: Assumed Program Dependence Graph vertex types for the R language.

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 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 expression corresponding to \( u \) in the following manner:

\[
\begin{align*}
& \text{Id} & \text{Color} & \text{Type} & \text{Description} \\
0 & \text{Olive} & \text{Entry} & \text{marks the beginning of a function} \\
1 & \text{Light yellow} & \text{Header} & \text{loop} \\
2 & \text{Gray} & \text{Next} & \text{next} \\
3 & \text{Gray} & \text{Break} & \text{break} \\
4 & \text{Orange} & \text{If} & \text{a conditional expression} \\
5 & \text{Light blue} & \text{If_part} & \text{an expression to execute conditionally} \\
6 & \text{Gray} & \text{assignment} & \text{an assignment expression, e.g., } \text{name <- val} \\
7 & \text{Violet} & \text{parameter} & \text{a function parameter} \\
8 & \text{Gray} & \text{oneBracketSingle} & \text{an expression involving } [i], \text{e.g., vector[-1]} \\
9 & \text{Gray} & \text{oneBracketDouble} & \text{an expression involving } [i, j] \\
10 & \text{Gray} & \text{oneBracketTripleOrMore} & \text{an expression involving } [i, j, k, ...] \\
11 & \text{Gray} & \text{twoBrackets} & \text{an expression involving } [[i]], \text{e.g., list[["nameditem"]]} \\
12 & \text{Gray} & \text{dollar} & \text{an expression involving } $, \text{e.g., df$column} \\
13 & \text{Dark green} & \text{funNoArguments} & \text{a function call with no actual parameters} \\
14 & \text{Red} & \text{funOneArgument} & \text{a function call with one actual parameter} \\
15 & \text{Red} & \text{funTwoArguments} & \text{a function call with two actual parameters} \\
16 & \text{Red} & \text{funThreeArguments} & \text{a function call with three actual parameters} \\
17 & \text{Red} & \text{funFourOrMoreArguments} & \text{a function call with four or more actual parameters} \\
18 & \text{Gray} & \text{stopifnot} & \text{a call to stopifnot()} \\
19 & \text{Green} & \text{logicalOperator} & \text{a logical operator-based expression, e.g., } \text{& } \text{or}! \\
20 & \text{Blue} & \text{arithmeticOperator} & \text{an arithmetic operator-based expression, e.g., } +, -, * \text{or } \%\% \\
21 & \text{Blue} & \text{comparisonOperator} & \text{a comparison operator-based expression, e.g., } \text{==, } \text{<=, or } \text{<} \\
22 & \text{Green} & \text{return} & \text{a call to return()} \\
23 & \text{Cyan} & \text{colon} & \text{a colon operator-based expression, e.g., from:to} \\
24 & \text{Dark green} & \text{symbol} & \text{a symbol (name)} \\
25 & \text{Dark green} & \text{constant} & \text{a constant value, e.g., 1, "string" or NA}
\end{align*}
\]
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 \((v, u) \in E_F \) whenever there is an edge from \( v \) to \( u \), \( \zeta_F : V_F \to \{ \text{Entry, Header, \ldots, constant} \} = \{0, \ldots, 25\} \) gives the type of each vertex and \( \xi_F : E_F \to \{ \text{DATA, CONTROL} \} \) 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.

```r
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.

```r
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 \( X \in \mathbb{R}^{d \times n} \) representing \( n \) vectors in \( \mathbb{R}^d \) and a vector \( y \in \mathbb{R}^d \), the closest vector in \( X \) to \( y \) with respect to the Euclidean metric can be determined by evaluating \( \min(\text{apply}( (X-y)^2 , 2, \text{sum})) \). 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 \( g(x) \leftarrow g(x); f(g(x)) \). 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:

```r
if (cond) {
  return(A)
} else {
  return(B)
}
```

is semantically equivalent to:

```r
if (cond)
  return(A)
else
  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 ‘\( \cdot \)’ 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 $\mu$ 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, \( \text{clamp1}() \) (whose PDG from now on we will denote with \( F = (V_F, \xi_F) \), see Fig. 3) and \( \text{standardise}() \) (denoted \( G = (V_G, \xi_G) \), see Fig. 4).

\[ \begin{array}{c|c|c|c|c|c|c|c|c|c|c|c|c} \hline v & Entry & x & min(x) & max(x) & - (1) & < & If & If_part & NULL & - (2) & / \\
\hline \text{raw} & \delta_F(v) & 7.80 & 3.04 & 1.35 & 1.12 & 0.93 & 0.65 & 0.30 & 0.20 & 0.10 & 0.21 & 0.10 \\
\text{normalized} & & 0.49 & 0.19 & 0.09 & 0.07 & 0.06 & 0.04 & 0.02 & 0.01 & 0.01 & 0.01 & 0.01 \\
\hline \end{array} \]

**Table 2:** Vertex importance degrees in \( \text{clamp1}() \) (sum = 15.79, median = 0.04).

\[ \begin{array}{c|c|c|c|c|c|c|c|c|c|c|c|c} \hline v & Entry & x & sd(x) & < & If & If_part & NULL & mean(x) & - & / \\
\hline \text{raw} & \delta_G(v) & 4.33 & 1.71 & 0.93 & 0.65 & 0.30 & 0.20 & 0.10 & 0.33 & 0.21 & 0.1 \\
\text{normalized} & & 0.49 & 0.19 & 0.10 & 0.07 & 0.03 & 0.02 & 0.01 & 0.04 & 0.02 & 0.01 \\
\hline \end{array} \]

**Table 3:** Vertex importance degrees in \( \text{standardise}() \) (sum = 8.86, median = 0.03).

\[ \begin{array}{c|c|c|c|c|c|c|c|c|c|c|c|c} \hline v & Entry & x & min(x) & max(x) & - (1) & < & If & If_part & NULL & - (2) & / \\
\hline \xi_F(v) & 0 & 24 & 14 & 14 & 20 & 21 & 4 & 5 & 25 & 20 & 20 \\
\xi_F^1(v) & 0 & 1 & 2 & 2 & 4 & 5 & 6 & 7 & 3 & 3 \\
\xi_F^2(v) & 7 & 1 & 2 & 2 & 3 & 4 & 5 & 0 & 6 & 6 \\
\xi_F^3(v) & 8 & 5 & 6 & 6 & 7 & 1 & 2 & 3 & 0 & 4 & 4 \\
\hline \end{array} \]

**Table 4:** \( \text{clamp1}() \): Labels assigned to vertices in each iteration.

\[ \begin{array}{c|c|c|c|c|c|c|c|c|c|c|c|c} \hline v & Entry & x & sd(x) & < & If & If_part & NULL & mean(x) & - & / \\
\hline \xi_G(v) & 0 & 24 & 14 & 14 & 21 & 20 & 4 & 5 & 25 & 14 & 20 & 20 \\
\xi_G^1(v) & 0 & 1 & 2 & 4 & 5 & 6 & 7 & 2 & 3 & 3 \\
\xi_G^2(v) & 7 & 1 & 9 & 3 & 4 & 5 & 0 & 2 & 6 & 6 \\
\xi_G^3(v) & 12 & 9 & 10 & 1 & 2 & 3 & 0 & 11 & 4 & 4 \\
\hline \end{array} \]

**Table 5:** \( \text{standardise}() \): Labels assigned to vertices in each iteration.
1. **Vertex importance degrees.** Firstly, each vertex in $F$ is assigned an importance degree, $\delta_F : V_F \rightarrow \mathbb{R}_+$,

$$
\delta_F(v) = 0.1 + \sum_{(v,u) \in E_F} \left( 1 + 0.1 I_{\text{cond}((v,u))} \right) \delta_F(u),
$$

where $I_{\text{cond}}$ 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 $\delta_F$ is well-defined. Otherwise, we shall be computing the importance degrees in the depth-first manner.
Next, the importance degrees are normalised, \( \delta_F(v) \rightarrow \delta_F(v)/\sum_{u \in V} \delta_F(u) \). 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, \( \zeta_F(v), \zeta_G(u) \in \{0, \ldots, 25\} \), based on the type of operation it represents (see Tables 4 and 5).

In the \( i \)-th (out of \( h \) total, here we fix \( h = 3 \); see Bartoszuk, 2018 for discussion) iteration of the SimiRA algorithm, we assign new labels \( \zeta^i_F, \zeta^i_G \) according to the labels previously considered.

(a) Iteration \( i = 1 \). In the first iteration, the initial labels, \( \zeta_F, \zeta_G \), are simply remapped to consecutive integers. This yields \( \zeta^1_F \) and \( \zeta^1_G \) 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 \( \zeta^i_F(v) = \zeta^i_G(u) \) and:

\[
\sum_{(v, w) \in E_F, \zeta^i_F(w) \notin C(v, w)} \delta_F(w) \leq \min \{M_F, M_G\},
\]

where \( C(v, u) = \{\zeta^i_F(v) \in \zeta^i_F, (v, w) \in E_F \} = \{u, w) \in E_G \} \) 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 in 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 \( \zeta^i_F(v) = \zeta^i_G(u) = 0 \). Outgoing neighbours of \( v \) have types: 1 (\( \times \)), 3 (\( \cdot \)), 2 (\( \min(x) \)), 2 (\( \max(x) \)), 4 (\( \circ \)), 3 \((\cdot -1)\) and 5 (\( \cdot \)). Neighbours of \( u \) are labelled as 1 (\( \times \)), 3 (\( \cdot \)), 2 (\( \min(x) \)), 3 \((\cdot /)\), 2 \((\cdot s(x))\), 4 (\( \circ \)), 5 (\( \cdot \)). 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 \( \zeta^i_F(v) = \zeta^i_G(y) = 1 \). Their neighbours are: - (2) (importance=0.01; type=6), \( \min(x) \) (importance=0.09; type=2), \( \max(x) \) (importance=0.07; type=2) and - (2) (importance=0.07; type=6), \( \max(x) \) (importance=0.04; type=2), \( \cdot s(x) \) (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 \( L^i_F = (L^i_{F,1}, \ldots, L^i_{F,m}) \) be a vector of label counts, where \( L^i_{F,j} = |\{v \in V_F : \zeta^i_F(v) = j\}| \), see Table 6. We define \( L^i_G \) in much the same way, see Table 7. We introduce the following “partial” similarity measures of the label sequences – the symmetric:

\[
\mu(L^i_F, L^i_G) = 1 - \frac{\sum_{k=1}^m |L^i_{F,k} - L^i_{G,k}|}{\sum_{k=1}^m L^i_{F,k} + \sum_{k=1}^m L^i_{G,k}},
\]

and its nonsymmetric version:

\[
\tilde{\mu}(L^i_F, L^i_G) = \frac{\sum_{k=1}^m \min(L^i_{F,k}, L^i_{G,k})}{\sum_{k=1}^m L^i_{F,k}}.
\]

The partial similarities for \( i = 1, 2, 3 \) are given in Table 8.

4. Final similarity degrees. The overall similarity degree is defined as the arithmetic mean of the \( h = 3 \) partial similarities (reported in Table 8):

\[
\mu(F, G) = \frac{1}{h} \sum_{i=1}^h \mu(L^i_F, L^i_G) = \frac{0.95 + 0.86 + 0.57}{3} = 0.79,
\]
We obtain its nonsymmetric versions in the same way:
\[
\tilde{\mu}(F, G) = \frac{1}{n} \sum_{i=1}^{n} \tilde{\mu}(L_{jF}, L_{jG}) = \frac{0.91 + 0.82 + 0.55}{3} = 0.76,
\]
\[
\tilde{\mu}(G, F) = \frac{1}{n} \sum_{i=1}^{n} \tilde{\mu}(L_{jG}, L_{jF}) = \frac{1.00 + 0.90 + 0.60}{3} = 0.83.
\]

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() 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
```

Table 6: \texttt{clamp1()}: Label counts in each iteration.

<table>
<thead>
<tr>
<th>( j )</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>( L_{1F} )</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( L_{2F} )</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( L_{3F} )</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 7: \texttt{standardise()}: Label counts in each iteration.

<table>
<thead>
<tr>
<th>( j )</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>( L_{1G} )</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( L_{2G} )</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( L_{3G} )</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 8: Similarity measures in each iteration.

<table>
<thead>
<tr>
<th>( i )</th>
<th>( \mu(L_{jF}^{i}, L_{jG}^{i}) )</th>
<th>( \tilde{\mu}(L_{jF}^{i}, L_{jG}^{i}) )</th>
<th>( \tilde{\mu}(L_{jG}^{i}, L_{jF}^{i}) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.95</td>
<td>0.91</td>
<td>1.00</td>
</tr>
<tr>
<td>2</td>
<td>0.86</td>
<td>0.82</td>
<td>0.90</td>
</tr>
<tr>
<td>3</td>
<td>0.57</td>
<td>0.55</td>
<td>0.60</td>
</tr>
<tr>
<td>Final</td>
<td>0.79</td>
<td>0.76</td>
<td>0.83</td>
</tr>
</tbody>
</table>
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
R> SimilaR_fromTwoFunctions(clamp1, standardise, aggregation="both")
```

<table>
<thead>
<tr>
<th>name1</th>
<th>name2</th>
<th>SimilaR12</th>
<th>SimilaR21</th>
<th>decision</th>
</tr>
</thead>
<tbody>
<tr>
<td>clamp1</td>
<td>clamp1</td>
<td>0.7575758</td>
<td>0.8333333</td>
<td>1</td>
</tr>
</tbody>
</table>

**Example:** clamp2(). Let us recall the source code of the clamp1() function:

```r
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
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
R> SimilaR_fromTwoFunctions(clamp1, clamp2, aggregation="both")
```

<table>
<thead>
<tr>
<th>name1</th>
<th>name2</th>
<th>SimilaR12</th>
<th>SimilaR21</th>
<th>decision</th>
</tr>
</thead>
<tbody>
<tr>
<td>clamp1</td>
<td>clamp2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**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
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.
CONTRIBUTED RESEARCH ARTICLES

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")

<table>
<thead>
<tr>
<th>name1</th>
<th>name2</th>
<th>SimilaR12</th>
<th>SimilaR21</th>
<th>decision</th>
</tr>
</thead>
<tbody>
<tr>
<td>clamp1_vectorised1</td>
<td>clamp1_vectorised2</td>
<td>0.7833333</td>
<td>0.9215686</td>
<td>1</td>
</tr>
</tbody>
</table>

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
+             file_text <- paste0(file_text, f, collapse="\n") # extract source code
+         }
+     writeLines(file_text, con=paste0("desc","", pkg,".R"))
+ }

R> SimilaR_fromTwoFunctions("DescTools", "nortest", aggregation="both")

<table>
<thead>
<tr>
<th>name1</th>
<th>name2</th>
<th>SimilaR12</th>
<th>SimilaR21</th>
<th>decision</th>
</tr>
</thead>
<tbody>
<tr>
<td>DescTools</td>
<td>nortest</td>
<td>0.7833333</td>
<td>0.9215686</td>
<td>1</td>
</tr>
</tbody>
</table>

Which indicates a significant degree of similarity, which is indeed the case.
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
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
R> print(head(results, 10))
```

For greater readability, the results are reported in Table 9.

<table>
<thead>
<tr>
<th>Name1 (DescTools)</th>
<th>Name2 (nortest)</th>
<th>SimilaR12</th>
<th>SimilaR21</th>
<th>Decision</th>
</tr>
</thead>
<tbody>
<tr>
<td>CramerVonMisesTest()</td>
<td>cvm.test()</td>
<td>1.0000000</td>
<td>1.0000000</td>
<td>1</td>
</tr>
<tr>
<td>LillieTest()</td>
<td>lillie.test()</td>
<td>1.0000000</td>
<td>1.0000000</td>
<td>1</td>
</tr>
<tr>
<td>PearsonTest()</td>
<td>pearson.test()</td>
<td>1.0000000</td>
<td>1.0000000</td>
<td>1</td>
</tr>
<tr>
<td>ShapiroFranciaTest()</td>
<td>sf.test()</td>
<td>1.0000000</td>
<td>1.0000000</td>
<td>1</td>
</tr>
<tr>
<td>CramerVonMisesTest()</td>
<td>ad.test()</td>
<td>0.9451477</td>
<td>0.9105691</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name1 (DescTools)</th>
<th>Name2 (nortest)</th>
<th>SimilaR12</th>
<th>SimilaR21</th>
<th>Decision</th>
</tr>
</thead>
<tbody>
<tr>
<td>CramerVonMisesTest()</td>
<td>lillie.test()</td>
<td>0.7299578</td>
<td>0.5339506</td>
<td>0</td>
</tr>
<tr>
<td>LillieTest()</td>
<td>cvm.test()</td>
<td>0.5339506</td>
<td>0.7299578</td>
<td>0</td>
</tr>
<tr>
<td>LillieTest()</td>
<td>ad.test()</td>
<td>0.5339506</td>
<td>0.7032520</td>
<td>0</td>
</tr>
<tr>
<td>Eps()</td>
<td>sf.test()</td>
<td>0.5333333</td>
<td>0.6315789</td>
<td>0</td>
</tr>
<tr>
<td>ShapiroFranciaTest()</td>
<td>ad.test()</td>
<td>0.7719298</td>
<td>0.3577236</td>
<td>0</td>
</tr>
</tbody>
</table>

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:

```r
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")
}
```
 \begin{verbatim}
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)
}
\end{verbatim}

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 \texttt{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 \texttt{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 \textit{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 \textit{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 \texttt{DescTools} and \texttt{nortest} packages. Recall that most of the cloned function pairs are correctly identified, proving the practical usefulness of the \texttt{SimilaR} package. The reported above-threshold similarity between \texttt{CramerVonMisesTest()} and \texttt{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) \texttt{AndersonDarlingTest()} (generic distribution) with its specific version of \texttt{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 \texttt{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 \texttt{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**


The R Journal Vol. 12/1, June 2020

ISSN 2073-4859


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. [p373]


Linear Fractional Stable Motion with the \texttt{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 \texttt{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. \texttt{rlfsm} could be found on CRAN and gitlab.

Introduction

The linear fractional stable motion (shortly, lfsm) \((X_t)_{t \in \mathbb{R}}\) on a filtered space \((\Omega, \mathcal{F}, (\mathcal{F}_t)_{t \in \mathbb{R}}, \mathbb{P})\) is defined via

\[
X_t = \int_{\mathbb{R}} \{ (t - s)^{H-1/\alpha} - (-s)^{H-1/\alpha} \} dL_s, \quad x_+ := \max\{x, 0\},
\]

where \(L_t\) is a symmetric \(\alpha\)-stable Lévy motion, \(\alpha \in (0, 2)\), with the scaling parameter \(\sigma > 0\) and the self-similarity parameter \(H \in (0, 1)\). The \texttt{lfsm} 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 \texttt{lfsm} in Mazur et al. (2020).

We proceed with introduction to existing software, with interest towards study of numerical properties of statistical estimators for \texttt{lfsm} 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 \texttt{lfsm} and related estimators prior to \texttt{rlfsm}. There is a particularly simple and useful numerical algorithm for simulating \texttt{lfsm}s 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 \texttt{lfsm} generator as a MATLAB function. However, some useful packages, that could be used in numerical routines with Lévy-driven processes (e.g. to create \texttt{lfsm} generator and perform unit testing), exist and have been implemented in R. For instance, R package \texttt{somebm} (Huang, 2013) contains functions for generation of fractional Brownian motion (fBM). Currently archived by CRAN \texttt{dvfBm} (Coeurjolly, 2009) has routines for generation of fBM and estimator of the Hurst parameter of the latter. \texttt{stabledist} (Wuertz et al., 2016) and \texttt{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 \texttt{lfsm} and its implementation in our \texttt{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, \(X_1, X_2, \ldots, X_n\). In the case of high-frequency, we use points with discretization equal to the length of the path vector, \(X_{1/m}, X_{2/m}, \ldots, X_1\). 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 lfsm 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 \((1, 2, \ldots, N)\) or \((1/n, 2/n, \ldots, 1)\), rescaling in time should be performed using the equality \( (aH_X)_t \geq 0 \overset{d}{=} (X_{at})_{t \geq 0} \) 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 lfsm 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 \( \alpha \)-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 lfsm. The idea underlying this sample path generator is that it should be always possible not only to obtain lfsm path, but also the underlying Lévy motion, generated during the procedure, and since the core function of lfsm is deterministic it should allow for lfsm 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 lfsm by path. Black arrows: when the algorithm initially is given the parameters, it generates Lévy motion, and then lfsm. Green arrows: when Lévy motion is needed without lfsm 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 lfsm.

The function path can be used by

```r
path(N, m, M, alpha, H, sigma, freq='H', 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 lfsm in order 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 lfsr generator. Technically, in the path the seed is set just before Lévy increments are generated. The path function returns a list containing the lfsr, 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 (\(c, \alpha, H\)) that were used to generate the lfsr, and the predefined frequency.

Generation of symmetric \(\alpha\)-stable (sas) random variables is powered by function \(r\text{stable}\) from package \text{stabledist} with \(8\) parametrization based on the Zolotarev’s representation for an \(\alpha\)-stable distribution with some modifications. \(8\) 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 lfsr - Lévy motion and to switch between them. We recall that direct computation of the sum approximating the integral in the definition of lfsr (1) would involve number of operations proportional to \(NMm\), which makes the method slow. Instead, the original algorithm by Stoev and Taqqu (2004) suggests computing increments of lfsr with the help of

\[
W(n) := \sum_{j=1}^{mM} a_{H,m}(j) Z_{\alpha}(n - j),
\]

where \(W(mk)\) is a discretized and truncated version of the increments of the lfsr, and in the limit has the same distribution as them

\[
\{W(mk), k = 1, \ldots, N\} \xrightarrow{d} \{X(k) - X(k - 1), k = 1, \ldots, N\};
\]

\(Z_{\alpha}(k)\) are i.i.d. sas random variables that have indexes \(-mM, \ldots, lfsr - 1\) and scaling parameter equal to 1, and

\[
a_{H,m}(j) := C_{H,\alpha}^{-1}(m, M) \left( (j/m)^{H-1/\alpha} - (j/m - 1)^{H-1/\alpha} \right) m^{-1/\alpha}, j \in \mathbb{N}
\]

with

\[
C_{H,\alpha}(m, M) := m^{-1} \left( \sum_{j=1}^{mM} \left| (j/m)^{H-1/\alpha} - (j/m - 1)^{H-1/\alpha} \right| \right)^{1/\alpha}.
\]

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 \(NMm\). 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

\[
X = \text{DTFT}\{x_n\}(\omega) = \sum_{m=-\infty}^{\infty} x_n \exp(-i\omega n).
\]
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

\[ (A \ast B)[n] := \sum_{m=-\infty}^{\infty} A[m] B[n - m]. \]

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

\[ (x \ast y)[n] = \text{IDFT} \{\text{DTFT} \{x_n\}(\cdot) \times \text{DTFT} \{y_n\}(\cdot)\}. \]

**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**

\[ (x_N \ast y)[n] = \text{IDFT} \{\text{DFT}(x_N) \text{DFT}(y_N)\} \]

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 \( (a \ast Z_{m(M+N)})[n] \), values with indexes \( [mM; m(M+N) - 1] \) 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 \( a \ast Z \) we compute one period of \( (a \ast Z_{m(M+N)})[n] \) 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.

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 lfsm sample path that is needed to be simulated. In this case path function performs an index shift using...
Figure 4: Example of transformation of vectors $a$ and $Z$ into sequences before computing their convolution.

the following property:

$$(a * x_c)[n] := \sum_{k=-\infty}^{\infty} a[k] \cdot x[n - k - c]$$

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.

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.

```r
# 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 ...
$ lfsm : 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_lfsm<-List[["lfsm"]]/max(abs(List[["lfsm"]]))
Norm_olm<-List[["levy_motion'"]]/max(abs(List[["levy_motion'"]]))
```

```
# Visualization of the paths
plot(Norm_lfsm, col=2, type="l", ylab="coordinate")
lines(Norm_olm, col=3)
leg.txt <- c("lfsm", "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 lfsm from scratch and computing based on an existing sample path of the Lévy motion.

```r
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 \textit{GenLowEstim} could be considered unbiased starting approximately from 1000 points.

The second example compares empirical standardized densities of estimates, obtained by \textit{GenLowEstim} with the limiting standard normal ones, Figure 8.

```r
S<-c(1e2,1e3,1e4)
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)

l_plot<-Plot_dens(par_vec=c('Var\sigma','Var\alpha','VarH'), MC_data=tilda_est, Nnorm=1e7)
ggg<-grid.arrange(l_plot[[1]],l_plot[[2]],l_plot[[3]],nrow=1,ncol=3)
```

Figure 8: Empirical distributions of tilde-estimates, described in Section 3.2.
In short, in these examples for different path lengths $s$, $N_{\text{Monte Carlo}}$ LFSM paths are simulated. To each path we apply tilde-statistic (see Section 3.2), therefore obtaining $N_{\text{Monte Carlo}}$ estimates ($\tilde{\sigma}_{\text{low}}, \tilde{\alpha}_{\text{low}}, \tilde{H}_{\text{low}}$) 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 LFSM 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 $N_{\text{Monte Carlo}}$ LFSM paths of the length $s[i]$ via path_fast function. To each of the paths we apply all the estimators to obtain $H$, $\alpha$, and $\sigma$ 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 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 $N_{\text{Monte Carlo}}$ 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 the \( k \)-th iterated increments of step \( r \) of a sample path. In particular, \( \Delta_{ij}^k X := X_i^n - X_{i-1}^n, \) and \( \Delta_{ij}^{0,k} X := X_i^n - 2X_{i-1}^n + X_{i-2}^n \). 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

\[
\Delta_{ik}^r X := \sum_{j=0}^{k} (-1)^j \binom{k}{j} X_{(i-rj)/n}. \tag{4}
\]

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 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 \( \sim 10 \) 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 \( \alpha-H \) 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 \( \alpha \) and \( H \) values. `Plot_list_paths()` takes the data frame as an argument and plots the sample paths on different panels based on their \( (\alpha, H) \) 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.

```r
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)
```

```
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
```

```r
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 1.1218238 -6.1284620 -2.2450357 ...
$ alpha  : Factor w/ 3 levels "0.5","1","1.5": 1 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 1 ...
$ freq   : Factor w/ 1 level "H": 1 1 1 1 1 1 1 1 1 1 ...
```

```
Plot_list_paths(arr)
```
Parameter estimation of the linear fractional stable motion

In this section, we describe estimators for the parameters $H$, $\alpha$, and $\sigma$ 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 $(X_t)_{t \in \mathbb{R}}$ is locally Hölder continuous of any order up to $H - 1/\alpha$. Moreover, this condition implies the following restrictions

$$a \in (1, 2) \quad \text{and} \quad H \in (1/2, 1) \quad (5)$$

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

$$\hat{H}_{\text{high}}(p, k)_n := \frac{1}{p} \log_2 \left( \frac{\sum_{i = 2k}^{n} | \Delta_{n,k}^{2} X \|^p}{\sum_{i = 2k}^{n} | \Delta_{n,k}^{1} X \|^p} \right),$$

$$\hat{H}_{\text{low}}(p, k)_n := \frac{1}{p} \log_2 \left( \frac{\sum_{i = 2k}^{n} | \Delta_{2,k}^{2} X \|^p}{\sum_{i = 2k}^{n} | \Delta_{2,k}^{1} X \|^p} \right).$$

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

$$V_{\text{high}}(f; k, r)_n := \frac{1}{n} \sum_{i = r}^{n} f \left( n^{H} \Delta_{n,k}^{2} X \right) \quad V_{\text{low}}(f; k, r)_n := \frac{1}{n} \sum_{i = r}^{n} f \left( n^{H} \Delta_{2,k}^{1} X \right), \quad (6)$$

where $f : \mathbb{R} \rightarrow \mathbb{R}$ is a measurable function. Estimators for the stability index $a$ of the driving stable
motion in high and low frequency setting are based on the empirical characteristic functions given by
\[
\phi_{\text{high}}(t; H, k) := V_{\text{high}}(\phi_r; k)_n \quad \text{and} \quad \phi_{\text{low}}(t; k) := V_{\text{low}}(\phi_r; k)_n
\]
with \(\phi_r(x) := \cos(tx)\), for two different values \(t_1\) and \(t_2\) such that \(t_2 > t_1 > 0\). Let us note that the empirical characteristic function \(\phi_{\text{high}}(t; H, k)\) depends on the parameter \(H\) while \(\phi_{\text{low}}(t; k)\) does not. Thus, we should infer the self-similarity parameter \(H\) by \(\hat{H}_{\text{high}}(p, k)_n\) and then we should use the plug-in estimator \(\hat{\phi}_{\text{high}}(t; \hat{H}_{\text{high}}(p, k)_n, k)_n\) to infer the stability index \(\alpha\) in high-frequency setting. Estimators for the parameter \(\alpha\) are given by
\[
\hat{\alpha}_{\text{high}} := \frac{\log |\log \phi_{\text{high}}(t_2; \hat{H}_{\text{high}}(p, k)_n, k)_n| - \log |\log \phi_{\text{high}}(t_1; \hat{H}_{\text{high}}(p, k)_n, k)_n|}{\log t_2 - \log t_1},
\]
\[
\hat{\alpha}_{\text{low}} := \frac{\log |\log \phi_{\text{low}}(t_2; k)_n| - \log |\log \phi_{\text{low}}(t_1; k)_n|}{\log t_2 - \log t_1}.
\]

Estimators for the scale parameter \(\sigma\) 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 \(h_{k,r} : R \to R\) as follows:
\[
h_{k,r}(x) = \sum_{j=0}^{k} (-1)^j \binom{k}{j} (x - jr)^{H - 1/2}, \quad x \in R,
\]
where \(k, r \in N\), and let \(\|h_{k,r}\|_{\infty} := \int_R |h_{k,r}(s)|^a ds\). Let us note that the function \(h_{k,r}\) depends on two parameters \(a\) and \(H\) which need to be pre-estimated. Estimators for the parameter \(\sigma\) are expressed as
\[
\hat{\sigma}_{\text{high}} := \left( -\log \phi_{\text{high}}(t_1; \hat{H}_{\text{high}}(p, k)_n, k) \right)^{1/\hat{\alpha}_{\text{high}}} / t_1 \|h_{k,1}\|_{\hat{\alpha}_{\text{high}}},
\]
\[
\hat{\sigma}_{\text{low}} := \left( -\log \phi_{\text{low}}(t_1; k) \right)^{1/\hat{\alpha}_{\text{low}}} / t_1 \|h_{k,1}\|_{\hat{\alpha}_{\text{low}}}.\]

**Parameter estimation in the general case**

Here, we consider general case when an explicit lower bound for \(\alpha\) 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
\[
\hat{H}_{\text{low}}(-p, k)_n := \frac{1}{p} \log_2 \left( \frac{\sum_{j=2k}^{n} |\Delta^2_j X^{-p}|}{\sum_{j=2k}^{n} |\Delta^1_j X^{-p}|} \right).
\]
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 \(k > H + 1/4\). That’s why we consider the preliminary estimator of \(\alpha\) with \(k = 1\) that is consistent given by
\[
\hat{\alpha}_{\text{low}}^0(t_1, t_2)_n = \log |\log \phi_{\text{low}}(t_2; 1)_n| - \log |\log \phi_{\text{low}}(t_1; 1)_n| / \log t_2 - \log t_1.
\]
Since we do not know if \(\hat{\alpha}_{\text{low}}^0(t_1, t_2)_n\) is in the domain of attraction, we define the estimator of the parameter \(k\) as
\[
k_{\text{low}}(t_1, t_2)_n := 2 + \lfloor \hat{\alpha}_{\text{low}}^0(t_1, t_2)_n^{-1} \rfloor.
\]
In the second step we use estimator \(\hat{k}_{\text{low}} := \hat{k}_{\text{low}}(t_1, t_2)_n\) for the estimation of parameters \(H, \alpha\) and \(\sigma\). In particular, we get the following consistent estimators
\[
\hat{H}_{\text{low}}(-p, \hat{k}_{\text{low}})_n := \frac{1}{p} \log_2 \left( \frac{\sum_{j=2\hat{k}_{\text{low}}}^{n} |\Delta^2_j X^{-p}|}{\sum_{j=2\hat{k}_{\text{low}}}^{n} |\Delta^1_j X^{-p}|} \right),
\]
\[
\hat{\alpha}_{\text{low}}(\hat{k}_{\text{low}}; t_1, t_2)_n = \log |\log \phi_{\text{low}}(t_2; \hat{k}_{\text{low}})_n| - \log |\log \phi_{\text{low}}(t_1; \hat{k}_{\text{low}})_n| / \log t_2 - \log t_1,
\]
\[
\hat{\sigma}_{\text{low}}(\hat{k}_{\text{low}}; t_1, t_2)_n = \left( -\log \phi_{\text{low}}(t_1; \hat{k}_{\text{low}}) \right)^{1/\hat{\alpha}_{\text{low}}} / t_1 \|h_{\hat{k}_{\text{low}},1}\|_{\hat{\alpha}_{\text{low}}}.
\]
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 \(\hat{H}_{\text{high}}(-p)_n = \).
We introduce function $\text{ContinEstim}$ which does not exist in the case when $p$ with $k = 1, f_{-p}(x) = |x|^{-p}$ and preliminary estimator $\hat{H}_\text{high}(-p)_n$ for the parameter $H$. It is remarkable that $\phi(\cdot)$ is always invertible for all $p \neq p'$ (see Dang and Istat (2017)). Consequently, we can define the estimator of $k$ in high-frequency setting by

$$\hat{k}_\text{high} := \hat{k}_\text{high}(p, p') = 2 + [\hat{a}_\text{high}(p, p')^{-1}]_n.$$

Thus, consistent estimators of $H, \alpha$ and $\sigma$, in high-frequency setting are given by

$$\hat{H}_\text{high}(-p, \hat{k}_\text{high})_n = \frac{1}{p} \log_2 \left( \frac{\sum_{i=2\hat{k}_\text{high}}^{n} \Delta_{i, \hat{k}_\text{high}} X^{-p}}{\sum_{i=2\hat{k}_\text{high}}^{n} \Delta_{i, \hat{k}_\text{high}}^{-1} X^{-p}} \right),$$

$$\hat{a}_\text{high}(\hat{k}_\text{high}; t_1, t_2)_n = \phi^{-1} \left( \frac{V_{\text{high}}(f_{-p}, \hat{H}_\text{high}(-p), \hat{k}_\text{high})_n}{V_{\text{high}}(f_{-p}, \hat{H}_\text{high}(-p), 0)_n} \right),$$

$$\hat{a}_\text{high}(\hat{k}_\text{high}; p, p')_n = \left( \hat{a}_\text{high}^{n-p} V_{\text{high}}(f_{-p}, \hat{H}_\text{high}(-p))_n \right) \phi/\|H_{\text{high}}\|_{\text{high}}.$$

### Implementation in R

We introduce function $\text{ContinEstim}$ 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 $\text{alpha, hat, }H, \text{hat and sigma\_hat$ responsible for retrieving the corresponding parameters. $\text{sigma\_hat$ is called using tryCatch as the former may return an error due to numerical integration in $\text{Norm\_alpha.$}}$

General low-frequency estimation technique, described in Section 3.2 is implemented in $\text{GenLowEstim}$.

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 $a_0$. Using these $H_0$ and $a_0$, a new $k$ is obtained through $2 + \text{floor}(\text{alpha, hat}(\cdot))$, and then the new $k$ is used for the same estimation procedure as in $\text{ContinEstim}$. This approach induces an effect, which does not exist in the case when $\text{ContinEstim}$ is applied. When $a$ is smaller than, or close to $2/N$, where $N$ is the observed lfsm 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 $\Delta_{i, \text{low}}^2 X$ in statistic $\hat{H}_\text{low}(-p, \hat{k}_\text{low})_N$. 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 $(\hat{H}, \hat{a}, \hat{\sigma})$, 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 $a$ and $H$. Real distributions of estimates in this case are left unexplored.

High-frequency estimator from the same section was implemented in $\text{GenHighEstim}$.

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 \((H,\alpha)\), \(N\text{monteC}\) sample paths of the both frequencies were generated, and to each of them the relevant procedures \(\text{ContinEstim}, \text{GenLowEstim}\) and \(\text{GenHighEstim}\) 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 \((\hat{H},\hat{\alpha})\in (0,1)\times(0,2)\).

(a) Comparison of success rates for \(\text{ContinEstim}\) and \(\text{GenLowEstim}\). Low frequency case. Path length \(N=200\), number of sample paths \(N\text{monteC}=300\).

(b) Comparison of success rates for \(\text{ContinEstim}\) and \(\text{GenHighEstim}\). High frequency case. Path length \(N=200\), number of sample paths \(N\text{monteC}=300\).

This experiment shows (Figures 11a and 11b) that in both high- and low-frequency cases \(\text{ContinEstim}\) gives much better precision than the corresponding general estimator. The outcome is rigorous in low-frequency technique since \(\text{ContinEstim}\) and \(\text{GenLowEstim}\) 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 \(\text{Errfilter}\) function in the experiment.

Figure 11: Comparison of success rates of estimators.
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 $\alpha$ and choose two sets of parameters- one for $H$ and one for $L$. Then, for each combination of $H$ 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 $k_{new} = 2 + \lfloor \alpha - 0.5 \rfloor$ where $\alpha$ is obtained via alpha_hat with parameters $k = 1, 2$, and freq=L plugged-in. This provides us simulated distribution of $k_{\text{new}}$ (Figure 12). Also, we fix a set $k_{\text{ind}} = \text{seq}(1,8,\text{by}=1)$ and, given a path, for each of these $k$’s extract statistics $q_{low}(t, k = k_{\text{ind}})$ and $\alpha_{low}(t_1, t_2; k = k_{\text{ind}})$, 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 $\alpha$ determines which regime is in presence.

Due to small variance of $\hat{\alpha}_{low}(t_1, t_2)_n$ (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.

![Figure 12: Histograms of preliminary estimations of $k, \hat{k}_{low}(1, 2)_n$. $\alpha$’s are on vertical labels, $H$’s- on horizontal.](image)

When $\alpha$ 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 $\alpha^{-1} \in N$

$$P \left( \hat{k}_{low} = 2 + \alpha^{-1} \right) \to \lambda \quad \text{and} \quad P \left( \hat{k}_{low} = 1 + \alpha^{-1} \right) \to 1 - \lambda$$

for some constant $\lambda \in (0, 1)$, see Mazur et al. (2020), Section 4.1. Surprisingly, $\lambda$ 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 $\alpha$ 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 $\tilde{\kappa}_{low}(1, 2)_n$. These values plugged-in to $\tilde{\alpha}_{low}(k, t_1, t_2)_n$ produce again very dispersed estimates of parameter $\alpha$. This mechanism explains why $\tilde{\alpha}_{low}$ 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 $\tilde{\alpha}_{low}$ behaves could be explained using pic.(13), where $\varphi_n$ and $V_{low}(\psi, k)_n$ are plotted. Cases wherein $\tilde{\alpha}_{low}$ performs poorly coincide with ones wherein $\varphi_n$ and $V_{low}(\psi, k)_n$ are significantly distant from each other, so convergence $V_{low}(\psi, k)_n \overset{d}{\rightarrow} \varphi_n(t; k)$ isn’t observed at the given length of sample paths, which ruins the whole idea of $(\sigma, \alpha)$ estimation. Of course, this effect doesn’t affect $H$-estimation because it is based on ratio statistic, which has a different form.

![Figure 13](image13.png)

**Figure 13:** Comparison of the real $\varphi_n(t = 1; k)$ and the one estimated via $V_{low}(\psi = 1, k)_n$ on the logarithmic scale. $\alpha$’s are on vertical labels, $H$’s- on horizontal. The lower and upper box sides correspond to the 25th and 75th percentiles.

![Figure 14](image14.png)

**Figure 14:** Convergence of $\tilde{\alpha}_{low}(k, t_1, t_2)_n$ to the real $\alpha$ (red line) for different $k$. $\alpha$’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 Lévy-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 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, ω, 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 lfsm could be used for other types of stochastic integral.

Classes for simulated lfsm

Here we describe the least general classes- “SimulatedLfsmLow” and “SimulatedLfsmHigh”, objects of which are obtained by simulating low- and high-frequency linear fractional stable motions. Figure

![Diagram of class structure](image)

Figure 15: Structure of the classes of simulated lfsm. 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.

```r
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,
```

---

The R Journal Vol. 12/1, June 2020 ISSN 2073-4859
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 'AlphaSigma' [package ".GlobalEnv"] with 3 slots
.. .. ..@ alpha: num 1.8
.. .. ..@ H : num 1.8
.. .. ..@ sigma: num 1.8
..@ 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 “Simu-
latedLfsmLow”. The last function, ContinInfer, is a generic which has a registered method for class
“StochasicProcLow”, general stochastic processes in low-frequency setting. Since “SimulatedLfsmLow”
inherits from “StochasicProcLow”, the generic dispatched this method and performed statistical
inference. ContinInfer was designed to perform inference according to Theorem 3.1 from (Mazur
et al., 2020) and is based on R function ContinInfer. One can see that plot (and, less obviously,
ContinInfer) 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


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. [p386]

Stepan Mazur
School of Business, Örebro University
Fakultetsgatan 1, SE-701 82 Örebro
Sweden
ORCID: 0000-0002-1395-9427
stepan.mazur@oru.se
Dmitry Otryakhin\textsuperscript{1}
Department of Mathematics, Aarhus University
Ny Munkegade 118, DK-8000 Aarhus C
Denmark
ORCID: 0000-0002-4700-7221
d.otryakhin.acad@protonmail.ch

\textsuperscript{1}Some 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-calibratated techniques. Confidence intervals can also 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, \( U = \{1, \ldots, k, \ldots, N\} \). Let \( sU \) be a volunteer non-probability sample of size \( n_U \), 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:
\[ \hat{Y}_{HT} = \sum_{k \in s} w_{vk} y_k \]  
(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 \( x_k \) be the value taken on unit \( k \) by a vector of auxiliary variables which population total is assumed to be known \( X = \sum_{k=1}^{N} x_k \). 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

\[ \sum_{k \in s} w_k x_k = X. \]  
(2)

Given a pseudo-distance \( G(w_k, w_{vk}) \), the calibration process consists in finding the solution to the minimization problem

\[ \min_{w_k} \{ \sum_{k \in s} G(w_k, w_{vk}) \} \]  
(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

\[ \hat{Y}_{reg} = \sum_{sv} w_{sv} y_k = \sum_{sv} d_{sv} y_k + (X - \sum_{sv} w_{sv} x_k) \beta_{sv} \]  
(4)

where \( \hat{\beta}_{sv} \) is

\[ \hat{\beta}_{sv} = T_s^{-1} \sum_{sv} w_{sv} x_k y_k \]  
(5)

being \( T_s = \sum_{sv} w_{sv} x_k x_k' \).
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, \( x \), 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, \( \pi \), can be displayed as

\[
\pi(x) = \frac{1}{e^{-(\gamma^T x)} + 1}
\]

for some vector \( \gamma \), 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):

\[
\hat{Y}_{PSA1} = \sum_{k \in s_V} w_{Vk} y_k / \hat{\pi}(x_k) = \sum_{k \in s_V} y_k \hat{w}^{PSA1}_k
\]

where \( \hat{\pi}(x_k) \) is the estimated response propensity for the individual \( k \) of the volunteer sample as predicted using covariates \( x \).

- 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

\[
\hat{w}^{PSA2}_k = \frac{1 - \hat{\pi}(x_k)}{\hat{\pi}(x_k)}
\]

and resulting estimator for the population total is given by

\[
\hat{Y}_{PSA2} = \sum_{k \in s_V} y_k \hat{w}^{PSA2}_k
\]

- Valliant and Dever (2011) use the propensity scores to post-stratify the sample. The process is: sort the combined sample by \( \hat{\pi}(x_k) \); 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:

\[
\hat{Y}_{PSA3} = \sum_{g} \sum_{k \in s_{Vg}} w_{Vk} y_k / \bar{\pi}_g = \sum_{g} \sum_{k \in s_{Vg}} y_k \hat{w}^{PSA3}_k
\]

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:

\[
f_g = \frac{\sum_{k \in s_{Rg}} w_{Rk}}{\sum_{k \in s_{Rg}} w_{Rk}} / \frac{\sum_{k \in s_{Vg}} w_{Vk}}{\sum_{k \in s_{Vg}} w_{Vk}}
\]
where \( s_{R_k} \) is the set of individuals in the reference sample that are in the \( g \)th class of propensity scores and \( s_{V_k} \) is the set of individuals in the volunteer sample that are in the \( g \)th class of propensity scores. Finally, the adjusted weights \( w_{PSA} \) are the product of the original weights and the adjustment factor; following the same notation, the adjusted weight for individual \( k \) in \( s_{V_k} \) (i.e. the individual \( k \) of the \( g \)th propensity class in the volunteer sample) is computed as

\[
w_{PSA}^g = w_{V_k} f_g
\]  

and the estimator is given by

\[
\hat{Y}_{PSA} = \sum_{k \in s_{V_k}} \sum_{j \in s_{R_k}} y_{k} w_{PSA}^g
\]

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, \( \hat{\gamma} \), 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 \( \hat{\gamma} = \frac{1}{n} \sum_{k \in s_{V_k}} w_{PSA}^g y_k \) be the estimator of the mean of \( y \), his Leave-One-Out Jackknife estimator of the variance is given by:

\[
V(\hat{\gamma}) = \frac{n - 1}{n} \sum_{j=1}^{n} (\hat{\gamma}_{(j)} - \hat{\gamma})^2
\]

where \( \hat{\gamma}_{(j)} \) is the value of the estimator \( \hat{\gamma} \) after dropping unit \( j \) from \( s_{V} \) and \( \hat{\gamma} \) is the mean of values \( \hat{\gamma}_{(j)} \).

Given that PSA weights are estimated from the available data, the exclusion of one unit can have an impact on the values of \( w \); 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:

\[
V_{rw}(\hat{\gamma}) = \frac{n - 1}{n} \sum_{j=1}^{n} (\hat{\gamma}_{rw(j)} - \hat{\gamma}_{rw})^2
\]

where \( \hat{\gamma}_{rw(j)} = \frac{1}{n} \sum_{k \in s_{V_k} - \{j\}} w_{PSA}^g (j) y_k \) with \( w_{PSA}^g (j) \) the PSA weight obtained from the sample \( s_{V} - \{j\} \) and \( \hat{\gamma}_{rw} \) is the mean of values \( \hat{\gamma}_{rw(j)} \).

### Statistical matching

The statistical matching method was introduced by Rivers (2007). The idea is to model the relationship between \( y_k \) and \( x_k \) using the volunteer sample \( s_{V} \) in order to predict \( y_k \) for the reference sample. That is, the matching estimator is given by:

\[
\hat{Y}_{SM} = \sum_{x_k} \hat{g}_k w_{Rk}
\]

being \( \hat{g}_k \) the predict value of \( y_k \).

The key is how to predict the values \( y_k \). Usually \( \hat{g}_k = x_k \hat{\beta} \) being \( \hat{\beta} = \frac{\sum_{k \in s_{V}} y_k x_k}{\sum_{k \in s_{V}} x_k^2} \) 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 (Buekens 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 $y = (y_1, ..., y_N)'$ is a realization of super-population random variables $Y = (Y_1, ..., Y_N)'$ having a superpopulation model $\xi$. To incorporate auxiliary information $x_k$ available for all $k \in U$ on assume a superpopulation for $y$ built on some mean function of $x$:

$$Y_k = m(x_k) + e_k, \quad k = 1, ..., N.$$  \hspace{1cm} (16)

The random vector $e = (e_1, ..., e_N)'$ 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, $x$, measured in $s_V$ and $s_V = U - s_V$ it is possible to estimate the values of $y$ in $s_V$ with regression modeling such that the estimated value of $y$ for an individual $k$ can be calculated through the following expression:

$$\hat{y}_k = E_m(y_k|x_k) \hspace{1cm} (17)$$

$m$ alludes to the specific model which provides the expectation of $y_k$, and $x_k$ are the values of the $k$-th individual in the covariates $x$.

We can use the auxiliary information in several ways to define several estimators:

- the model-based estimator:
  $$\hat{Y}_m = \sum_{k \in s_V} y_k + \sum_{k \in s_V} \hat{y}_k \hspace{1cm} (18)$$

- the model-assisted estimator:
  $$\hat{Y}_{ma} = \sum_{k \in U} \hat{y}_k + \sum_{k \in s_V} (y_k - \hat{y}_k)w_V k \hspace{1cm} (19)$$

- the model-calibrated estimator:
  $$\hat{Y}_{mcal} = \sum_{k \in s_V} y_k w_{CAL} k \hspace{1cm} (20)$$

where $w_{CAL}$ are such that they minimize $\sum_{k \in s_V} G(w_{CAL}, w_v k)$, where $G(\cdot, \cdot)$ is a particular distance function, subject to

$$\sum_{k \in s_V} w_{CAL} k \hat{y}_k = \sum_{k \in U} \hat{y}_k.$$

Usually the linear regression model is used, $E_m(y_k|x_k) = x_k'\beta$ 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 use 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. (Bueltens 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 Ferni-Garcia and Rueda (2018); sampleP represents a probability sample of size \(n_p = 500\) extracted by simple random sampling from a frame covering the entire population, while sampleNP represents a non-probability sample of size \(n_p = 1000\) extracted 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 \((\text{vote}_\text{gen}, \text{vote}_\text{pens}, \text{vote}_\text{pir})\) measuring whether an individual would vote to a given party \("\text{gen}\", "\text{pens}\" or "\text{pir}\") in an election or not. Probabilities of voting to party "\text{gen}\", "\text{pens}\" or "\text{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

<table>
<thead>
<tr>
<th>Weight recalculation</th>
<th>PSA algorithm</th>
<th>R function</th>
<th>Elapsed time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>Logistic regression</td>
<td>glm</td>
<td>0.0049999876</td>
</tr>
<tr>
<td>Yes</td>
<td>Logistic regression</td>
<td>glm</td>
<td>75.56034</td>
</tr>
<tr>
<td>Yes</td>
<td>CART</td>
<td>rpart</td>
<td>102.3409</td>
</tr>
<tr>
<td>Yes</td>
<td>Random Forest</td>
<td>rf</td>
<td>203.7737</td>
</tr>
<tr>
<td>Yes</td>
<td>GBM</td>
<td>gbm</td>
<td>453.731</td>
</tr>
<tr>
<td>Yes</td>
<td>Neural Network</td>
<td>nnet</td>
<td>719.733</td>
</tr>
</tbody>
</table>

Table 1: Total elapsed time of Leave-One-Out Jackknife variance estimation under recalculation 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.

```r
> library(NonProbEst)
> head(sampleNP)
vote_gen vote_pens vote_pir education_primaria education_secundaria education_terciaria age sex language
1 0 0 1 0 0 0 66 1 1
2 0 0 1 0 0 0 30 1 1
3 0 0 1 0 0 0 62 0 1
4 0 0 1 0 0 0 30 0 1
5 0 0 1 0 0 0 69 1 1
6 0 0 0 0 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 auxiliary variables, as they have been measured for the entire population. They can be retrieved from the population dataset:
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.

```r
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:

```r>
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).

```r>
sampleNP$vote_gen/nrow(sampleNP)
[1] 0.096
> sampleNP$vote_pens/nrow(sampleNP)
[1] 0.346
> 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
```
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:

```r
### 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.

```r
> 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.

```r
> head(sampleP)
education_primaria education_secundaria education_terciaria age sex
1      1             0             0      35     1
2      0             0             0      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.
```r
> 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:

```r
> 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 situa-
tions. 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:

```r
> 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 neigh-
bours technique for coupling field and remotely sensed data in forest surveys. Remote Sensing of
B. Buelens, J. Burger, and J. A. van den Brakel. Comparing inference methods for non-probability


The R Journal Vol. 12/1, June 2020 ISSN 2073-4859


Maria 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  
lúiscastro193@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 Bergantños 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 ProjectManagement¹ (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 last times of

¹https://github.com/Juan-Goncalves-Dosantos/ProjectManagement.git
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 $i = (x_1, x_2) \in N$ starts at node $x_1 \in X$ and ends at $x_2 \in X$. A node $x_a \in X$ is a source node if there is no arc $i \in N$ such that $x_2 = x_a$. A node $x_c \in X$ is a sink node if there is no arc $i \in N$ such that $x_1 = x_c$.

A cycle is a set of arcs $i_0, i_1, \ldots, i_m \in N$ such that $x_{i_0} = x_{i_m}$ and $x_{i_1} = x_{i_2}$. To illustrate the concept of directed graph consider the following example. Take graph $G = (X, N)$ given by $X = \{a, b, c, d\}$ and $N = \{(1, a), (2, a, c), (3, b, d), (4, c, d)\}$. 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.](image)

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 $x$, $\text{Pred}(x) = \{i \in N/ x_{i2} = x\}$, and the immediate successors of $x$ as $\text{Suc}(x) = \{i \in N/ x = x_{i1}\}$. We define the earliest time $D^F_i(G, x^0)$ of an activity $i \in N$ as the minimum time required to complete all immediate predecessor activities of $x_{i1}$, i.e. the earliest start time the activity $i$ can start taking into account the graph

$$D^F_i(G, x^0) = \max_{j \in \text{Pred}(x_{i1})} \{D^F_j(G, x^0) + x^0_j\}.$$
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

\[
D_i^L (G, x^0) = \begin{cases} 
\max_{j \in N : \text{Suc} (j, i) = \emptyset} \{ D_j^E (G, x^0) + x_i^j \} & \text{if } \text{Suc} (i, 2) = \emptyset, \\
\min_{j \in \text{Suc} (i, 2)} \{ D_j^L (G, x^0) - x_i^j \} & \text{otherwise.}
\end{cases}
\]

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

\[
FS_i (G, x^0) = \min_{j \in \text{Suc} (i, 2)} \{ D_j^F (G, x^0) \} - D_i^E (G, x^0) - x_i^0.
\]

The independent slack of an activity is the maximum time that the activity duration can be increased without affecting the times of others activities

\[
IS_i (G, x^0) = \max \{ \min_{j \in \text{Suc} (i, 2)} \{ D_j^F (G, x^0) \} - D_i^L (G, x^0) - x_i^0, 0 \}.
\]

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

\[
D_i^L (G, x^0) = D_i^E (G, x^0) + S_i (G, x^0)
\]

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 \((P, x^0, c, D)\), where \(P\) is a deterministic project, \(x^0 \in \mathbb{R}_+^n\) is the vector of minimum durations, that is, for each activity \(i \in N\), \(x_i^0\) 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
\[
C (D (G, y)) = D (G, y) - \delta
\]
with \( \delta \in \mathbb{R}_+ \), for example \( \delta = D (G, x^0) \).

We denote by \( CP^N \) the family of all deterministic projects with delays with set of activities \( N \), and by \( CP \) the family of all deterministic projects with delays.

In a deterministic project with delays \( CP \in CP^N \), 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 \( x_i \geq x_j^0 \), 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 \( \phi \) is defined, for each \( i \in N \), by
\[
\phi_i = \frac{x_i - x_j^0}{\sum_{j \in N} x_j - x_j^0} \cdot C (D (G, x)).
\]

The Truncated Proportional rule for deterministic scheduling problems with delays \( \bar{\phi} \) is defined, for each \( i \in N \), by
\[
\bar{\phi}_i = \frac{\min \{ x_i - x_j^0, C (D (G, x)) \}}{\sum_{j \in N} \min \{ x_j - x_j^0, C (D (G, x)) \}} \cdot C (D (G, x)).
\]

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 one 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 \( (N, \nu) \) where \( N \) is a finite non-empty set, and \( \nu \) is a map from \( 2^N \) to \( \mathbb{R} \) with \( \nu (\emptyset) = 0 \). We say that \( N \) is the player (activity) set of the game and \( \nu \) is the characteristic function of the game, and we usually identify \( (N, \nu) \) with its characteristic function \( \nu \). The Shapley value, an allocation rule in cooperative game theory, is a map \( \Phi \) that associates to each TU-game \( (N, \nu) \) a vector \( \Phi (\nu) \in \mathbb{R}^N \) satisfying \( \sum_{S \subseteq N} \Phi_i (\nu) = \nu (N) \) and providing a fair allocation of \( \nu (N) \) among the players in \( N \). The explicit formula of the Shapley value for every TU-game \( (N, \nu) \) and every \( i \in N \) is given by
\[
\Phi_i (\nu) = \frac{\sum_{S \subseteq N \setminus \{ i \}} (|N| - |S| - 1)! |S|!}{|N|!} (\nu (S \cup \{ i \}) - \nu (S)).
\]

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 Patrone (2008)).

Bergantiños et al. (2018) define the Shapley rule for deterministic projects with delays \( Sh \) as \( Sh (CP) = \Phi (\nu^{CP}) \), where for all \( CP = (G, x^0, x, C) \in CP^N \):

- \( \nu^{CP} \) is the TU-game with set of players \( N \) given by
  \[
  \nu^{CP} (S) = C \left( D \left( G, (x_S, x^0_{N \setminus S}) \right) \right)
  \]
  for all \( S \subseteq N \), where \( (x_S, x^0_{N \setminus S}) \) denotes the vector in \( \mathbb{R}^N \) whose \( i \)-th component is \( x_i \) if \( i \in S \) or \( x^0_i \) if \( i \in N \setminus S \), and

- \( \Phi (\nu^{CP}) \) denotes the proposal of the Shapley value for \( \nu^{CP} \).

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 \( x^0_i \), the planned duration of activity \( i \in N \), we consider the non-negative random variable \( X^0_i \) 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 $0 \in \mathbb{R}^n$ 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 $\phi$ 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 $\hat{\phi}$ is defined, for each $i \in N$, by

$$\hat{\phi}_i = \frac{\min\{x_i - E(X_i^0) : C(D(G, x))\}}{\sum_{j \in N} \min\{x_j - E(X_j^0) : C(D(G, x))\}} \cdot C(D(G, x)).$$

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
  $$v^{SCP}(S) = E\left(C\left(D\left(G, \left(x_S, X^0_{\bar{N}\backslash S}\right)\right)\right)\right)$$

  for all non-empty $S \subseteq N$, 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 $SSh2(SCP) = \Phi(v^{SCP}_1) + \Phi(v^{SCP}_2)$, where

- $v^{SCP}_1$ is the TU-game with set of players $N$ given by
  $$v^{SCP}_1(S) = E\left(C\left(D\left(G, \left(x_S, X^0_{\bar{N}\backslash S}\right)\right)\right)\right) - E\left(C\left(D\left(G, X^0\right)\right)\right)$$

  for all $S \subseteq N$,

- $v^{SCP}_2$ is the TU-game with set of players $N$ given by
  $$v^{SCP}_2(S) = E\left(C\left(D\left(G, X^0, 0_{\bar{N}\backslash S}\right)\right)\right)$$

  for all $S \subseteq N$, where $0 \in \mathbb{R}^n$ is the vector with all components equal to zero, and

- $\Phi(v^{SCP}_1)$ and $\Phi(v^{SCP}_2)$ denote the proposal of the Shapley value for $v^{SCP}_1$ and $v^{SCP}_2$.

In general, the calculation of $v^{SCP}$, $v^{SCP}_1$ and $v^{SCP}_2$ 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 to plot graphs, the next two are related to game-theoretic concepts and the last one to solve linear programming problems.

---

2As in all TU-games, we define $v^{SCP}(\emptyset) = 0$. 

The R Journal Vol. 12/1, June 2020 ISSN 2073-4859
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.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dag.plot</td>
<td>Plots the AON graph of a project.</td>
</tr>
<tr>
<td>delay.pert</td>
<td>Calculates the delay cost of a deterministic project and allocates it among the activities.</td>
</tr>
<tr>
<td>delay.stochastic.pert</td>
<td>Calculates the delay cost of a stochastic project and allocates it among the activities.</td>
</tr>
<tr>
<td>early.time</td>
<td>Calculates the earliest start time for each activity.</td>
</tr>
<tr>
<td>last.time</td>
<td>Calculates the latest completion time for each activity.</td>
</tr>
<tr>
<td>levelling.resources</td>
<td>Calculates the schedule of the project so that the consumption of resources is as uniform as possible.</td>
</tr>
<tr>
<td>mce</td>
<td>Calculates the costs per activity needed to accelerate the project.</td>
</tr>
<tr>
<td>organize</td>
<td>Relabels the activities of a project (if i precedes j then i ≤ j).</td>
</tr>
<tr>
<td>rebuild</td>
<td>Builds a type 1 precedence matrix.</td>
</tr>
<tr>
<td>resource.allocation</td>
<td>Calculates the project schedule so that resource consumption does not exceed the maximum available per time period.</td>
</tr>
<tr>
<td>schedule.pert</td>
<td>Calculates the duration of a project and the schedule of each activity, and plots the schedule and the AON graph.</td>
</tr>
<tr>
<td>stochastic.pert</td>
<td>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.</td>
</tr>
</tbody>
</table>

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.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>duration</td>
<td>Vector with the expected duration for each activity.</td>
</tr>
<tr>
<td>prec1and2</td>
<td>Matrix indicating precedence type 1 or type 2 between the activities (Default=matrix(0)).</td>
</tr>
<tr>
<td>prec3and4</td>
<td>Matrix indicating precedence type 3 or type 4 between the activities (Default=matrix(0)).</td>
</tr>
<tr>
<td>observed.duration</td>
<td>Vector with the actual duration for each activity.</td>
</tr>
<tr>
<td>delta</td>
<td>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.</td>
</tr>
<tr>
<td>distribution</td>
<td>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.</td>
</tr>
<tr>
<td>values</td>
<td>Matrix with the arguments of the distribution function of the duration for each activity. By rows the activities, and by columns the arguments.</td>
</tr>
<tr>
<td>percentile</td>
<td>Value used to calculate the maximum time allowed for the duration of the project without delay. This value is only used if no delta value is assigned.</td>
</tr>
<tr>
<td>compilations</td>
<td>Number of simulations that the function uses for estimation (Default=1000).</td>
</tr>
<tr>
<td>cost.function</td>
<td>Delay cost function. If this value is not added, the package uses equation 1.</td>
</tr>
<tr>
<td>early.times</td>
<td>Vector with the early time for each activity.</td>
</tr>
<tr>
<td>PRINT</td>
<td>Logical parameter indicating if the schedule and the AON graph are depicted (Default=TRUE).</td>
</tr>
<tr>
<td>plot.activities.times</td>
<td>Vector of selected activities from which it is shown the distribution of their early and last times (Default=NULL).</td>
</tr>
<tr>
<td>minimum.durations</td>
<td>Vector with the minimum duration an activity can take even if the resources are increased.</td>
</tr>
<tr>
<td>critical.activities</td>
<td>Vector with the critical activities to represent them in a different color in the AON graph (Default=NULL).</td>
</tr>
<tr>
<td>duration.project</td>
<td>Value indicating the minimum time sought in the project (Default=NULL).</td>
</tr>
<tr>
<td>activities.costs</td>
<td>Vector indicating the cost of accelerating a unit of time the duration for each activity.</td>
</tr>
<tr>
<td>resources</td>
<td>Vector indicating the necessary resources for each activity per period of time.</td>
</tr>
<tr>
<td>int</td>
<td>Value indicating the duration of each period of time (Default=1).</td>
</tr>
<tr>
<td>max.resources</td>
<td>Value indicating the maximum number of resources that can be used in each period of time.</td>
</tr>
</tbody>
</table>

Table 2: Summary of parameters in `ProjectManagement`. 
<table>
<thead>
<tr>
<th>Function</th>
<th>duration</th>
<th>prec1and2</th>
<th>prec3and4</th>
<th>observed.duration</th>
<th>delta</th>
</tr>
</thead>
<tbody>
<tr>
<td>dag.plot</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>delay.pert</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>delay.stochastic.pert</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>early.time</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>last.time</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>levelling.resources</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mce</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>organize</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>resource.allocation</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>schedule.pert</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>stochastic.pert</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3: Arguments used by each function in ProjectManagement.

<table>
<thead>
<tr>
<th>Function</th>
<th>distribution</th>
<th>values</th>
<th>percentile</th>
<th>cost.function</th>
<th>compilations</th>
</tr>
</thead>
<tbody>
<tr>
<td>dag.plot</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>delay.pert</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>delay.stochastic.pert</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>early.time</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>last.time</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>levelling.resources</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mce</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>organize</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>resource.allocation</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>schedule.pert</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>stochastic.pert</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 4: Arguments used by each function in ProjectManagement.

<table>
<thead>
<tr>
<th>Function</th>
<th>early.times</th>
<th>PRINT</th>
<th>plot.activities.times</th>
<th>minimum.durations</th>
<th>critical.activities</th>
</tr>
</thead>
<tbody>
<tr>
<td>dag.plot</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>delay.pert</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>delay.stochastic.pert</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>early.time</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>last.time</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>levelling.resources</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mce</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>organize</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>resource.allocation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>schedule.pert</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>stochastic.pert</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 5: Arguments used by each function in ProjectManagement.
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

```R
> library("ProjectManagement")
```

Next we analyse the following deterministic project with 10 activities. Their durations and precedence relations are given in Table 7.

<table>
<thead>
<tr>
<th>N</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Immediate precedence type 1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>1,4</td>
<td>2</td>
<td>5,8</td>
<td>6</td>
</tr>
<tr>
<td>Immediate precedence type 2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Immediate precedence type 3</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>8</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Immediate precedence type 4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>9</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Durations

2 1.5 1 4.5 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.

```R
> 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:

```R
> 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]]

<table>
<thead>
<tr>
<th>Activities</th>
<th>Duration</th>
<th>Earliest start time</th>
<th>Latest start time</th>
<th>Earliest completion time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.0</td>
<td>0.0</td>
<td>5.5</td>
<td>2.0</td>
</tr>
<tr>
<td>2</td>
<td>1.5</td>
<td>0.0</td>
<td>0.0</td>
<td>1.5</td>
</tr>
<tr>
<td>3</td>
<td>1.0</td>
<td>0.0</td>
<td>2.0</td>
<td>1.0</td>
</tr>
<tr>
<td>4</td>
<td>1.5</td>
<td>1.5</td>
<td>6.0</td>
<td>3.0</td>
</tr>
<tr>
<td>5</td>
<td>2.0</td>
<td>1.5</td>
<td>6.5</td>
<td>3.5</td>
</tr>
<tr>
<td>6</td>
<td>2.5</td>
<td>2.0</td>
<td>3.0</td>
<td>5.5</td>
</tr>
<tr>
<td>7</td>
<td>3.0</td>
<td>3.0</td>
<td>7.5</td>
<td>6.0</td>
</tr>
<tr>
<td>8</td>
<td>4.0</td>
<td>1.5</td>
<td>1.5</td>
<td>5.5</td>
</tr>
<tr>
<td>9</td>
<td>2.0</td>
<td>5.5</td>
<td>8.5</td>
<td>7.5</td>
</tr>
<tr>
<td>10</td>
<td>5.0</td>
<td>5.5</td>
<td>5.5</td>
<td>10.5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Activities</th>
<th>Latest completion time</th>
<th>Slack</th>
<th>Free Slack</th>
<th>Independent Slack</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7.5</td>
<td>5.5</td>
<td>1.0</td>
<td>0.0</td>
</tr>
<tr>
<td>2</td>
<td>1.5</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>3</td>
<td>3.0</td>
<td>2.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>4</td>
<td>7.5</td>
<td>4.5</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>5</td>
<td>8.5</td>
<td>5.0</td>
<td>2.0</td>
<td>0.0</td>
</tr>
<tr>
<td>6</td>
<td>5.5</td>
<td>2.0</td>
<td>2.0</td>
<td>0.0</td>
</tr>
<tr>
<td>7</td>
<td>10.5</td>
<td>4.5</td>
<td>4.5</td>
<td>0.0</td>
</tr>
<tr>
<td>8</td>
<td>5.5</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>9</td>
<td>10.5</td>
<td>3.0</td>
<td>3.0</td>
<td>0.0</td>
</tr>
<tr>
<td>10</td>
<td>10.5</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

[[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.
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

\[ x^0 = (1, 1, 0.5, 1, 2, 2, 3, 1, 3) \]

and the costs per unit time to shorten each activity

\[ c = (1, 2, 1, 3, 2, 1, 2, 3, 5) \]

```r
> minimum.durations<-c(1,1,0.5,1,2,2,3,1,3)
> activities.costs<-c(1,2,1,3,2,1,2,3,5)
> 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 =

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
</tr>
<tr>
<td>1.5</td>
<td>1.5</td>
<td>1.0</td>
</tr>
<tr>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>1.5</td>
<td>1.5</td>
<td>1.5</td>
</tr>
<tr>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
</tr>
<tr>
<td>2.5</td>
<td>2.5</td>
<td>2.5</td>
</tr>
<tr>
<td>3.0</td>
<td>3.0</td>
<td>3.0</td>
</tr>
<tr>
<td>3.5</td>
<td>3.0</td>
<td>3.0</td>
</tr>
<tr>
<td>5.0</td>
<td>5.0</td>
<td>4.5</td>
</tr>
</tbody>
</table>

Costs per solution =

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>0.0</td>
<td>0.0</td>
<td>1.5</td>
</tr>
<tr>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

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

```r
> resources<-c(6,6,6,3,4,2,1,2,3,1)
> levelling.resources(duration,prec1and2,prec3and4,resources,int=0.5)
> 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.

```r
> max.resources<-10
> resource.allocation(duration,prec1and2,prec3and4,resources,
max.resources,int=0.5)
> 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=
\([1] 6 6 6 8 8 7 7 8 8 6 5 10 10 10 10 2 2 2 2 1\)
```
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

\[x = (2.5, 1.5, 2, 2, 6, 4, 6, 3, 5.5)\].

> observed.duration<-c(2.5,1.5,2,2,6,4,6,3,5.5)
> 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

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>The proportional payment</td>
<td>0.15000</td>
<td>0.00000</td>
<td>0.30000</td>
<td>0.15000</td>
<td>0.00000</td>
</tr>
<tr>
<td>The truncated proportional payment</td>
<td>0.15789</td>
<td>0.00000</td>
<td>0.31579</td>
<td>0.15789</td>
<td>0.00000</td>
</tr>
<tr>
<td>Shapley rule</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.33333</td>
<td>0.00000</td>
<td>0.00000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>The proportional payment</td>
<td>1.05000</td>
<td>0.30000</td>
<td>0.60000</td>
<td>0.30000</td>
<td>0.15000</td>
</tr>
<tr>
<td>The truncated proportional payment</td>
<td>0.94737</td>
<td>0.31579</td>
<td>0.63158</td>
<td>0.31579</td>
<td>0.15789</td>
</tr>
<tr>
<td>Shapley rule</td>
<td>1.08333</td>
<td>0.00000</td>
<td>1.08333</td>
<td>0.00000</td>
<td>0.50000</td>
</tr>
</tbody>
</table>

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 \(\exp(a)\) denotes
the exponential distribution with parameter \(a,\) 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
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.

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.

```r
> delay.stochastic.pert(prec1and2,prec3and4,distribution,values,
observed.duration,percentile=NULL,delta=NULL,cost.function)

Total delay of the stochastic project = 3
```

Figure 7: Density estimation of project duration time and earliest start and latest completion times for activities 7 and 8.
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.

<table>
<thead>
<tr>
<th>Activities</th>
<th>2</th>
<th>4</th>
<th>6</th>
<th>8</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>delay.pert</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.03</td>
<td>0.11</td>
</tr>
<tr>
<td>delay.stochastic.pert</td>
<td>0.06</td>
<td>0.44</td>
<td>1.68</td>
<td>6.23</td>
<td>30.58</td>
</tr>
<tr>
<td>early.time</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>last.time</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>levelling.resources</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.02</td>
<td>0.03</td>
</tr>
<tr>
<td>mce</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>organize</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>resources.allocation</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.01</td>
<td>0.02</td>
</tr>
<tr>
<td>schedule.pert</td>
<td>0.08</td>
<td>0.11</td>
<td>0.13</td>
<td>0.11</td>
<td>0.12</td>
</tr>
<tr>
<td>stochastic.pert</td>
<td>0.02</td>
<td>0.03</td>
<td>0.05</td>
<td>0.04</td>
<td>0.04</td>
</tr>
</tbody>
</table>

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


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. [p423]


The Rockerverse: Packages and Applications for Containerisation with R

by Daniel Nüst, Dirk Eddelbuettel, Dom Bennett, Robrecht Cannoodt, Dao Clark, Gergely Daróczi, 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 (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

---

1 https://www.r-project.org/conferences/
4 “Reproducible” in the sense of the Claerbout/Donoho/Peng terminology (Barba, 2018).
5 https://en.wikipedia.org/wiki/LXC
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 containers, 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.

**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

---

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.

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 API 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 generation.

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). 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.

---

7https://docs.docker.com/engine/api/latest/
8See https://github.com/richfitz/stevedore/blob/master/development.md.
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 dockermachine 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 dockermachine (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 Hub. 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_O_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 environments 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 containers 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

---

See https://github.com/Bioconductor/bioconductor_docker and https://hub.docker.com/u/bioconductor respectively.
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/data-science-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/rsms-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 packages. 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 processing15, 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 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 Gigantum’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.

---

11https://conda.io/
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
14Dockerfile available on GitHub: https://github.com/radiant-rstats/docker.
15https://docs.gigantum.com/docs/contributing/setting-up-environment-management
17See https://docs.gigantum.com/docs/environment-management
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 experts, 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 such as `renv` and `versions` or by using MRAN for R, can greatly increase the reproducibility of built images and are common approaches.

`dockerfile` 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 it’s mapping of package system dependencies to platform-specific installation package names. `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 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 computers and to assert a specific degree of confidence in the stability of a developed software (cf. README.Rmd in Marwick, 2017).

---

15See, 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/doc/docker.

16See https://sysreqs.r-hub.io/.

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/OGC 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 libraries.

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). xch/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-ubsan-clang 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 compilers. The images go beyond the configurations, or flavours, used by CRAN for checking packages, 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 tests. 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

23 https://cran.r-project.org/web/checks/check_flavors.html
24 dockertest is not actively maintained, but mentioned still because of its interesting approach.
intended for publication on Docker Hub. \textit{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 \textit{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 \texttt{.travis.yml} with each change, even this manuscript is rendered into a PDF and deployed to a GitHub-hosted website), 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 \texttt{Map} function for various schedulers (Lang et al., 2017). For example, the package can schedule jobs with Docker Swarm. \texttt{googleComputeEngineR} has the function \texttt{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. \texttt{googleComputeEngineR} integrates with the R parallelisation package \texttt{future} (Bengtsson, 2020a) to enable this with only a few lines of R code\textsuperscript{25}. 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 \texttt{cloudRunR}\textsuperscript{26}, which uses Cloud Run to create a scalable R-based API using \texttt{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 \texttt{googleCloudRunner} demonstrates how R can be used as the control language for one-time or batch processing jobs and scheduling of jobs\textsuperscript{27}. \texttt{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. \texttt{drake} has been demonstrated to run inside containers for high reproducibility\textsuperscript{28}. Furthermore, \texttt{drake} workflows have been shown to use future package’s function \texttt{makeClusterPSOCK()} for sending parts of the workflow to a Docker image for execution\textsuperscript{29} (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 \textit{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., \texttt{containerit},

\textsuperscript{25}https://cloudytech.github.io/googleComputeEngineR/articles/massive-parallel.html
\textsuperscript{26}https://github.com/MarkEdmondson1234/cloudRunR
\textsuperscript{27}https://code.markedmonson.me/googleCloudRunner/articles/cloudbuild.html
\textsuperscript{28}See 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.
\textsuperscript{29}https://docs.ropensci.org/drake/index.html?q=docker#with-docker
provide Docker images that can be used very similar to a CLI, but this usage is cumbersome. 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 GitLab 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.

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 (Guayder 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 preamade Docker image, which is configured and run in the cloud VM. For example, by specifying the template parameter with 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.

---

30https://o2r.info/containerit/articles/container.html
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 credit.1 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 R2 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 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 builds.3 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

---

1 See “Deploying a prediction service with Plumber” vignette for details: https://cran.r-project.org/web/packages/AzureContainers/vignettes/vig01_plumber_deploy.html.
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. 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 Nolins and Werdel, 2019), and others tell which customers are likely to make a repeat purchase. If a data scientist creates 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://github.com/att/rcloud/tree/master/docker) is a cloud-based platform for data analysis, visualisation and collaboration using R. It provides a rocker/dr base image for easy evaluation of the platform.

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 description file, e.g., project dependency configurations or simple configuration files. 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 and rApache (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

See useR2017 talk ‘Stream processing with R in AWS’.
35https://github.com/att/rcloud/tree/master/docker
36See supported file types at https://repo2docker.readthedocs.io/en/latest/config_files.html. For R, the
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://wholetale.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 objects\(^\text{37}\) 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\(^\text{38}\). The build pack combines a rocker/geospatial image with repo2docker’s composability\(^\text{39}\). 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 desktop and laptop computers. The first to 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-AP1’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 (Devissetty 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 HTCondor and Kubernetes for orchestrating container-based analysis and integrates with external HPC, i.e., NSF-XSEDE, through TAPIS (TACC-AP1’s). 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.

\(^{37}\)In 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.

\(^{38}\)See https://github.com/whole-tale/repo2docker_wholetale.

\(^{39}\)Composability refers to the ability to combine multiple package managers and their configuration files, such as R’s `R CMD install`, `R CMD build`, and `R CMD check`, and to include them in the Dockerfile along with a RockerBuildPack. See https://github.com/cyverse-ci/repo2docker.

\(^{37}\)https://github.com/cyverse-vc/ for Dockerfiles and configuration scripts; images are auto-built on DockerHub at https://hub.docker.com/u/cyversevice.
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 a very useful tool for interfacing 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 Dodona. It also installs the packages required for the testing framework and the exercises so that this does not have

41 https://github.com/dodona-edu/docker-images/blob/master/dodona-r.dockerfile
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). The rocker-pl image extends rocker/r-base with the pfR 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.

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 the 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 extended, 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 rttools (https://github.com/benmarwick/rttools) 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

---

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.

43See full literature list at https://research-compendium.science/.
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 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/docker, 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, dockerfiler, r-ci and r-db. EH contributed content on dockyard. SL contributed content on dockr. 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 interfaces for Docker in R (dockyard). DC contributed content on analogsea. 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

The R Journal Vol. 12/1, June 2020
ISSN 2073-4859
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-BR004. LS and NT are supported by US NIH / NHGRI awards U41HG00405 and U24HG010263. CW is supported by the Whole Tale project (https://wholetale.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


C. Boettiger, R. Lovelace, M. Howe, and J. Lamb. rocker-org/geospatial, Dec. 2019. [p438]


Datadog. 8 surprising facts about real Docker adoption, June 2018. URL https://www.datadoghq.com/docker-adoption/. [p437, 438]


R Core Team. 1999. URL: https://cran.r-project.org/doc/manuals/r-devel/R-exts.html. [p443, 450]

R-hub project. R-hub Docs, 2019. URL: https://docs.r-hub.io/. [p443]

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. [p449]


Trestle Technology, LLC. *plumber: An API Generator for R*, 2018. URL: https://CRAN.R-project.org/package=plumber. R package version 0.4.6. [p444]


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 Daróczi

Mark Edmondson  
IIH Nordic A/S, Google Developer Expert for Google Cloud Platform  
Artillerivej 86  
2300 København S, Denmark

Colin Fay  
ThinkR  
50 rue Arthur Rimbaud  
93300 Aubervilliers, France

Ellis Hughes  
Fred Hutchinson Cancer Research Center  
Vaccine and Infectious Disease  
1100 Fairview Ave. N., P.O. Box 19024  
Seattle, WA 98109-1024, USA

Lars Kjeldgaard  
Danish Tax Authorities  
Oestbanegade 123  
2100, København Oe

Sean Lopp  
RStudio, Inc  
250 Northern Ave  
Boston, MA 02210, USA

Ben Marwick  
University of Washington  
Department of Anthropology  
Denny Hall 230, Spokane Ln  
Seattle, WA 98105, USA

Heather Nolis  
T-Mobile  
12920 Se 38th St.  
Bellevue, WA, 98006, USA

Jacqueline Nolis  
Nolis, LLC  
Seattle, WA, USA

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
0000-0002-5910-4010
lori.shepherd@roswellpark.org

Péter Sólymos
Analythium Solutions
#258 150 Chippewa Road
Sherwood Park, AB, T8A 6A2, Canada
0000-0001-7337-1740
peter@analythium.io

Tyson Lee Swetnam
University of Arizona
1657 E Helen St.
Tucson, AZ, 85721, USA
0000-0002-6639-7181
tswetnam@arizona.edu

Nitesh Turaga
Roswell Park Comprehensive Cancer Center
Elm & Carlton Streets
Buffalo, NY, 14263, USA
0000-0002-0224-9817
nitesh.turaga@roswellpark.org

Charlotte Van Petegem
Ghent University
Department WE02
Krijgslaan 281, S9
9000 Gent, Belgium
0000-0003-0779-4897
charlotte.vanpetegem@ugent.be

Jason Williams
Cold Spring Harbor Laboratory
1 Bungtown Rd.
Cold Spring Harbor, NY, 11724, USA
0000-0003-3049-2010
williams@cshl.edu

Craig Willis
University of Illinois at Urbana-Champaign
501 E. Daniel St.
Champaign, IL 61820, USA
0000-0002-6148-7196
willis8@illinois.edu

Nan Xiao
Seven Bridges Genomics
529 Main St, Suite 6610
Charlestown, MA 02129, USA
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 (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 eventually 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 (4BC, 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 with the name and arguments defined in the interface code. The arguments in a call to the function.
Figure 1: Design for an Interactive System. May 5, 1976
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:

\[ r \leftarrow \text{reg(myData[,1], myData[,2:4])} \]

Information from the regression object such as \( r\text{coef} \) and \( r\text{resid} \) 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

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 technologically 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 organi-
zations 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


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 467, 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.
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

\[
r1 \leftarrow \text{lm}( \, \text{runTime} \sim \text{age}, \, \text{data} = \text{cbMen})
\]

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. \( \log(\text{age}) \).

The object returned is from a class, "lm" in this case, for which functional methods simplify further study, such as specialized graphical displays:

\[
\text{plot(r1)}
\]

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 \( n \times p \) 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:

```r
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 (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


*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

$$\arg\min_x f(x)$$

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

$$g(x) = \frac{\partial f(x)}{\partial x}.$$  

The Hessian of \(f(x)\) is the matrix of second partial derivatives

$$H(x) = \frac{\partial^2 f(x)}{\partial x^2}.$$  

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 \(\delta\) 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 vmmin is called. However, the source for vmmin 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 vmmin. In src/appl/optim.c, above this routine is the comment

```c
/* 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 \( 3 \times n \)), 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 (1980), 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 behavior of nlmib() 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.


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 embodied were around for a decade or half, 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/lbldbgs/, 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


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

by Michał Burdukiewicz, Filip Pietluch, Jarosław Chilimoniuk, Katarzyna Sidorczuk, Dominik Rafacz, Leon Eyrich Jessen, Stefan Rödiger, Marcin Kosiński and Piotr Wójcik

Why R? conferences have been the hallmark of the Why R? Foundation (whyr.pl). Our goal has been to establish a series of international R-related events in Poland. After three years, we are happy to announce that our main event, the Why R? conference, has become one of the largest annual R conferences in Central Europe.

Why R? 2019 was the third part of Why R? conference event. After the last edition that was held in Wrocław (Burdukiewicz et al., 2018), our conference has returned to Warsaw. A total of approximately 300 people from 20 countries attended the main conference event. The event took place from 26th to 29th September 2019 and was co-organised by the Faculty of Economic Sciences of the University of Warsaw (wne.uw.edu.pl/en/), a leading academic institution in Poland, having important achievements in quantitative methods and data science. We received major support from ML in PL Society (mlinpl.org), a group of young researchers, aiming to promote machine learning events in Poland, who shared their resources and experience to make the conference more accessible.

For the first time, this year the conference featured a language-agnostic data visualizations hackathon (whyr.pl/2019/hackathon). Such an event gives the Why R? community a chance to exchange experience and inspirations with the users of any other languages and tools.

Participants

In spite of the fact that Why R? events are aimed at experienced data science practitioners, each conference gathers a high percentage of students (around 30%). Our participants have very diverse scientific backgrounds, where mathematics (mainly statistics) and computer science are the most common. All of them have jobs related to data science, including professional R developers (programmers), data engineers, machine learning practitioners...
and business analysts. One of the key advantages of Why R? is that it gathers participants both from academia and the industry.

Conference program

![Conference program diagram](image)

**Figure 2:** Why R? 2019 conference programme.

The format of the conference was aimed at exposing participants to recent developments in the R language as well as a wide range of application examples. The event consisted of workshops, invited keynote talks, field-specific series of talks, lightning-talks, special interest groups and a full-day data visualizations hackathon. It offered extensive networking opportunities. The welcome party was held at the conference venue on the first day of lectures. In addition, many informal gatherings were organised during each conference day, as the event took place close to the Old Town.

To sum up, Why R? 2019 consisted of: one day of hackathon (60 attendees), one day of workshops (150 attendees), one evening of round tables, two days of lectures (250 attendees) and one evening Welcome paRty (100 attendees). In 2019 we hosted a total of 315 unique attendees. During lectures there were carried out: 6 keynote talks, 42 regular talks and 14 lightning talks. Below you can find the conference agenda.
Materials from the conference are available on GitHub and YouTube:
- abstracts github.com/WhyR2019/abstracts,
- presentations github.com/WhyR2019/presentations
- videos whyr.pl/youtube/

Data Visualizations Hackathon

On the day before the conference we organized the free Data Visualizations Hackathon. It was a great opportunity for networking and exchange of experiences between data scientists that use different programming languages. The challenge was based on the data from Google Places API, which allows to search for places in a particular area. Thanks to this API we gathered data related to places in Warsaw, their working hours and occupancy. Based on this source of data participants, divided into 10 teams, were asked to prepare useful business application powered data visualizations solutions and techniques.

Pre-meetings

In 2019, Why R? 2019 was preceded by fourteen pre-meetings in eight countries. The purpose of those meetings was to provide the space for professional networking and knowledge exchange for practitioners and students, from the area of statistical machine learning, programming, optimization and data science. The Why R? Foundation supported organisation of pre-meetings financially and/or by sending speakers.

The organisation of pre-meetings would not be possible without the wonderful support of local R communities. Aside from the promotion of Why R? we had a great opportunity to interact with other R enthusiasts.
Figure 4: Locations and dates of the main Why R? 2019 conference and Why R?-branded pre-meetings.

Workshops

<table>
<thead>
<tr>
<th>Time</th>
<th>Room B</th>
<th>Room C</th>
<th>Room E</th>
<th>Room A103</th>
<th>Room A203</th>
</tr>
</thead>
<tbody>
<tr>
<td>08:00-09:00</td>
<td></td>
<td></td>
<td></td>
<td>Registration</td>
<td></td>
</tr>
<tr>
<td>09:00-10:30</td>
<td></td>
<td></td>
<td></td>
<td>mr3 and drake</td>
<td>Generalized Additive Models</td>
</tr>
<tr>
<td>10:30-11:00</td>
<td></td>
<td></td>
<td></td>
<td>data.table</td>
<td>Shiny basics</td>
</tr>
<tr>
<td>11:00-12:30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>auditor + DALEX</td>
</tr>
<tr>
<td>12:30-14:00</td>
<td></td>
<td></td>
<td></td>
<td>Lunch</td>
<td></td>
</tr>
<tr>
<td>14:00-15:30</td>
<td></td>
<td></td>
<td></td>
<td>Deep Learning in R (with Keras)</td>
<td>Speeding up R with C++ (Rcpp)</td>
</tr>
<tr>
<td>15:30-16:00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Basics of spatial data analysis</td>
</tr>
<tr>
<td>16:00-17:30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Explainable Artificial Intelligence in Business</td>
</tr>
<tr>
<td>17:30-18:00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Coffee Break</td>
</tr>
</tbody>
</table>

Figure 5: Why R? 2019 workshops.

Why R? 2019 conference had a wide portfolio of workshops that are listed below. One can find materials from workshops at this GitHub repository: [github.com/WhyR2019/workshops](https://github.com/WhyR2019/workshops)

- **Introduction to modern Generalized Additive Models in R (with mgcv)** by Matteo Fasiolo (University of Bristol). The assumption of the full-day workshop was firstly to give its participants some theoretical background about GAMs and some practical experience in R and finally to make attendees ready to start applying these models themselves. GAM models are a non-parametric extension of traditional regression model and were proved to be highly useful for both predictive and inferential purposes. Their popularity is based on a good balance between flexibility and interpretability as
well as on the possible application on large datasets. Matteo started with explanation of standard GAMs and related R packages. He explained what an additive model is, how the smooth effects and random effects are introduced. GAM models fitting was accompanied by the explanation of additional diagnostic and model selection tools, and Big Data GAM methods. In the end more recent developments were also described, i.e. quantile GAM models. The practical sessions were based on the mgcv (Wood, 2017), qgam (Fasiolo et al., 2017) and mgcViz (Fasiolo et al., 2018) packages.

- **datatable introduction & time-series** by Jan Gorecki (H2O.AI). The workshop was divided into two parts – the first part was devoted to the introduction of data.table query concept while the second focused on a particular use case of working with time-series data. In the first part Jan showed syntax similarities and differences between data.table and data.frame approaches. He used Arun Srinivasan workshops materials from useR!2017, with a few extras: chaining of data.table queries, reference semantics, subset of data: .SD and R function argument matching. In the second part Jan showed the application of efficient data processing on financial time series data of high-frequency (tick data quotations), including efficient aggregation to OHLC data, calculation of moving averages and using rolling join.

- **Straightforward introduction to Deep Learning in R (with Keras)** by Mikołaj Bogucki and Mikolaj Olszewski (iDash). The workshop started with the explanation of what Deep Learning and Neural Networks are (complex functions) and what components they include (input, output, hidden layers and weights). Then Keras (Allaire et al., 2018) was presented as a high level library allowing to build neural networks with an easy to use set of commands. The practical example using airBnB data and Keras R codes showed all stages of building a Neural Network: (1) defining the structure of the network, (2) defining the way of training (the loss function and the optimizer algorithm), (3) training (together with its visualization), (4) evaluation and (5) prediction. In addition, training, validation, test set division, simple imputation of missing data, using non-linear activation functions and basic feature engineering was shortly explained.

- **auditor + DALEX: a powerful duet for validation and explanation of machine learning models** by Alicja Gosiewska and Tomasz Mikolajczyk (MI2 Data Lab). The aim of the workshop was to familiarize participants with modern methods of model verification and exploration. In the first part Alicja and Tomasz introduced the idea of DALEX (Biecek, 2018) explainers, showing how to use them to assess the performance of a model and explain the model’s predictions (including global and local explanations). In the second part they focused on additional functionalities of the auditor (Gosiewska and Biecek, 2018) package, showing how the analysis of residuals may be applied to select the best model or even improve models.

- **Black is the new White - using eXplainable Artificial Intelligence in Business** by Marcin Chlebus (Faculty of Economic Sciences, University of Warsaw, Data Juice Lab, Data Donuts). Marcin presented XAI as a possible solution for understanding “Black-box” model complexity and fuzziness. He showed how XAI helps in stability and sensitivity analysis, prediction quality assessment and identification of decision drivers. The use cases showing the application of XAI in cross-sell marketing campaigns and risk management were presented. With the use of step by step analysis, it was shown that XAI is a set of tools enabling application of “black box” models in many business industries through in-depth understanding of advanced machine learning modelling.

- **Shiny Basics** by Theo Roe (Jumping Rivers). This workshop was intended as a quick introduction to creating interactive visualisations of data using shiny. Theo started with some basic examples of using markdown and htmlwidgets, then showed input and output bindings to interact with R data structures and using inputs to render output tables and graphs. In the end, Theo showed how to create own page layouts using shiny and shinydashboard and input and output "slots".
• **Speeding up R with C++ (Rcpp) – from basics to more advanced applications** by Piotr Wójcik (Faculty of Economic Sciences, University of Warsaw, Data Science Lab). Piotr discussed various aspects of Rcpp that helps to easily replace the R code with often significantly faster counterparts in C++. Writing R functions in C++ was explained, starting from simple examples with the focus on similarities and differences between R and C++ syntax. Then Piotr at first explained writing loops and recursive calls in C++, using Rcpp sugar and secondly presented how to store C++ code in *.cpp files, using Standard Template Library, iterators, algorithms and range-based loops. In the end complex input/output objects (S3 and S4) were discussed.

• **Machine Learning Pipelines and Reproducible Research with mlr3 and drake** by Jakob Richter (TU Dortmund University) and Patrick Schratz (LMU Munich). The workshop was divided into two parts – the former introduced the new mlr3 package (Lang et al., 2019) framework (the successor of the mlr package) while the latter presented a brief overview of the drake package (Landau, 2018) in R. In the first part Jakob and Patrick explained the philosophy and ingredients of mlr3 package. They presented how to define the data and the target variable, using learners provided by mlr3, set and tune hyperparameters, make predictions and evaluate their performance, including resampling techniques and comparing multiple learners. The practical example showed hyperparameter tuning and training of a random forest classifier on the iris dataset. The practical part also involved benchmark analysis of multiple learners, using different hyperparameter ranges on the iris and spam datasets. A particular emphasis was put on machine learning workflows that might be easily controlled with mlr3pipelines package (Binder et al., 2019). In the second part the drake package was presented. It helps to set up a reproducible workflow of the project and it easily integrates with the mlr3 package and its extensions.

• **Basics of spatial data analysis** by Jakub Nowosad (Adam Mickiewicz University, Poznan). The emphasis in this workshop was put on getting started with spatial data analysis. Jakub demonstrated key packages for spatial analysis and making maps, explained spatial data representation in R, using sf (Pebesma, 2018), for spatial vector data, and raster (Hijmans et al., 2017) packages. Then he gave a lot of examples of spatial data visualization, using a powerful tmap package (Tennekes, 2018), including some vector-raster interactions. In the end, he also showed data manipulation examples with the tidyverse (dplyr) approach, in which sf spatial objects are simply special data frames.

**Invited talks**

The invited talks topics included domain knowledge from statistics, computer science, natural sciences and economics. The speakers list presents as follows:

**Marvin Wright**

Random forests used to be everywhere, from Microsoft Kinect to meteorology, but their popularity considerably dropped with the advent of deep learning. During his keynote talk at Why R? 2019 Marvin R. Wright has shown that random forests still can be used in machine learning routines, making the whole process time- and cost-efficient.

Implementing a real-life machine learning solution is not only about the best performance. Marvin has shown that considering trade-off between performance and costs of the analysis, random forests are still unbeatable. Aside from the methodological background, Marvin has given an overview of random forest implementations in R (Wright and Ziegler, 2017).

Marvin is a Postdoc at the Leibniz Institute for Prevention Research and Epidemiology in Bremen, Germany. He is the author of several R packages, including the fastest implementation of random forest in R, ranger. He holds a Ph.D. in Biostatistics from the University
of Lübeck, supervised by Andreas Ziegler. In the past, Marvin worked at the University of Lübeck. He was a visiting researcher at the University of Copenhagen. Also, he spent some time in the automotive and health insurance industries. His main research interests are interpretable machine learning, genetic epidemiology and survival analysis.

**Jakub Nowosad**

Jakub Nowosad’s keynote lecture was a great opportunity to learn about geostatistics. Jakub, a co-author of the Geocomputation with R (Lovelace et al., 2019), has focused on tools used to solve real-life problems in spatial data analysis.

The growing importance of spatial data stimulates a rapid evolution of geostatistical methods. Jakub, as the active member of #spatial community, not only presented cutting-edge tools but also gave his unique insight into the future of the spatial data analysis.

Jakub is an assistant professor in the Department of Geoinformation at the Adam Mickiewicz University in Poznan, Poland. His main research is focused on developing and applying spatial methods in order to expand our understanding of processes and patterns in the environment. He has extensive teaching experience in the fields of spatial analysis, geostatistics, statistics, and machine learning.

**Sigrid Keydana**

We know how accurate are our predictions but do we really know how certain they are? This question has been answered by Sigrid Keydana (RStudio) during her keynote lecture.

Sigrid has presented tfprobability, an interface to TensorFlow Probability, a tool for obtaining uncertainty estimates from deep neural networks. This exciting tool can be extended beyond a classic deep learning framework into complex hierarchical models.

Sigrid is an Applied Researcher at RStudio. She has experience as a psychologist, software developer and data scientist. She is passionate about exploring the borders of deep learning, especially by helping users to apply the power of deep learning in R.

**Steph Locke**

Machine learning models find their place in almost every area of our life, influencing things as small as the video recommendations on YouTube or as big as the length and severity of a sentence in a criminal procedure. With the growing importance of machine learning, it becomes more and more important to train models while keeping in mind their ethical consequences.

During her keynote talk at Why R? 2019, Steph Locke showed us ethical concerns about data science. Apart from pointing out existing issues, she has also presented solutions leading to more fair and transparent machine learning models.

Steph is the founder of a consultancy in the UK. Her talks, blog posts, conferences, and business all have one thing in common – they help people get started with data science. Steph holds the Microsoft MVP award for her community contributions. In her spare time, Steph plays board games with her husband and takes copious pictures of her doggos.

**Wit Jakuczun**

Wit Jakuczun from WLOG Solutions presented his talk about deploying - How to make R great for machine learning in (not only) Enterprise.

For many years software engineers have put enormous effort to develop best practices to deliver stable and maintainable software. How R users can benefit from this experience? Wit answered this question by going through several concepts and tools that are natural for software engineers but are often undervalued by R users.
Paula Brito

During her keynote lecture at Why R? 2019 Paula Brito has given a unique insight into the world of symbolic data, where data points are represented not as single values, but more complex structures, like sets or intervals (Noirhomme-Fraiture and Brito, 2011).

A classical paradigm of data science assumes that categorical variables, like gender or educational stage, are represented as the single value per observation. Paula has shown how to utilize her package, MAINT.Data, to model interval data, using its symbolic representation which leads to more accurate and robust models.

Paula is Associate Professor at the Faculty of Economics of the University of Porto, and member of the Artificial Intelligence and Decision Support Research Group (LIAAD) of INESC TEC, Portugal. Her current research focuses on the analysis of multidimensional complex data, known as symbolic data, for which she develops statistical approaches and multivariate analysis methodologies.

Round tables

Round tables are networking-oriented social mixers devoted to connecting people with similar interests. The exact points discussed during the round table and its style depend on the moderators who are shaping out the details, based on the general agenda provided by the Why R? organizers. The organizing committee both selects the topics of round tables and invites appropriate moderators.

Diversity in Data Science

This board aims to inspire members of affinity groups to pursue careers in data science. We hope that this platform for networking will reduce the diversity of R community. Moderator: Barbara Sobkowiak (Women in Machine Learning & Data Science Poland).

Career-planning in Data Science

Participants of WhyR will have a chance to learn from more experienced R enthusiasts about their career paths. Moderator: Kamil Kosiński (PwC).

Teaching Data Science

Practitioners will share their experiences in introducing their students to basic and advanced concepts of data science. Moderator: Patrick Schratz (Ludwig Maximilian University of Munich).

Data Visualizations

Discuss data visualizations good practices and approaches to various presentation challenges. Moderator: Michał Burdukiewicz (Warsaw University of Technology).

Ethics in Data Science

With the increased importance of machine learning, we are becoming more and more concerned about the ethics of data science. Moderator: Steph Locke (Locke Data).
Conference organizers

The organizing committee consisted of Klaudia Korniluk, Marcin Kosiński, Michał Burdukiewicz, Jarosław Chilimoniuk, Katarzyna Sidorczuk, Filip Pietluch, Weronika Puchała and Dominik Rafacz.

The quality of the scientific program of the conference was the achievement of Stefan Rödiger (Brandenburg University of Technology Cottbus-Senftenberg), Piotr Wójcik (University of Warsaw) and Bernd Bischl (Ludwig Maximilian University of Munich).

Acknowledgements

We would like to express our gratitude to all our sponsors, the Faculty of Economic Sciences (University of Warsaw), ML in PL Society, local organizers of the pre-meetings and student helpers.

Additional information


Bibliography


Michał Burdukiewicz  
Warsaw University of Technology, Why R? Foundation  
Pl. Politechniki 1, 00-661 Warsaw  
Poland  
michal@why.r.pl

Filip Pietluch  
University of Wrocław  
Pl. Uniwersytecki 1, 50-137 Wrocław  
Poland  
fpietluch@gmail.com

Jarosław Chilimoniuk  
University of Wrocław  
Pl. Uniwersytecki 1, 50-137 Wrocław  
Poland  
jaroslaw.chilimoniuk@gmail.com

Katarzyna Sidorczuk  
University of Wrocław  
Pl. Uniwersytecki 1, 50-137 Wrocław  
Poland  
sidorczuk.katarzyna17@gmail.com

Dominik Rafacz  
Warsaw University of Technology  
Pl. Politechniki 1, 00-661 Warsaw  
Poland  
dominikrafacz@gmail.com
Leon Eyrich Jessen  
Technical University of Denmark  
Anker Engelunds Vej 1, 2800 Kgs. Lyngby, Denmark  
Denmark  
ljess@dtu.dk

Stefan Rödiger  
Brandenburg University of Technology Cottbus–Senftenberg  
Universitätsplatz 1, Senftenberg  
Germany  
ORCiD: 0000-0002-1441-6512  
stefan.roediger@b-tu.de

Marcin Kosiński  
Gradient Metrics LLC, Why R? Foundation  
Warsaw  
Poland  
marcin@whyr.pl

Piotr Wójcik  
University of Warsaw, Data Science Lab  
ul. Długa 44/50, 00-241 Warsaw  
Poland  
pwojcik@wne.uw.edu.pl
Changes on CRAN
2020-01-01 to 2020-08-31

by Kurt Hornik, Uwe Ligges and Achim Zeileis

In the past 8 months, 1554 new packages were added to the CRAN package repository. 96 packages were unarchived and 843 were archived. The following shows the growth of the number of active packages in the CRAN package repository:

On 2020-08-31, the number of active packages was around 16174.

Changes in the CRAN Repository Policy

The Policy now says the following:

- All correspondence with CRAN must be sent to CRAN-submissions@R-project.org (for submissions) or CRAN@R-project.org (for published packages) and not to members of the team, in plain text ASCII and not HTML.

- A package listed in ‘Suggests’ or ‘Enhances’ should be used conditionally in examples or tests if it cannot straightforwardly be installed on the major R platforms. (‘Writing R Extensions’ recommends that they are always used conditionally.)

- Orphaned CRAN packages should not be strict requirements (in the ‘Depends’, ‘Imports’ or ‘LinkingTo’ fields, including indirectly). They are allowed in ‘Suggests’ if used conditionally, although this is discouraged.

- Packages which use Internet resources should fail gracefully with an informative message if the resource is not available or has changed (and not give a check warning nor error).

- Compiled code should never terminate the R process within which it is running. Thus C/C++ calls to assert/abort/exit/std::terminate, Fortran calls to STOP and so on must be avoided.
• Updates to previously-published packages must have an increased version. Increasing the version number at each submission reduces confusion so is preferred even when a previous submission was not accepted.

CRAN package submissions

During the first 8 months of 2020 (January to August), CRAN received 22020 package submissions. For these, 38551 actions took place of which 25063 (65%) were auto processed actions and 13488 (35%) manual actions.

Minus some special cases, a summary of the auto-processed and manually triggered actions follows:

<table>
<thead>
<tr>
<th></th>
<th>archive</th>
<th>inspect</th>
<th>newbies</th>
<th>pending</th>
<th>pretest</th>
<th>publish</th>
<th>recheck</th>
<th>waiting</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto</td>
<td>5710</td>
<td>4300</td>
<td>5269</td>
<td>0</td>
<td>0</td>
<td>6216</td>
<td>1987</td>
<td>1581</td>
</tr>
<tr>
<td>manual</td>
<td>5437</td>
<td>95</td>
<td>936</td>
<td>783</td>
<td>291</td>
<td>4532</td>
<td>1181</td>
<td>233</td>
</tr>
</tbody>
</table>

These include the final decisions for the submissions which were

<table>
<thead>
<tr>
<th>action</th>
<th>archive</th>
<th>publish</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto</td>
<td>5401 (25.2%)</td>
<td>5258 (24.5%)</td>
</tr>
<tr>
<td>manual</td>
<td>5315 (24.8%)</td>
<td>5449 (25.4%)</td>
</tr>
</tbody>
</table>

where we only count those as auto processed whose publication or rejection happened automatically in all steps.

The CRAN team has changed. Martina Schmir and Jelena Sef left the team. Thanks a lot to both of you! New members are Gregor Seyer who is very actively processing newbies submissions and Julia Haider who just joined the team. Welcome to CRAN!

CRAN mirror security

Currently, there are 101 official CRAN mirrors, 65 of which provide both secure downloads via ‘https’ and use secure mirroring from the CRAN master (via rsync through ssh tunnels). Since the R 3.4.0 release, chooseCRANmirror() offers these mirrors in preference to the others which are not fully secured (yet).

New packages in CRAN task views

Bayesian  BEST, BVAR, BayesPostEst, Bergm, NGSSEML, acebayes, bbricks, conting, mcmcse, stableGR.
ChemPhys  spectrino.
Cluster  DatabionicSwarm, ProjectionBasedClustering, genieclust.
Databases  RClickhouse, dbx, dplyr, sparklyr.
Econometrics  REndo, collapse, fixest, mfx, mhurdle, mnlogit, skedastic.
Environmetrics  PMCMRplus, dsm, rioja.
Finance  copulaData, nvmix, qrmdata, qrmtools.
FunctionalData  fdANOVA, fdaACF.
HighPerformanceComputing  pbdBASE.
Hydrology  AWAPer, RNCRS, climate, fasstr, metScanR, stationaRy.
MetaAnalysis  NMADiagT, SPAtest, getspres, metagam, metapower, metarep, metawho, miniMeta, poolr, publibha.

MissingData  CircSpaceTime, ClustImpute, ECLRMC, EditImputeCont, FSMUMI, IPWboxplot, NPBayesImputeCat, RBtest, RMixtComp, StemCens, areal, biclustermd, bootImpute, cassandRa, iai, impimp, imputeFin, imputeR, isotree, lodi, metasens, miWQS, miceRanger, mipred, misaem, missSBM, missingHE, naïvebayes, plsRbeta, psfmi, robustrank, rrcovNA, rsparse, sievePH, tensorBF, ui.

ModelDeployment  RestRserve.

NaturalLanguageProcessing  BTM, LexisNexisTools, corporaexplorer, crfsuite, ruimtehol, textplot, tokenizers.bpe, topicdoc.

NumericalMathematics  Carlson, JuliaConnectoR, RcppBigIntAlgos, caracas, clifford, dual, polyMatrix, rmatio, symengine.

OfficialStatistics  cancensus, cansim, collapse.

Optimization  mixsqp.

Psychometrics  BGGM, EGAnet, IsingFit, IsingSampler, NetworkComparisonTest, SemNet, TestDesign, bootnet, edina, edmdata, elasticlsing, errum, gimme, glasso, graphicalVAR, iarm, irplay, lvnet, mgm, mlVAR, networktools, networktree, thurstonianIRT.

Robust  rrcovNA.

Spatial  cancensus, gear.

TeachingStatistics  arm, msos, wooldridge.

TimeSeries  BGVAR, BMTAR, BayesARIMAX, DTSg, EBMAforecast, NGsseML, ProbReco, UComp, bootUR, changepoint.geo, collapse, data.table, disaggR, fable.prophet, fabletools, fdaACF, fsMTS, garma, gratis, gravitas, mbsts, mixAR, pcts, rhosa, runner, scoringutils, seer, slider, smoots, statespacer, testcorr.

Tracking  rerddapXtracto.

WebTechnologies  dash, rromeo.

(* = core package)

Kurt Hornik
WU Wirtschaftsuniversität Wien, Austria
Kurt.Hornik@R-project.org

Uwe Ligges
TU Dortmund, Germany
Uwe.Ligges@R-project.org

Achim Zeileis
Universität Innsbruck, Austria
Achim.Zeileis@R-project.org
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) 田中明彦 (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) Alistair Cullum (United States) Ajit de Silva (United States) Dubravko Dolc (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) Seungdoo Lee (Korea, Republic of) Bernhard Lehner (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 Oehlschlägler (Germany) Jaesung James Park (Korea, Republic of) Matt Parker (United States) Bill Pikounis (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