The R Journal

Volume 13/1, June 2021

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

Contents

News and Notes

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.

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

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 Dianne Cook

On behalf of the editorial board, I am pleased to present Volume 13 Issue 1 of the R Journal.

First, some news about the journal board. Welcome to Gavin Simpson, who joins as a new Executive Editor! In addition, welcome to our new Associate Editors Nicholas Tierney, Isabella Gollini, Rasmus Bååth, Mark van der Loo, Elizabeth Sweeney, Louis Aslett and Katarina Domijan. With the large volume of submissions, the Associate Editors now play a vital role in processing articles.

There are some new developments in the journal operations under way. We are working on a new package **rjtools** which will operate a little like the **devtools** package and help you to create a new article from a template, and check that it conforms to the style and requirements of the R Journal.

We are also working on supporting articles written in RMarkdown, which will be rendered in html through a modified **distill** web site. The exciting feature is that interactive graphics could be included directly in the article. You can see how this current issue would look in the new style at https://rjournal.r-project.org/dev. Particularly, look at articles Conversations in Time by Wang and Cook as an example that has two examples of how interactive graphics might be included. Other articles rendered in html are "Finding Optimal Normalizing Transformations" by Peterson, "Automating Reproducible, Collaborative Clinical Trial Document Generation" by Kane, Jiang and Urbanek, and "Towards a Grammar for Processing Clinical Trial Data" by Kane. All remaining articles in the new site style are the current pdf style.

To experiment with creating a new article, or to check that your article, conforms with the R Journal author guidelines, go to https://rjournal.github.io/rjtools/. Note that it is still ok to use the **rticles** package R Journal Rmarkdown template to create your article. This will generate the files that are compiled to pdf using latex, but it is an easy translation for us to convert them into the new style.

The operational support and the experiments have been supported with generous funding from the R Consortium (https://www.r-consortium.org).

Behind the scenes, several people are assisting with the journal operations and the new developments. Mitchell O'Hara-Wild has worked on infrastructure, the new article submission system, a new issue build system and now the new article delivery system providing html format. H. Sherry Zhang has taken over from Stephanie Kobakian, in developing the **rjtools** package including check functions for new articles to help authors get the style constraints correct. In addition, articles in this issue have been painstakingly copy edited by Dewi Amaliah.

In this issue

News from the R Core, CRAN, Bioconductor, the R Foundation, and the foRwards Taskforce are included in this issue along with a summary of activities at the R Medicine and Why R? 2021 conferences.

This issue features 37 contributed research articles covering these topics:

- Multivariate analysis
  - **SeedCCA**: An integrated R-package for Canonical Correlation Analysis and Partial Least Squares
  - Unidimensional and Multidimensional Methods for Recurrence Quantification Analysis with **crqa**
  - **clustcurv**: An R Package for Determining Groups in Multiple Curves

  - **gofCopula**: Goodness-of-Fit Tests for Copulae
  - **ROCnReg**: An R Package for Receiver Operating Characteristic Curve Inference With and Without Covariates

- Non-parametric methods
  - **npcure**: An R Package for Nonparametric Inference in Mixture Cure Models
  - ROBustness In Network (**robin**): an R package for Comparison and Validation of Communities
  - **krippendorffsalpha**: An R Package for Measuring Agreement Using Krippendorff's Alpha Coefficient

- Temporal and longitudinal methods
  - **JMcmprsk**: An R Package for Joint Modelling of Longitudinal and Survival Data with Competing Risks
  - Linear Regression with Stationary Errors: the R Package **slm**
  - **penPHcure**: Variable Selection in Proportional Hazards Cure Model with Time-Varying Covariates
  - **pdynmc**: A Package for Estimating Linear Dynamic Panel Data Models Based on Nonlinear Moment Conditions
  - **DChaos**: An R Package for Chaotic Time Series Analysis
  - **IndexNumber**: An R Package for Measuring the Evolution of Magnitudes
  - **garchx**: Flexible and Robust GARCH-X Modelling
  - Working with CRSP/COMPUSTAT in R: Reproducible Empirical Asset Pricing
  - Analysing Dependence Between Point Processes in Time Using **IndTestPP**
  - Conversations in Time: Interactive Visualisation to Explore Structured Temporal Data

- Computing infrastructure
  - A Method for Deriving Information from Running R Code
  - Wide-to-tall Data Reshaping Using Regular Expressions and the **nc** Package
  - The **bdpar** Package: Big Data Pipelining Architecture for R
  - Benchmarking R packages for calculation of Persistent Homology
  - **distr6**: R6 Object-Oriented Probability Distributions Interface in R
  - Automating Reproducible, Collaborative Clinical Trial Document Generation
  - Reproducible Summary Tables with the **gtsummary** Package
  - Towards a Grammar for Processing Clinical Trial Data

- Simulation and optimisation
  - Finding Optimal Normalizing Transformations via **bestNormalize**
  - Package **wsbackfit** for Smooth Backfitting Estimation of Generalized Structured Models
  - **RLumCarlo**: Simulating Cold Light using Monte Carlo Methods
  - **OneStep**: Le Cam's one-step estimation procedure
  - The **HBV.IANIGLA** Hydrological Model
  - Regularized Transformation Models: The **tramnet** Package

- Other topics

- **exPrior**: An R Package for the Formulation of Ex-Situ Priors
- **BayesSPsurv**: An R Package to Estimate Bayesian (Spatial) Split-Population Survival Models
- Statistical Quality Control with the **qcr** Package
- The R Package **smicd**: Statistical Methods for Interval-Censored Data
- **stratamatch**: Prognostic Score Stratification Using a Pilot Design

Happy reading, and code testing!

Dianne Cook
Monash University

https://journal.r-project.org
r-journal@r-project.org

SEEDCCA: An Integrated R-Package for Canonical Correlation Analysis and Partial Least Squares

by Bo-Young Kim, Yunju Im and Jae Keun Yoo

Abstract Canonical correlation analysis (CCA) has a long history as an explanatory statistical method in high-dimensional data analysis and has been successfully applied in many scientific fields such as chemometrics, pattern recognition, genomic sequence analysis, and so on. The so-called **seedCCA** is a newly developed **R** package that implements not only the standard and seeded CCA but also partial least squares. The package enables us to fit CCA to large-p and small-n data. The paper provides a complete guide. Also, the seeded CCA application results are compared with the regularized CCA in the existing **R** package. It is believed that the package, along with the paper, will contribute to high-dimensional data analysis in various science field practitioners and that the statistical methodologies in multivariate analysis become more fruitful.

Introduction

Explanatory studies are important to identify patterns and special structures in data prior to developing a specific model. When a study between two sets of a p-dimensional random variables $\mathbf{X}$ ($\mathbf{X} \in \mathbb{R}^p$) and an r-dimensional random variable $\mathbf{Y}$ ($\mathbf{Y} \in \mathbb{R}^r$), are of primary interest, one of the popular explanatory statistical methods would be canonical correlation analysis (CCA; Hotelling (1936)). The main goal of CCA is the dimension reduction of two sets of variables by measuring an association between the two sets. For this, pairs of linear combinations of variables are constructed by maximizing the Pearson correlation. The CCA has successful application in many scientific fields such as chemometrics, pattern recognition, genomic sequence analysis, and so on.

In Lee and Yoo (2014), it is shown that the CCA can be used as a dimension reduction tool for high-dimensional data, but also it is connected to the least square estimator. Therefore, the CCA is not only an explanatory and dimension reduction method but also can be utilized as an alternative to least square estimation.

If $\max(p, r)$ is bigger than or equal to the sample size, n, usual CCA application is not plausible due to no incapability of inverting sample covariance matrices. To overcome this, a regularized CCA is developed by Leurgans et al. (1993), whose idea was firstly suggested in Vinod (1976). In practice, the **CCA** package by González et al. (2008) can implement a version of the regularized CCA. To make the sample covariance matrices saying $\hat{\mathbf{\Sigma}}_x$ and $\hat{\mathbf{\Sigma}}_y$, invertible, in González et al. (2008), they are replaced with

$$\hat{\mathbf{\Sigma}}_x^{\lambda_1} = \hat{\mathbf{\Sigma}}_x + \lambda_1 \mathbf{I}_p \text{ and } \hat{\mathbf{\Sigma}}_y^{\lambda_2} = \hat{\mathbf{\Sigma}}_y + \lambda_1 \mathbf{I}_r.$$

The optimal values of λ_1 and λ_2 are chosen by maximizing a cross-validation score throughout the two-dimensional grid search. Although it is discussed that a relatively small grid of reasonable values for λ_1 and λ_2 can lesson intensive computing in González et al. (2008), it is still time-consuming as observed in later sections. Additionally, fast regularized CCA and robust CCA via projection-pursuit are recently developed in Cruz-Cano (2012) and Alfons et al. (2016), respectively.

Another version of CCA to handle $\max(p, r) > n$ is the so-called seeded canonical correlation analysis proposed by Im et al. (2014). Since the seeded CCA does not require any regularization procedure, which is computationally intensive, its implementation to larger data is quite fast. The seeded CCA requires two steps. In the initial step, a set of variables bigger than n is initially reduced based on iterative projections. In the next step, the standard CCA is applied to two sets of variables acquired from the initial step to finalize the CCA of data. Another advantage is that the procedure of the seeded CCA has a close relation with partial least square, which is one of the popular statistical methods for large p-small n data. Thus the seed CCA can yield the PLS estimates.

The **seedCCA** package is recently developed mainly to implement the seeded CCA. However, the package can fit a collection of the statistical methodologies, which are standard canonical correlation and partial least squares with uni/multi-dimensional responses, including the seeded CCA. The package is already uploaded to CRAN (`https://cran.r-project.org/web/packages/seedCCA/index.html`).

The main goal of the paper is to introduce and illustrate the **seedCCA** package. Accordingly, three real data are fitted by the standard CCA, the seeded CCA, and partial least square. Two of the three data are available in the package. One of them has been analyzed in González et al. (2008). So, the

implementation results by the seeded and regularized CCA are closely compared.

The organization of the paper is as follows. The collection of three methodologies is discussed in Section 2. The implementation of **seedCCA** is illustrated, and compared with **CCA** in Section 3. In Section 4, we summarize the work.

We will use the following notations throughout the rest of the paper. A p-dimensional random variable $\mathbf{X}$ will be denoted as $\mathbf{X} \in \mathbb{R}^p$. So, $\mathbf{X} \in \mathbb{R}^p$ means a random variable, although there is no specific mention. For $\mathbf{X} \in \mathbb{R}^p$ and $\mathbf{Y} \in \mathbb{R}^r$, we define that $\text{cov}(\mathbf{X}) = \mathbf{\Sigma}_x$, $\text{cov}(\mathbf{Y}) = \mathbf{\Sigma}_y$, $\text{cov}(\mathbf{X}, \mathbf{Y}) = \mathbf{\Sigma}_{xy}$ and $\text{cov}(\mathbf{Y}, \mathbf{X}) = \mathbf{\Sigma}_{yx}$. Moreover, it is assumed that $\mathbf{\Sigma}_x$ and $\mathbf{\Sigma}_y$ are positive-definite.

Collection of implemented methodologies in seedCCA

Canonical correlation analysis

Suppose the two sets of variable $\mathbf{X} \in \mathbb{R}^p$ and $\mathbf{Y} \in \mathbb{R}^r$ and consider their linear combinations of $U = \mathbf{a}^\mathrm{T}\mathbf{X}$ and $V = \mathbf{b}^\mathrm{T}\mathbf{Y}$. Then we have $\text{var}(U) = \mathbf{a}^\mathrm{T}\mathbf{\Sigma}_x\mathbf{a}$, $\text{var}(V) = \mathbf{b}^\mathrm{T}\mathbf{\Sigma}_y\mathbf{b}$, and $\text{cov}(U, V) = \mathbf{a}^\mathrm{T}\mathbf{\Sigma}_{xy}\mathbf{b}$, where $\mathbf{a} \in \mathbb{R}^{p\times 1}$ and $\mathbf{b} \in \mathbb{R}^{r\times 1}$. Then Pearson-correlation between U and V is as follows:

$$\text{cor}(U, V) = \frac{\mathbf{a}^\mathrm{T}\mathbf{\Sigma}_{xy}\mathbf{b}}{\sqrt{\mathbf{a}^\mathrm{T}\mathbf{\Sigma}_x\mathbf{a}}\sqrt{\mathbf{b}^\mathrm{T}\mathbf{\Sigma}_y\mathbf{b}}}. \tag{1}$$

We seek to find $\mathbf{a}$ and $\mathbf{b}$ to maximize $\text{cor}(U, V)$ by satisfying the following criteria.

1. The first canonical variate pair ($U_1 = \mathbf{a}_1^\mathrm{T}\mathbf{X}, V_1 = \mathbf{b}_1^\mathrm{T}\mathbf{Y}$) is obtained from maximizing (1).
2. The second canonical variate pair ($U_2 = \mathbf{a}_2^\mathrm{T}\mathbf{X}, V_2 = \mathbf{b}_2^\mathrm{T}\mathbf{Y}$) is constructed from the maximization of (1) with restriction that $\text{var}(U_2) = \text{var}(V_2) = 1$ and (U_1, V_1) and (U_2, V_2) are uncorrelated.
3. At the k step, the kth canonical variate pair ($U_k = \mathbf{a}_k^\mathrm{T}\mathbf{X}, V_k = \mathbf{b}_k^\mathrm{T}\mathbf{Y}$) is obtained from the maximization of (1) with restriction that $\text{var}(U_k) = \text{var}(V_k) = 1$ and (U_k, V_k) are uncorrelated with the previous $(k-1)$ canonical variate pairs.
4. Repeat Steps 1 to 3 until k becomes q $(= \min(p, r))$.
5. Select the first d pairs of (U_k, V_k) to represent the relationship between $\mathbf{X}$ and $\mathbf{Y}$.

Under this criteria, the pairs $(\mathbf{a}_i, \mathbf{b}_i)$ are constructed as follows: $\mathbf{a}_i = \mathbf{\Sigma}_x^{-1/2}\psi_i$ and $\mathbf{b}_i = \mathbf{\Sigma}_y^{-1/2}\phi_i$ for $i = 1, \ldots, q$, where $(\psi_1, ..., \psi_q)$ and $(\phi_1, ..., \phi_q)$ are, respectively, the q eigenvectors of $\mathbf{\Sigma}_x^{-1/2}\mathbf{\Sigma}_{xy}\mathbf{\Sigma}_y^{-1}\mathbf{\Sigma}_{yx}\mathbf{\Sigma}_y^{-1/2}$ and $\mathbf{\Sigma}_y^{-1/2}\mathbf{\Sigma}_{yx}\mathbf{\Sigma}_x^{-1}\mathbf{\Sigma}_{xy}\mathbf{\Sigma}_x^{-1/2}$ with the corresponding common ordered-eigenvalues of $\rho_1^{*2} \geq \cdots \geq \rho_q^{*2} \geq 0$. Then, matrices of $\mathbf{M}_x = (\mathbf{a}_1, ..., \mathbf{a}_d)$ and $\mathbf{M}_y = (\mathbf{b}_1, ..., \mathbf{b}_d)$ are called *canonical coefficient matrices* for $d = 1, ..., q$. Also, $\mathbf{M}_x^\mathrm{T}\mathbf{X}$ and $\mathbf{M}_y^\mathrm{T}\mathbf{Y}$ are called *canonical variates*. In the sample, the population quantities are replaced with their usual moment estimators. For more details regarding this standard CCA, readers may refer to Johnson and Wichern (2007).

Seeded canonical correlation analysis

Since the standard CCA application requires the inversion of $\hat{\mathbf{\Sigma}}_x$ and $\hat{\mathbf{\Sigma}}_y$ in practice, it is not plausible for high-dimensional data with $\max(p, r) > n$. In Im et al. (2014), a seeded canonical correlation analysis approach is proposed to overcome this deficit. The seeded CCA is a two-step procedure consisting of initialized and finalized steps. In the initialized step, the original two sets of variables are reduced to m-dimensional pairs without loss of information on the CCA application. In the initialized step, it is essential to force $m << n$. In the finalized step, the standard CCA is implemented to the initially-reduced pairs for the repairing and orthonormality. A more detailed discussion on the seeded CCA is as follows in the next subsections.

Development

Define a notation of $\mathcal{S}(\mathbf{M})$ as the subspace spanned by the columns of $\mathbf{M} \in \mathbb{R}^{p\times r}$. Lee and Yoo (2014) show the following relation:

$$\mathcal{S}(\mathbf{M}_x) = \mathcal{S}(\mathbf{\Sigma}_x^{-1}\mathbf{\Sigma}_{xy}) \text{ and } \mathcal{S}(\mathbf{M}_y) = \mathcal{S}(\mathbf{\Sigma}_y^{-1}\mathbf{\Sigma}_{yx}). \tag{2}$$

The relation in (2) directly indicates that $\mathbf{M}_x$ and $\mathbf{M}_y$ form basis matrices of $\mathcal{S}(\mathbf{\Sigma}_x^{-1}\mathbf{\Sigma}_{xy})$ and $\mathcal{S}(\mathbf{\Sigma}_y^{-1}\mathbf{\Sigma}_{yx})$ and that $\mathbf{M}_x$ and $\mathbf{M}_y$ can be restored from $\mathbf{\Sigma}_x^{-1}\mathbf{\Sigma}_{xy}$ and $\mathbf{\Sigma}_y^{-1}\mathbf{\Sigma}_{yx}$.

Now, we define the following two matrices:

$$\begin{aligned} \mathbf{R}_{x,u_1} \in \mathbb{R}^{p\times ru_1} &= (\mathbf{\Sigma}_{xy}, \mathbf{\Sigma}_x\mathbf{\Sigma}_{xy}, \ldots, \mathbf{\Sigma}_x^{u_1-1}\mathbf{\Sigma}_{xy}) \text{ and} \\ \mathbf{R}_{y,u_2} \in \mathbb{R}^{r\times pu_2} &= (\mathbf{\Sigma}_{yx}, \mathbf{\Sigma}_y\mathbf{\Sigma}_{yx}, \ldots, \mathbf{\Sigma}_y^{u_2-1}\mathbf{\Sigma}_{yx}). \end{aligned} \tag{3}$$

In $\mathbf{R}_{x,u_1}$ and $\mathbf{R}_{y,u_2}$, the numbers of u_1 and u_2 are called termination indexes. They decide the number of projections of $\mathbf{\Sigma}_{xy}$ and $\mathbf{\Sigma}_{yx}$ onto $\mathbf{\Sigma}_x$ and $\mathbf{\Sigma}_y$, respectively. Also define that

$$\begin{aligned} \mathbf{M}^0_{x,u_1} \in \mathbb{R}^{p\times r} &= \mathbf{R}_{x,u_1}(\mathbf{R}_{x,u_1}^{\mathrm{T}}\mathbf{\Sigma}_x\mathbf{R}_{x,u_1})^{-1}\mathbf{R}_{x,u_1}^{\mathrm{T}}\mathbf{\Sigma}_{xy} \text{ and} \\ \mathbf{M}^0_{y,u_2} \in \mathbb{R}^{r\times p} &= \mathbf{R}_{y,u_2}(\mathbf{R}_{y,u_2}^{\mathrm{T}}\mathbf{\Sigma}_y\mathbf{R}_{y,u_2})^{-1}\mathbf{R}_{y,u_2}^{\mathrm{T}}\mathbf{\Sigma}_{yx}. \end{aligned} \tag{4}$$

In Cook et al. (2007), it is shown that $\mathcal{S}(\mathbf{M}^0_{x,u_1}) = \mathcal{S}(\mathbf{\Sigma}_x^{-1}\mathbf{\Sigma}_{xy})$ and $\mathcal{S}(\mathbf{M}^0_{y,u_2}) = \mathcal{S}(\mathbf{\Sigma}_y^{-1}\mathbf{\Sigma}_{yx})$ in (4). Hence $\mathbf{M}^0_{x,u_1}$ and $\mathbf{M}^0_{y,u_2}$ can be used to infer $\mathbf{M}_x$ and $\mathbf{M}_y$, respectively. One clear advantage to use $\mathbf{M}^0_{x,u_1}$ and $\mathbf{M}^0_{y,u_2}$ is no need of the inversion of $\mathbf{\Sigma}_x$ and $\mathbf{\Sigma}_y$.

Practically, it is important to select proper values for the termination indexes u_1 and u_2 as they define that $\Delta_{x,u_1} = \mathbf{M}^0_{x,u_1+1} - \mathbf{M}^0_{x,u_1}$ and $\Delta_{y,u_2} = \mathbf{M}^0_{y,u_2+1} - \mathbf{M}^0_{y,u_2}$. Finally, the following measure for increment of u_1 and u_2 is defined: $nF_{x,u_1} = n\mathrm{trace}(\mathbf{\Delta}_{x,u_1}^{\mathrm{T}}\mathbf{\Sigma}_x\mathbf{\Delta}_{x,u_1})$ and $nF_{y,u_2} = n\mathrm{trace}(\mathbf{\Delta}_{y,u_2}^{\mathrm{T}}\mathbf{\Sigma}_y\mathbf{\Delta}_{y,u_2})$. Then, a proper value of u is set to have little changes in nF_{x,u_1} and nF_{x,u_1+1} and in nF_{y,u_2} and nF_{y,u_2+1}. It is not necessary that the selected u_1 and u_2 for $\mathbf{M}^0_{x,u_1}$ and $\mathbf{M}^0_{y,u_2}$ are common.

Next, the original two sets of variables of $\mathbf{X}$ and $\mathbf{Y}$ are replaced with $\mathbf{M}^{0\,\mathrm{T}}_{x,u_1}\mathbf{X} \in \mathbb{R}^r$ and $\mathbf{M}^{0\,\mathrm{T}}_{y,u_2}\mathbf{Y} \in \mathbb{R}^p$. This reduction of $\mathbf{X}$ and $\mathbf{Y}$ does not cause any loss of information on CCA in the sense that $\mathcal{S}(\mathbf{M}^0_{x,u_1}) = \mathcal{S}(\mathbf{M}_x)$ and $\mathcal{S}(\mathbf{M}^0_{y,u_2}) = \mathcal{S}(\mathbf{M}_y)$, and it is called *initialized CCA*. The initialized CCA has the following two cases.

> **case 1**: Suppose that $\min(p,r) = r << n$. Then, the original $\mathbf{X}$ alone is replaced with $\mathbf{M}^{0\,\mathrm{T}}_{x,u_1}\mathbf{X}$ and the original $\mathbf{Y}$ is kept.
>
> **case 2**: If $\min(p,r) = r$ is not fairly smaller than n, $\mathbf{\Sigma}_{xy}$ and $\mathbf{\Sigma}_{yx}$ are replaced by their m largest eigenvectors in the construction of $\mathbf{R}_{x,u_1}$, $\mathbf{R}_{y,u_2}$, $\mathbf{M}^0_{x,u_1}$ and $\mathbf{M}^0_{y,u_2}$. The following two ways to determine a proper value of m is recommended among many. One is a graphical determination by a scree plot for eigenvalues of $\mathbf{\Sigma}_{xy}$. The other is the number of eigenvalues whose sum is to cover 90% or above of the total variations of $\mathbf{\Sigma}_{xy}$.

The primary goal in the initialized step is the reduction of $\mathbf{X}$ and $\mathbf{Y}$ less than n without loss of information on CCA. In case 1, $\mathbf{X}$ and $\mathbf{Y}$ are reduced to r-dimensional variates, while they are replaced with the m-dimensional sets of variables in case 2. After the initialized step, r and m are fairly smaller than n.

The next step is to conduct the standard CCA for $\mathbf{M}^{0\,\mathrm{T}}_{x,u_1}\mathbf{X}$ and $\mathbf{M}^{0\,\mathrm{T}}_{y,u_2}\mathbf{Y}$ for the repairing and orthonormality. This CCA application is called *finalized CCA*. Finally, this two-step procedure for CCA is called *seeded CCA*.

Partial least squares

The main goal of the two CCA methods is dimension reduction based on the joint relation of $\mathbf{X}$ and $\mathbf{Y}$ rather than the conditional relation of $\mathbf{Y}|\mathbf{X}$. For simplicity, in this subsection, $\mathbf{Y}$ with $r = 1$ is assumed as a response variable in a regression of $\mathbf{Y}|\mathbf{X}$.

Recall $\mathbf{R}_{x,u_1}$ in (3) and $\mathbf{M}^0_{x,u_1}$ in (4):

$$\mathbf{R}_{x,u_1} = (\mathbf{\Sigma}_{xy}, \mathbf{\Sigma}_x\mathbf{\Sigma}_{xy}, \mathbf{\Sigma}_x^2\mathbf{\Sigma}_{xy}, \ldots, \mathbf{\Sigma}_x^{u_1-1}\mathbf{\Sigma}_{xy}) \text{ and } \mathbf{M}^0_{x,u_1} = \mathbf{R}_{x,u_1}(\mathbf{R}_{x,u_1}^{\mathrm{T}}\mathbf{\Sigma}_x\mathbf{R}_{x,u_1})^{-1}\mathbf{R}_{x,u_1}^{\mathrm{T}}\mathbf{\Sigma}_{xy}.$$

According to Helland (1990), the population partial least square (PLS) with u components on the regression of $\mathbf{Y}|\mathbf{X}$ is as follows:

$$\boldsymbol{\beta}_{u_1,\mathrm{PLS}} = \mathbf{M}^0_{x,u_1}. \tag{5}$$

It is noted that this PLS representation in (5) is equivalent to the canonical matrix for $\mathbf{X}$ via the seeded CCA.

Illustration of seedCCA package

Outline of seedCCA package

The methods discussed in the previous section are implemented through the main function `seedCCA`. Its arguments are as follows.

```
seedCCA(X, Y, type="seed2", ux=NULL, uy=NULL, u=10, eps=0.01, cut=0.9, d=NULL, AS=TRUE,
scale=FALSE)
```

The main function `seedCCA` returns "seedCCA" class and three subclasses depending on the values of `type`. The values of `type` and its resulting subclasses are as follows.

`type="cca"`: standard CCA ($\max(p,r) < n$ and $\min(p,r) > 1$) / "finalCCA" subclass

`type="cca"`: ordinary least squares ($\max(p,r) < n$ and $\min(p,r) = 1$) / "seedols" subclass

`type="seed1"`: seeded CCA with case1 ($\max(p,r) \geq n$) / "finalCCA" subclass

`type="seed2"`: seeded CCA with case2 ($\max(p,r) \geq n$) / "finalCCA" subclass

`type="pls"`: partial least squares ($p \geq n$ and $r < n$) / "seedpls" subclass

The function `seedCCA` prints out estimated canonical coefficient matrices for all subclasses, and additionally does canonical correlations for "finalCCA" subclasses, although it produces more outputs. For details, the readers are recommended to run `?seedCCA` after loading the **seedCCA** package. It should be noted that the **seedCCA** package must be loaded before using all functions in the package.: of **CCA** and **corpcor** (Schafer et al. (2017)).

For illustration purpose, three data sets will be considered. Pulp data is used for the standard CCA, which is available from the author's webpage (`http://home.ewha.ac.kr/~yjkstat/pulp.txt`). For the seeded CCA, along with the comparison with the regularized CCA and the partial least squares, `cookie` and `nutrimouse` in **seedCCA** package will be illustrated.

Standard CCA: pulp data

Pulp data is measurements of properties of pulp fibers and the paper made from them. It contains two sets of variables with 62 sample sizes. The first set, **Y**, is for the pulp fiber characteristics, which are arithmetic fiber length, long fiber fraction, fine fiber fraction, and zero spans tensile. The second set, **X**, is regarding the paper properties such as breaking length, elastic modulus, stress at failure, and burst strength. To implement the standard CCA application, the function `seedCCA` with `type="cca"` should be used. In this case, `seeCCA` results in the "finalCCA" subclass. The function requires two matrix-type arguments, and it returns the following five components of `cor`, `xcoef`, `ycoef`, `Xscores` and `Yscores`. The first component is `cor` is the sample canonical correlations. The next two ones, `xcoef`, and `ycoef`, are the estimated canonical matrices for **X** and **Y**. The last two components, which are `Xscores` and `Yscores`, are the estimated canonical variates for **X** and **Y**. A command `plot(object)` constructs a plot of the cumulative correlations against the number of canonical pairs. The `plot(object)` will provide a 90% reference line as default, and users can change the reference line with `plot(object,ref=percent)`.

```
## loading pulp data
> pulp <- read.table("http://home.ewha.ac.kr/~yjkstat/pulp.txt", header=TRUE)
> Y <- as.matrix(pulp[,1:4])
> X <- as.matrix(pulp[,5:8])

## standard CCA for X and Y
> fit.cca <- seedCCA(X, Y, type="cca")
NOTE:  The standard CCA is fitted for the two sets.

> names(fit.cca)
[1] "cor"     "xcoef"   "ycoef"   "Xscores" "Yscores"

## plotting cumulative canonical correlation
> par(mfrow=c(1, 2))
> plot(fit.cca, ref=80)
> plot(fit.cca)
```

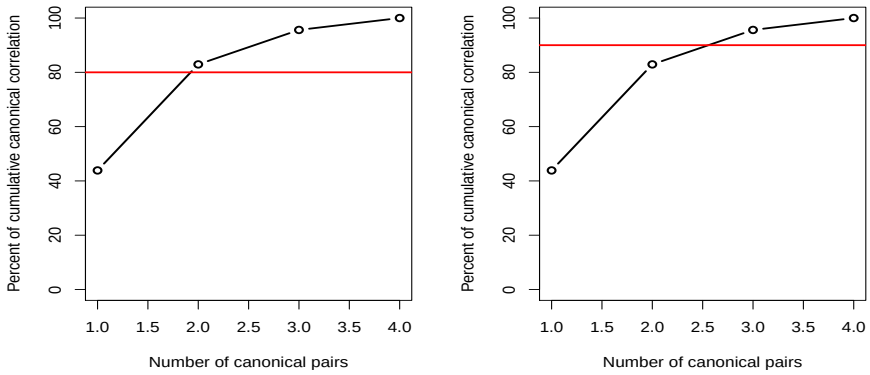


Figure 1: Cumulative canonical correlation plot in pulp data in Section 3.2

```
## first two canonical pairs
> X.cc <- fit.cca$Xscores[,1:2]
> Y.cc <- fit.cca$Yscores[,1:2]
```

According to Figure 1(a) and (b), with 80% cumulative canonical correlations, two canonical pairs are enough, while three canonical pairs should be good with the default 90%.

Ordinary least squares: pulp data

If the dimension of either **X** or **Y** is equal to one, the estimated canonical coefficient matrix from the standard CCA is equivalent to that from ordinary least squares. In such case, the command `seedCCA(X,Y[,1],type="cca")` results in the ordinary least squares estimate, which is "seedols" subclass. The output of "seedols" has three components, which are the estimated coefficients and the two sets of variables. For example, assume that a regression study of arithmetic fiber length, which is the first column of **Y**, given **X** is of specific interest. It should be noted that the order of `seedCCA(X,Y[,1],type="cca")` and `seedCCA(Y[,1],X,type="cca")` does not matter, and any of them yields the same results. Also, the commands of `coef(object)` and `fitted(object)` return the estimated coefficients and fitted values from the ordinary least squares, respectively.

```
## extracting arithmatic fiber from Y
> fit.ols <- seedCCA(X, Y[, 1], type="cca")
NOTE:  One of the two sets are 1-dimensional, so a linear regression via ordinary least
square is fitted.

> names(fit.ols)
[1] "coef"  "X"  "Y"

> coef(fit.ols)
> fitted(fit.ols)
```

Seeded CCA (case 1): cookie data

The biscuit dough data set called `cookie` in **seedCCA** comes from the experiment of analyzing the composition of biscuits by NIR spectroscopy. Two sets of variables are obtained from 72 biscuit samples. The first set of variables is wavelengths measured by spectroscopy. In the original data set, wavelengths at 700 different points from 1100 to 2798 nanometers (NM) at the steps of 2nm were measured. However, since some of the figures seemed to contain little information, wavelengths from 1380nm to 2400 at an interval of 4nm were analyzed. The second set of variables is the percentages of four ingredients: biscuits- fat, sucrose, dry flour, and water. Since the 23rd and 61st samples in the data set were believed to be outliers, they were deleted from the data set. The standard CCA is not applicable because of $p = 256 > n = 72$, and case 1 of the seeded CCA should be fitted, considering that $n = 72 >> r = 4$.

The basic command for this is `seedCCA(X,Y,type="seed1")`, which results in "finalCCA" subclass. Regardless of the order of `X` and `Y`, the lower dimensional set alone is reduced in the initial step.

Therefore, `seedCCA(X,Y,type="seed1")` and `seedCCA(Y,X,type="seed1")` basically produce the same seeded CCA results. For `type="seed1"`, the values of the options of `ux`, `uy`, `u`, `eps`, and `AS` affect the implementation, whose defaults are `NULL`, `NULL`, 10, 0.01, and `TRUE`, respectively.

Option `u` controls the maximum number of projections unless both `ux` and `uy` are specified. The option `ux=k` works only when the dimension of the first set `X` is bigger than that of the second set `Y`. Then, the maximum number of projections becomes the value given in `ux=k`. The option `uy` works in the opposite way to `ux`. The options of `AS=TRUE` and `eps` control automatic termination of the projections before reaching the maximum given in `u`, `ux`, or `uy`. The projection is terminated if the increment gets less than the value given in `eps`. Then, the first candidate value, which satisfies the stopping criteria, is suggested as a proper value of projections. If any of `ux`, `uy`, and `u` are specified not enough to guarantee the automatic stopping, a notice is provided to increase it.

After running `seedCCA(X,Y,type="seed1")`, a plot for the proper selection of `u` is automatically constructed, and a blue vertical bar in the plot is the suggested value of `u`.

```
## loading cookie data
> data(cookie)
> myseq<-seq(141, 651, by=2)
> A <- as.matrix(cookie[-c(23, 61), myseq])
> B <- as.matrix(cookie[-c(23, 61), 701:704])

## seedec CCA with case 1
> fit.seed1.ab <- seedCCA(A, B, type="seed1") ## the first set A has been initial-CCAed.
NOTE: Seeded CCA with case 1 is fitted. The set with larger dimension is initially reduced.
The first and second sets are denoted as X and Y, respectively.

> fit.seed1.ba <- seedCCA(B, A, type="seed1") ##  the second set A has been initial-CCAed.
NOTE: Seeded CCA with case 1 is fitted. The set with larger dimension is initially reduced.
The first and second sets are denoted as X and Y, respectively.

> names(fit.seed1.ab)
[1] "cor"  "xcoef"  "ycoef"  "proper.u"  "initialMX0"  "newX" "Y"  "Xscores"  "Yscores"

> names(fit.seed1.ba)
[1] "cor"  "xcoef"  "ycoef"  "proper.u"  "X"  "initialMY0"  "newY"  "Xscores"  "Yscores"

> fit.seed1.ab$xcoef[, 3] <- -fit.seed1.ab$xcoef[, 3] ## changing the sign
> fit.seed1.ab$xcoef[, 4] <- -fit.seed1.ab$xcoef[, 4] ## changing the sign

> all(round(fit.seed1.ab$cor, 5)== round(fit.seed1.ba$cor, 5))
[1] TRUE

> fit.seed1.ab$proper.u
[1] 3

> fit.seed1.ba$proper.u
[1] 3

> all(round(fit.seed1.ab$xcoef, 5) == round(fit.seed1.ba$ycoef, 5))
[1] TRUE

> fit.seed1.ab.ux <- seedCCA(A, B, type="seed1", ux=2)
The maximum number of iterations is reached. So, users must choose u bigger than 2.

> fit.seed1.ab.ux$proper.u
[1] 2
```

For `fit.seed1.ab`, the first set `A` is reduced in the initial step. The output component `initialMX0` is the estimate of $\mathbf{M}^0_{x,u_1}$ and `newX` is $\hat{\mathbf{M}}^{0\,\mathrm{T}}_{x,u_1}\mathbf{X}$. On the contrary, in case of `fit.seed1.ba`, the second set `A` is initially reduced, so `initialMY0` and `newY` are produced. So, it is observed that the canonical correlations and suggested values of `u` from `fit.seed1.ab` and `fit.seed1.ba` are equal, not to mention that `fit.seed1.ab$xcoef` and `fit.seed1.ba$ycoef` are the same. The selection plot for `u` is reported in Figure 2, and three projections are suggested. Since `ux` is not given big enough in `seedCCA(A,B,type="seed1",ux=2)`, the following warning is given:

```
The maximum number of iterations is reached. So, users must choose u bigger than 2.
```

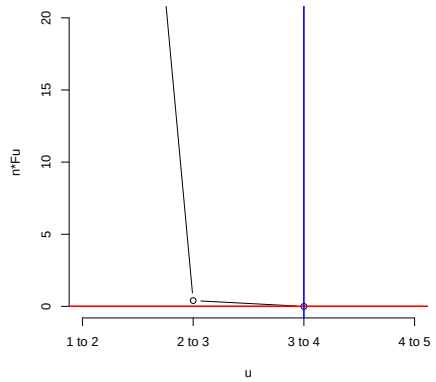


Figure 2: Selection plot of u generated from `seedCCA(A, B, type="seed1")` in Section 3.4

Next, we change the values of `ux`, `uy`, `AS`, and `eps`. Since the usage of these options for `type="seed1"` are the same as that for `type="seed2"` and `type="pls"`. To measure the computing time, the **tictoc** package (Izrailev (2014)) is used with Intel(R) Core(TM)i7 2.9GHz and 12GB Ram computer.

```
> seedCCA(A, B, type="seed1", ux=5)$proper.u
[1] 3

> seedCCA(B, A, type="seed1", eps=0.000001)$proper.u
[1] 4

> library(tictoc)
> tic()
> seedCCA(B, A, type="seed1", u=30)$proper.u
> toc()
0.03 sec elapsed

> tic()
> seedCCA(B, A, type="seed1", u=30, AS=FALSE)$proper.u
> toc()
0.29 sec elapsed
```

Usage of `AS` should be noted. With bigger choices of u and `AS=FALSE`, the running time of the function will be longer.

Seeded CCA (case 2) versus Regularized CCA: nutrimouse data

The nutrimouse data was collected from a nutrition study in 40 mice $(n = 40)$. One of two sets of variables was expressions of 120 genes measured in liver cells by microarray technology. The other set of variables was concentrations of 21 hepatic fatty acids(FA) measured through gas chromatography. In addition, the forty mice are cross-classified based on two factors, genotype and diet. There are two genotypes, wild-type (WT) and PPARα deficient (PPARα) mice and five diets: corn and colza oils (50/50 REF), hydrogenated coconut oil for a saturated FA diet (COC), sunflower oil for ω6 FA-rich diet (SUN), linseed oil for ω3-rich diet (LIN) and corn/colza/enriched fish oils (42.5/42.5/15, FISH). The nutrimouse data is contained in the **seedCCA** package.

In this data, case 2 of the seeded CCA should be used because $\min(120, 21)$ is relatively big compared to $n = 40$. Then, case 2 of the seeded CCA requires to choose how many eigenvectors of $\hat{\mathbf{\Sigma}}_{xy}$ should be enough to replace it. This is another tuning parameter for case 2 of the seeded CCA along with the number of projections. The option `cut` in `seedCCA` controls automatic selection of the number of eigenvectors of $\hat{\mathbf{\Sigma}}_{xy}$. The option `cut=`$\alpha$ determines a set of the eigenvectors whose cumulative proportions of their corresponding eigenvalues is bigger than equal to α. For the set of eigenvectors to be chosen conservatively, we set the default of `cut` at 0.9. Also, users can directly give the number of eigenvectors using `d`. Unless `d` is `NULL`, the option `cut` is discarded. This means that `cut` works only when `d=NULL`. If users want to use `d`, then a function `covplot` should be run first. The function `covplot` has the option `mind`, which set the number of the eigenvalues to show their cumulative percentages. Its default is `NULL`, and then it becomes $\min(p, r)$. The function returns the eigenvalues, the cumulative

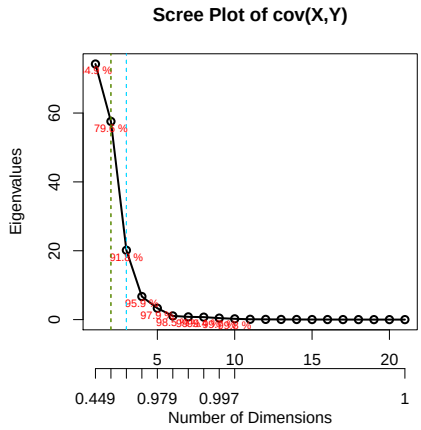


Figure 3: Scree plots for the selection of sets of eigenvectors to replace $\text{cov}(\mathbf{X}, \mathbf{Y})$ generated from `covplot(X, Y, mind=10)` in Section 3.5

percentages, and the number of the eigenvectors to account for 60%, 70%, 80%, and 90% of the total variation along with the scree plot of the eigenvalues.

The results of the seeded and regularized CCAs are compared. Since the regularized CCA is necessary to choose proper values of the two parameters, we compare running times for the automatic searches for the regularized and seeded CCAs via the **tictoc** package. For **seedCCA**, we use the default value of `cut`.

```
> library(CCA)
> library(tictoc)

## loading nutrimouse data
> data(nutrimouse)
> X <- scale(as.matrix(nutrimouse$gene))
> Y <- scale(as.matrix(nutrimouse$lipid))

## determining the number of the eigenvectors of cov(X,Y) with cut=0.9
> tic("SdCCA")
> fit.seed2 <- seedCCA(X, Y)
> toc()
SdCCA: 0.13 sec elapsed

## finding the optimal values of lambda1 and lambda2 for RCCA
> tic("Regularized CCA")
> res.regul <- estim.regul(X, Y, plt=TRUE, grid1=seq(0.0001, 0.2, l=51), grid2=seq(0, 0.2, l=51))
> toc()
Regularized CCA 819.58 sec elapsed

## scree plot of cov(X, Y)
> names(covplot(X, Y, mind=10))
[1] "eigenvalue"  "cum.percent" "num.evecs"

> names(fit.seed2)
[1] "cor"  "xcoef"  "ycoef"  "proper.ux"  "proper.uy"  "d"  "initialMX0"  "initialMY0"
[9] "newX"  "newY"  "Xscores"    "Yscores"

> fit.seed2$d
[1] 3
```

Since `type="seed2"` reduces the dimensions of `X` and `Y` at the initialized CCA step, the output components of `initialMX0`, `initialMY0`, `newX` and `newY` and `d` are reported.

The plot generated from `covplot(X,Y,mind=10)` is given in Figure 3. According to Figure 3, the first two, three, and four eigenvalues account for 79.6%, 91.8%, and 95.9% of the total variation of $\hat{\mathbf{\Sigma}}_{xy}$, respectively. Using 90% conservative guideline, it is determined that the first three largest eigenvectors replace $\hat{\mathbf{\Sigma}}_{xy}$ well enough.

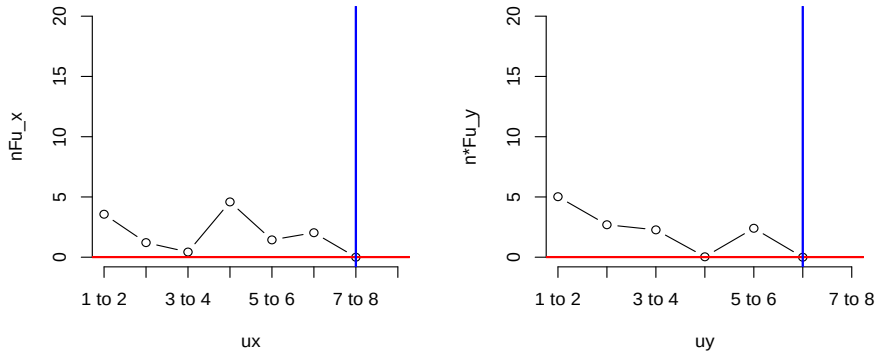


Figure 4: Selection plot of ux and uy generated from `seedCCA(X, Y)` in Section 3.5

The selection plot of ux and uy is given in Figure 4. The figure suggests that u_x and u_y are equal to 7 and 6, respectively.

Now we compare the parameter selection time. For the regularized CCA, it can be done with `estim.regul`, and users must provide a small enough range for them to reduce the computing time. The resulted optimal λ_1 and λ_2 are 0.168016 and 0.004, respectively. With Intel(R) Core(TM)i7 2.9GHz and 12GB Ram, the seeded CCA took 0.32 seconds, while 819.58 seconds, around 13.5 minutes, lapsed for the regularized CCA. This difference is really huge, so time consumed in the selection of ux, uy and, d is trivially small compared to the regularized CCA. This is a clear desirable aspect and advantage of the seeded CCA over the regularized one.

Next, we compare the first two pairs of estimated canonical variates. The results shown in Figures 5–6 are equivalent to the analysis discussed in González et al. (2008).

```
## Extracting the first two pairs of canonical variates
> sx1 <- fit.seed2$Xscores[, 1]
> sx2 <- fit.seed2$Xscores[, 2]
> sy1 <- fit.seed2$Yscores[, 1]
> sy2 <- fit.seed2$Yscores[, 2]

## fitting the regularized CCA
> res.rcc <- rcc(X, Y, 0.168016, 0.004)
> RCCA.X <- X%*%res.rcc$xcoef
> RCCA.Y <- Y%*%res.rcc$ycoef
> rx1 <- RCCA.X[,1]
> rx2 <- RCCA.X[,2]
> ry1 <- RCCA.Y[,1]
> ry2 <- RCCA.Y[,2]

par(mfrow=c(1,2))
> with(plot(rx1, ry1, col=c(2,4)[genotype], pch=c(1,2)[genotype],
+ main="1st pair from RCCA", xlab="rx1", ylab="ry1"), data=nutrimouse)
> with(legend(-1.4, 1.4, legend=levels(genotype), col=c(2,4), pch=c(1,2), cex=1.5),
+ data=nutrimouse)
> with(plot(-sx1, -sy1, col=c(2,4)[genotype], pch=c(1,2)[genotype],
+ main="1st pair from seedCCA", xlab="sx1", ylab="sy1"), data=nutrimouse)
> with(legend(-1.5, 1.6, legend=levels(genotype), col=c(2,4), pch=c(1,2), cex=1.5),
+ data=nutrimouse)

> par(mfrow=c(1,2))
> with(plot(rx2, ry2, col=c(1:4,6)[diet], pch=c(15,16,17,18,20)[diet], cex=1.5,
+ main="2nd pair from RCCA", xlab="rx2", ylab="ry2"), data=nutrimouse)
> with(legend(-2.3, 1.9, legend=levels(diet), col=c(1:4,6), pch=c(15:18,20)),
+ data=nutrimouse)
> with(plot(sx2, sy2, col=c(1:4,6)[diet], pch=c(15,16,17,18,20)[diet], cex=1.5,
+ main="2nd pair from seedCCA", xlab="sx2", ylab="sy2"), data=nutrimouse)
with(legend(-2.5, 1.9, legend=levels(diet), col=c(1:4,6), pch=c(15:18,20)),
```

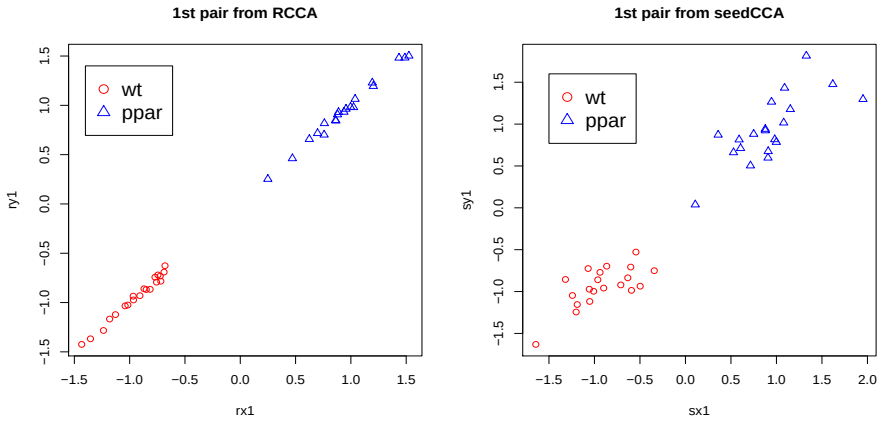


Figure 5: the first pair of canonical variates from regularized CCA and seeded CCA marked with genotype in Section 3.5

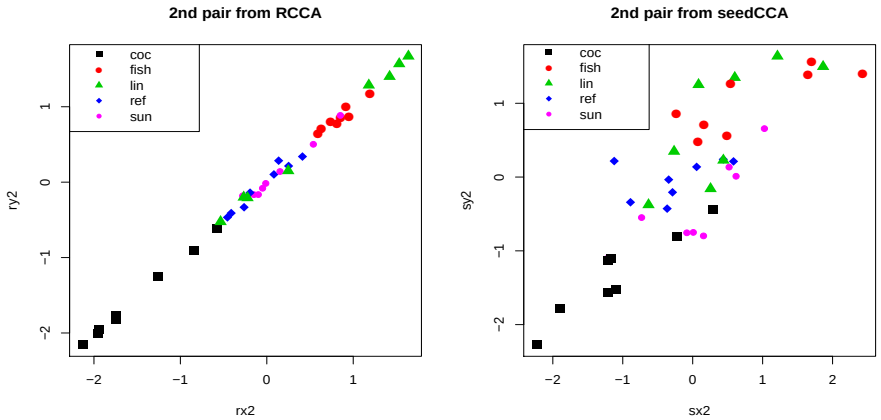


Figure 6: the second pair of canonical variates from regularized CCA and seeded CCA marked with diet in Section 3.5

```
+ data=nutrimouse)
```

According to Figures 5–6, the first pair of canonical variates from both CCAs distinguish genotype very well, but their second pairs marked with diet are quite complex. To have more insight into the results for the second pair on a diet, multivariate analysis of variance is fitted. Further pairwise comparison is done via **lsmeans** (Lenth (2016)) with level 5% and p-values adjusted by false discovery rate Benjamini and Hochberg (1995).

```
> library(lsmeans)
> fit2r <- manova(cbind(rx2, ry2)~diet, data=nutrimouse)
> fit3sd <- manova(cbind(sx2, sy2)~diet, data=nutrimouse)
> test( contrast( lsmeans(fit2r, "diet"), "pairwise"), side = "=",  adjust = "fdr")
 contrast     estimate        SE df t.ratio p.value
 coc - fish -2.3842686 0.2684019 35  -8.883  <.0001
 coc - lin  -2.1749708 0.2684019 35  -8.103  <.0001
 coc - ref  -1.4881111 0.2684019 35  -5.544  <.0001
 coc - sun  -1.6582635 0.2684019 35  -6.178  <.0001
 fish - lin  0.2092978 0.2684019 35   0.780  0.4897
 fish - ref  0.8961575 0.2684019 35   3.339  0.0040
 fish - sun  0.7260051 0.2684019 35   2.705  0.0175
 lin - ref   0.6868597 0.2684019 35   2.559  0.0214
 lin - sun   0.5167073 0.2684019 35   1.925  0.0780
 ref - sun  -0.1701524 0.2684019 35  -0.634  0.5302

Results are averaged over the levels of: rep.meas
P value adjustment: fdr method for 10 tests
```

```
> test( contrast (lsmeans(fit3sd, "diet"), "pairwise"), side = "=",  adjust = "fdr")
 contrast      estimate        SE df t.ratio p.value
 contrast      estimate        SE df t.ratio p.value
 coc - fish -2.14838660 0.3163067 35  -6.792  <.0001
 coc - lin  -1.79396325 0.3163067 35  -5.672  <.0001
 coc - ref  -1.07697196 0.3163067 35  -3.405  0.0035
 coc - sun  -1.03303440 0.3163067 35  -3.266  0.0041
 fish - lin  0.35442334 0.3163067 35   1.121  0.3001
 fish - ref  1.07141463 0.3163067 35   3.387  0.0035
 fish - sun  1.11535219 0.3163067 35   3.526  0.0035
 lin - ref   0.71699129 0.3163067 35   2.267  0.0371
 lin - sun   0.76092885 0.3163067 35   2.406  0.0308
 ref - sun   0.04393756 0.3163067 35   0.139  0.8903

Results are averaged over the levels of: rep.meas
P value adjustment: fdr method for 10 tests
```

For the regularized CCA, the "coc" diet is different from the others. Moreover "fish" differs from "sun". However, the other pairwise comparisons are quite mixed. It is determined that there are no significant differences between "fish-lin", "lin-sun", and "ref-sun". On the contrary, reasonable pairwise comparison results come from the seeded CCA. Like the others, the "coc" diet is different from the others. Furthermore, "fish-lin" is not significantly different, and "ref-sun" is concluded to be similar. Fish oil is known to contain $\omega 3$, and linseed oil is designed for it. Therefore, this conclusion would be reasonable. Also, the reference oil diet consists of corn and colza oil, which is known to contain $\omega 6$. Since sun-flower oil is, indeed, for $\omega 6$-rich diet, this result is also reasonable. In this regard, the seeded CCA results would be preferable to the regularized CCA.

Partial least square application with nutrimouse data

With the nutrimouse data, consider a regression of the first one, " C14.0" in concentrations of 21 hepatic fatty acids given expressions of 120 genes measured in liver cells. In this case, partial least squares is a front-runner choice. Then, to obtain the partial least square estimator in **seedCCA**, one needs to implement `seedCCA(X,Y,type="pls")`. This results in "seedpls" subclass. An important matter in partial least squares is that the first set of the variable must be predictors. The response variable can be either univariate or multivariate. Option `u` is recommended to set reasonably small because the estimated coefficients are reported up to the value given in `u`. If `scale=TRUE`, the predictors are standardized to have zero sample means and the sample correlation matrix.

The estimated coefficients and fitted values by partial least square can be obtained via `coef(object,u=NULL)` and `fitted(object,u=NULL)`. The default of `u` in both `coef` and `fitt` is `NULL`. In both functions, usage of `u` is equivalent. If `u=k` is specified, only the estimated coefficients and fitted values computed from `k` projections are reported. All of the coefficient estimates and fitted values are reported up to `u`, if `u=NULL`.

For `type="pls"`, the automatic procedure to suggest a proper value of projections is not conducted. For the "seedpls" subclass, `plot(object)` suggests a proper value of projections along with other output components. If the terminating condition is not satisfied before reaching the value of `u`, then `plot(object)` provides a caution to increase the value of `u`.

```
> data(nutrimouse)
> Y <- as.matrix(nutrimouse$lipid)
> X <- as.matrix(nutrimouse$gene)
> Y1 <- as.matrix(Y[, 1])  ## univariate response
> Y12 <- as.matrix(Y[, 1:2])  ## multivariate response

## fitting partial least square and obtaining the estimated coefficient vector
> fit.pls1.10 <- seedCCA(X, Y1, u=10, type="pls")
> fit.pls1.3 <- seedCCA(X, Y1, u=3, type="pls", scale=TRUE)

> names(fit.pls1.10)
[1] "coef"  "u"     "X"     "Y"     "scale"

> names(fit.pls1.10$coef)
[1] u=1"  "u=2"  "u=3"  "u=4"  "u=5"  "u=6"  "u=7"  "u=8"  "u=9"  "u=10"
```

```
> names(fit.pls1.3$coef)
[1] "u=1" "u=2" "u=3"

> fit.pls1.3$scale
[1] TRUE

> par(mfrow=c(1,2))
> plot(fit.pls1.10)
$proper.u
[1] 6
$nFu
[1] 6.344725e+00 2.383108e+00 1.681329e+00 2.669394e+00 1.853061e+00 3.217472e-04 5.296046e-05
[8] 6.017641e-06 4.895905e-07 3.117371e-08
$u
[1] 10
$eps
[1] 0.01
> title("fit.pls1.10")

> plot(fit.pls1.3)
Caution: The terminating condition is NOT satisfied. The number of projections should be bigger than 3.
$proper.u
[1] 3
$nFu
[1] 6.344725 2.383108 1.681329
$u
[1] 3
$eps
[1] 0.01
> title("fit.pls1.3")

> names(fitted(fit.pls1.10))
[1] "u=1"  "u=2"  "u=3"  "u=4"  "u=5"  "u=6"  "u=7"  "u=8"  "u=9"  "u=10"

> fitted(fit.pls1.10, u=6)
     1     2     3     4     5     6     7     8     9    10    11    12    13    14
 0.137 0.368 0.317 0.346 0.492 1.620 0.722 0.003 0.065 1.212 0.458 0.640 0.272 0.397
    15    16    17    18    19    20    21    22    23    24    25    26    27    28
-0.103 0.426 1.448 0.287 1.264 0.517 2.803 0.914 0.043 0.028 0.234 0.598 0.875 0.434
    29    30    31    32    33    34    35    36    37    38    39    40
 0.694 0.666 2.958 2.350 0.620 0.958 0.495 2.790 0.701 0.168 0.767 0.535

> fit.pls.m <- seedCCA(X, Y12, u=5, type="pls")
> dim(fit.pls.m$coef$'u=1')
[1] 120   2
```

The selection of projections for two partial least squares by `seedCCA(X,Y1,u=10,type="pls")` and `seedCCA(X,Y1,u=3,type="pls",scale=TRUE)` is given in Figure 7. According to Figure 7, the proper value of projection is suggested at 6 for `fit.pls1.10` object, while the termination condition is not satisfied for `fit.pls1.3` object, so a caution statement is given.

Discussion

When a study between two sets of variables, saying ($\mathbf{X} \in \mathbb{R}^p, \mathbf{Y} \in \mathbb{R}^r$), is of primary interest, canonical correlation analysis (CCA; Hotelling (1936)) is still popularly used in explanatory studies. The CCA has successful application in many science fields such as chemometrics, pattern recognition, genomic sequence analysis, and so on.

The recently developed **seedCCA** package implements a collection of CCA methodologies including the standard CCA application, seeded CCA, and partial least squares. The package enables us to fit CCA to large-p and small-n data. The paper provides a complete guide for the package to implement all the methods, along with three real data examples. Also, the seeded CCA application results are compared with the regularized CCA in the existing **CCA** package.

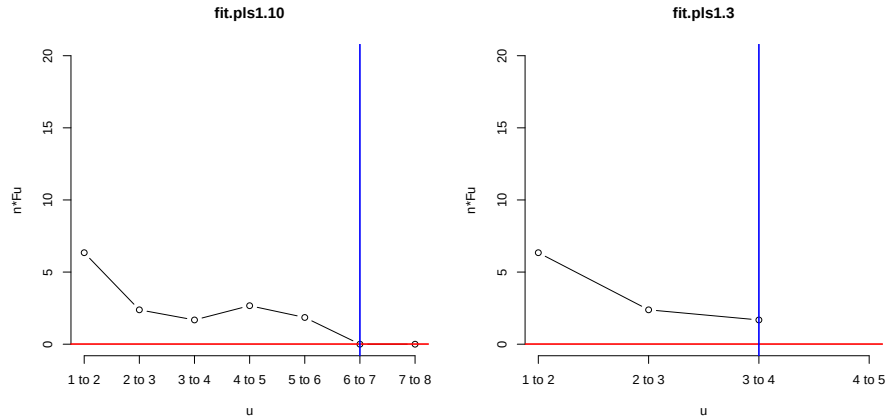


Figure 7: Selection plot of u generated from `seedCCA(X, Y1, u=10, type="pls")`(left) and `seedCCA(X, Y1, u=3, type="pls", scale=TRUE)`(right) in Section 3.6

It is believed that the package, along with the paper, will contribute to high-dimensional data analysis in various scientific field practitioners and that the statistical methodologies in multivariate analysis become more fruitful.

Acknowledgments

For the corresponding author Jae Keun Yoo, this work was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Korean Ministry of Education (NRF-2019R1F1A1050715). For Bo-Young Kim, this work was supported by the BK21 Plus Project through the National Research Foundation of Korea (NRF) funded by the Korean Ministry of Education (22A20130011003).

Bibliography

A. Alfons, C. Croux, and P. Filzmoser. Robust maximum association between data sets: The R package ccaPP. *Austrian Journal of Statistics*, 45(1):71–79, 2016. [p7]

Y. Benjamini and Y. Hochberg. Controlling the false discovery rate: A practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society, Series B*, 57(1):289–300, 1995. URL https://doi.org/10.1111/j.2517-6161.1995.tb02031.x. [p16]

R. D. Cook, B. Li, and F. Chiaromonte. Dimension reduction in regression without matrix inversion. *Biometrika*, 94(3):569–584, 2007. URL https://doi.org/10.1093/biomet/asm038. [p9]

R. Cruz-Cano. *FRCC: Fast Regularized Canonical Correlation Analysis*, 2012. URL https://CRAN.R-project.org/package=FRCC. R package version 1.0. [p7]

I. González, S. Déjean, P. G. P. Martin, and A. Baccini. Cca: an r package to extend canonical correlation analysis. *Journal of Statistical Software*, 23(12):1–13, 2008. URL https://doi.org/10.18637/jss.v023.i12. [p7, 15]

I. S. Helland. Partial least squares regression and statistical models. *Scandinavian Journal of Statistics*, 17(2):97–114, 1990. URL https://www.jstor.org/stable/4616159. [p9]

H. Hotelling. Relations between two sets of variates. *Biometrika*, 28(3):321–377, 1936. URL https://www.jstor.org/stable/2333955. [p7, 18]

Y. Im, H. Gang, and J. Yoo. High-throughput data dimension reduction via seeded canonical correlation analysis. *Journal of Chemometrics*, 29(3):193–199, 2014. URL http://dx.doi.org/10.1002/cem.2691. [p7, 8]

S. Izrailev. *tictoc: Functions for timing R scripts, as well as implementations of Stack and List structures.*, 2014. URL https://CRAN.R-project.org/package=tictoc. R package version 1.0. [p13]

R. Johnson and D. W. Wichern. *Applied Multivariate Statistical Analysis*. Pearson Prentice Hall, 2007. [p8]

K. Lee and J. Yoo. Canonical correlation analysis through linear modeling. *Australian and New Zealand Journal of Statistics*, 56(1):59–72, 2014. URL http://dx.doi.org/10.1111/anzs.12057. [p7, 8]

R. V. Lenth. Least-squares means: The R package lsmeans. *Journal of Statistical Software*, 69(1):1–33, 2016. doi: 10.18637/jss.v069.i01. [p16]

S. E. Leurgans, R. A. Moyeed, and B. W. Silverman. Canonical correlation analysis when the data are curves. *Journal of the Royal Statistical Society, Series B*, 55(3):725–740, 1993. URL https://www.jstor.org/stable/2345883. [p7]

J. Schafer, R. Opgen-Rhein, M. A. V. Zuber, A. P. D. Silva, and K. Strimmer. *corpcor: Efficient Estimation of Covariance and (Partial) Correlation*, 2017. URL https://CRAN.R-project.org/package=corpcor. R package version 1.6.9. [p10]

H. D. Vinod. Canonical ridge and econometrics of joint production. *Journal of Econometrics*, 4(2): 147–166, 1976. URL https://doi.org/10.1016/0304-4076(76)90010-5. [p7]

Bo-Young Kim, Researcher
Celltrion
Incheon, 22014
Republic of Korea

Yunju Im, Postdoctoral Associate
Department of Biostatistics, Yale University
New Haven, CT 06520
United States of America

Jae Keun Yoo, Professor
Department of Statistics, Ewha Womans University
Seoul, 03760
Republic of Korea
peter.yoo@ewha.ac.kr

npcure: An R Package for Nonparametric Inference in Mixture Cure Models

by Ana López-Cheda, M. Amalia Jácome, Ignacio López-de-Ullibarri

Abstract Mixture cure models have been widely used to analyze survival data with a cure fraction. They assume that a subgroup of the individuals under study will never experience the event (cured subjects). So, the goal is twofold: to study both the cure probability and the failure time of the uncured individuals through a proper survival function (latency). The R package **npcure** implements a completely nonparametric approach for estimating these functions in mixture cure models, considering right-censored survival times. Nonparametric estimators for the cure probability and the latency as functions of a covariate are provided. Bootstrap bandwidth selectors for the estimators are included. The package also implements a nonparametric covariate significance test for the cure probability, which can be applied with a continuous, discrete, or qualitative covariate.

Introduction

In classical survival analysis, it is assumed that all the individuals will eventually experience the event of interest. However, there are many contexts in which this assumption might not be true. Noticeable examples are the lifetime of cancer patients after treatment, time to infection in a risk population, or time to default in credit scoring, among many others. Cure models are a stream of methods recently developed in survival analysis that take into account the possibility that subjects could never experience the event of interest. See Maller and Zhou (1996) for early references and Amico and Van Keilegom (2018) for an updated review.

Let $\mathbf{X}$ be a set of covariates and Y the time to the event of interest with conditional survival function $S(t|\mathbf{x}) = P(Y > t|\mathbf{X} = \mathbf{x})$. Mixture cure models, initially proposed by Boag (1949), consider the population as a mixture of two types of subjects: the susceptible of experiencing the event if followed for long enough ($Y < \infty$) and the cured ones ($Y = \infty$). Hence, the survival function of Y can be written as

$$S(t|\mathbf{x}) = 1 - p(\mathbf{x}) + p(\mathbf{x}) S_0(t|\mathbf{x}),$$

where $1 - p(\mathbf{x}) = P(Y = \infty|\mathbf{X} = \mathbf{x}) = \lim_{t\to\infty} S(t|\mathbf{x})$ is the cure probability, and the (proper) survival function of the uncured subjects or *latency* is $S_0(t|\mathbf{x}) = P(Y > t|Y < \infty, \mathbf{X} = \mathbf{x})$. A major advantage of these models over the non-mixture approach is that they allow the covariates to have different effect on cured and uncured individuals.

The cure probability, $1 - p(\mathbf{x})$, is usually estimated parametrically by assuming a logistic form $\log(p(\mathbf{x}) / (1 - p(\mathbf{x}))) = \boldsymbol{\beta}'\mathbf{x}$, with $\boldsymbol{\beta}$ a parameter vector. Estimation of $S_0(t|\mathbf{x})$ can be done by assuming a particular parametric distribution for the failure time of the uncured subjects, or more generally, by applying, e.g., proportional hazards (PH) or accelerated failure time (AFT) assumptions. These two approaches lead to parametric (see, e.g., Farewell, 1982, 1986; Denham et al., 1996) or semiparametric (see, e.g., Kuk and Chen, 1992; Peng et al., 1998; Peng and Dear, 2000; Li and Taylor, 2002) mixture cure models.

An attractive feature of parametric and semiparametric models is that they provide close expressions for relevant parameters and functions. On the other hand, the sound inference is guaranteed only if the chosen model fits the data suitably. A problem with these methods is that the parametric assumptions may be incorrect. For example, regarding the cure rate $1 - p(\mathbf{x})$, there is no reason to believe that the cure rate is monotone in $\mathbf{x}$, let alone that it follows a logistic model. To solve this hassle, Müller and Van Keilegom (2019) propose a test statistic to assess whether the cure rate, as a function of $\mathbf{X}$, satisfies a certain parametric model. As for the latency function, $S_0(t|\mathbf{x})$, it is difficult to verify the distributional assumptions of the model. The goodness of fit for the latency function has only been addressed in settings without covariates and in an informal way (Maller and Zhou, 1996). The challenge of developing procedures for testing the parametric form of the conditional survival function of the uncured with covariates is even more ambitious. It would lead to curse-of-dimensionality problems and remains an open question.

As a result of the increasing demand for the use of cure models, the number of packages in R accounting for the possibility of cure in survival analysis has grown significantly over the last decade: see the CRAN task view on survival analysis (`https://CRAN.R-project.org/view=Survival`). The **smcure** package (Cai et al., 2012) fits the semiparametric PH and AFT mixture cure models (see Kalbfleisch and Prentice, 2002). Besides, the **NPHMC** package (Cai et al., 2013) allows to calculate the sample size of a survival trial with or without cure fractions. More recently, the **flexsurvcure** package (Amdahl, 2017) provides flexible parametric mixture and non-mixture cure models for time-

to-event data, and the **rcure** package (Han et al., 2017) incorporates methods related to robust cure models for survival data which include a weakly informative prior in the logistic part. The **geecure** package (Niu and Peng, 2018) features the marginal parametric and semiparametric PH mixture cure models for analyzing clustered survival data with a possible cure fraction. Furthermore, the **miCoPTCM** package (Bertrand et al., 2020) fits semiparametric promotion time cure models with possibly mis-measured covariates, while the **mixcure** package (Peng, 2020) implements parametric and semiparametric mixture cure models based on existing R packages. For interval-censored data with a cure fraction, the **GORcure** package (Zhou et al., 2017) implements the generalized odds rate mixture cure model, including the PH mixture cure model and the proportional odds mixture cure model as special cases. The **intercure** package (Brettas, 2016) provides an implementation of semiparametric cure rate estimators for interval-censored data using bounded cumulative hazard and frailty models.

In contrast with (semi)parametric procedures, nonparametric methods do not rely on data belonging to any particular parametric family or fulfilling any parametric assumption. They estimate the goal functions without making any assumptions about its shape, so they have much wider applicability than alternative parametric methods. A completely nonparametric mixture cure model must consider purely nonparametric estimators for both the cure rate, $1 - p(\mathbf{x})$, and latency function, $S_0(t|\mathbf{x})$. Unlike the (semi)parametric approach, nonparametric mixture cure models have been under study only in recent years. Laska and Meisner (1992), building on the Kaplan-Meier (KM) product-limit (PL) estimator of the survival function $S(t) = P(Y > t)$ (Kaplan and Meier, 1958), derive nonparametric estimators of the cure rate and latency function, but their model does not allow for covariates. More recently, Xu and Peng (2014) propose a nonparametric estimator of the cure rate with one or more covariates, showing its consistency and asymptotic normality. This estimator was further studied by López-Cheda et al. (2017a), who, besides proving that it is the maximum likelihood nonparametric estimator of the cure probability, also obtain an i.i.d. representation and proposed a bootstrap-based bandwidth selector. As for the latency function, López-Cheda et al. (2017b) introduce a completely nonparametric estimator, studied some theoretical properties, and proposed a bandwidth selector based on the bootstrap.

Although some of the aforementioned packages have a nonparametric flavor, their approach to mixture cure modeling is not completely nonparametric. Our R package **npcure** (López-de-Ullibarri et al., 2020) fills the gap by providing implementations of the nonparametric estimator of the cure rate function proposed by Xu and Peng (2014) (further studied by López-Cheda et al., 2017a) and of the nonparametric estimator of the latency function proposed by López-Cheda et al. (2017b).

Furthermore, the generalized PL estimator of the conditional survival function, $S(t|x)$, proposed by Beran (1981), is implemented. Note that the estimators of the cure rate and latency implemented in **npcure** relate strongly to Beran estimator. In any case, Beran estimator is of outstanding importance by its own, as evidenced by the variety of R packages with functions for computing it, like, e.g., `Beran()` in package **condSURV** (Meira-Machado and Sestelo, 2016), `prodlim()` in package **prodlim** (Gerds, 2018) and `Beran()` in package **survidm** (Meira-Machado et al., 2019). The function in our package compares advantageously with the aforementioned functions with respect to the issue of bandwidth selection. This smoothing parameter plays an essential role in the bias-variance tradeoff of every nonparametric smoothing method. In Dabrowska (1992), an expression for the bandwidth minimizing the asymptotic mean squared error (MSE) of this estimator was obtained, and a plug-in bandwidth selector was proposed based on suitable estimators of the unknown functions in that expression. However, the performance of this bandwidth selector is unsatisfying for small sample sizes, and a cross-validation (CV) procedure is usually preferred (see Iglesias-Pérez, 2009; Gannoun et al., 2007, among others). Recently, Geerdens et al. (2017) propose an improved CV bandwidth selector, especially with a high censoring rate. To the best of our knowledge, there are not any R packages allowing to compute Beran estimator with a suitable bandwidth selector: while the **condSURV** and **survidm** packages do not consider any bandwidth selectors, the **prodlim** package uses nearest neighborhoods as the smoothing parameter. The **npcure** package, available from the Comprehensive R Archive Network (CRAN) at `https://CRAN.R-project.org/package=npcure`, fulfills this need with the implementation of the CV bandwidth selector for the Beran estimator in Geerdens et al. (2017).

In this paper, we explain how the **npcure** package can be used in the context of nonparametric mixture cure models with right-censored data. The main objective is to estimate the cure probability and latency functions, as well as to perform covariate significance tests for the cure rate. In the next section, we describe our approach to nonparametric estimation in mixture cure models. The methodology applied in the covariate significance tests is presented in another section. Two sections follow, devoted respectively to explain the package functions and to illustrate their use with an application to a medical dataset.

Nonparametric estimation in mixture cure models

One of the specificities of time-to-event data is related to the presence of individuals that have not experienced the event by the end of the study. The observed survival times of these individuals are said to be *right-censored* and underestimate the true unknown time to the occurrence of the event. This situation is usually modeled by considering a censoring variable C, with distribution function G, which is conditionally independent of Y given the covariate $\mathbf{X}$. The observed data are then $\{(\mathbf{X}_i, T_i, \delta_i) : i = 1, \ldots, n\}$, where $T = \min(Y, C)$ is the observed lifetime and $\delta = \mathbf{1}(Y \leq C)$ is the uncensoring indicator. For a one-dimensional continuous covariate X, Xu and Peng (2014) propose the following nonparametric kernel-type estimator of the cure rate:

$$1 - \hat{p}_h(x) = \prod_{i=1}^{n} \left(1 - \frac{\delta_{[i]} B_{h[i]}(x)}{\sum_{r=i}^{n} B_{h[r]}(x)}\right) = \hat{S}_h(T^1_{\max}|x), \tag{1}$$

where, for $i = 1, \ldots, n$, $\delta_{[i]}$ and $X_{[i]}$ are the concomitant status indicator and covariate corresponding to the ith ordered time $T_{(i)}$, and

$$B_{h[i]}(x) = \frac{K_h\left(x - X_{[i]}\right)}{\sum_{j=1}^{n} K_h\left(x - X_{[j]}\right)} \tag{2}$$

are the Nadaraya-Watson weights, where $K_h(\cdot) = \frac{1}{h}K\left(\frac{\cdot}{h}\right)$ is a rescaled kernel with bandwidth $h \to 0$. Although some different kernel functions could be considered, the Epanechnikov kernel, defined as

$$K(u) = \frac{3}{4}(1 - u^2)\mathbf{1}(|u| \leq 1),$$

is the one implemented in the **npcure** package. Moreover, $\hat{S}_h$ is the estimator of the conditional survival function S in Beran (1981), and $T^1_{\max} = \max_{\{i:\delta_i=1\}} T_i$ is the largest uncensored failure time. Xu and Peng (2014) prove the consistency and asymptotic normality of the estimator in (1), and López-Cheda et al. (2017a) show that it is the local maximum likelihood estimator of the cure rate, and obtained an i.i.d. representation and an asymptotic expression for the MSE.

The nonparametric latency estimator proposed by López-Cheda et al. (2017a), and further studied in López-Cheda et al. (2017b), is:

$$\hat{S}_{0,b}(t|x) = \frac{\hat{S}_b(t|x) - (1 - \hat{p}_b(x))}{\hat{p}_b(x)}, \tag{3}$$

where $\hat{S}_b$ is the PL estimator of the conditional survival function S (Beran, 1981) and $\hat{p}_b$ is the estimator in (1). As in the case of the cure rate estimator, a smoothing parameter b, not necessarily equal to h, is needed to compute $\hat{S}_{0,b}$ in (3).

Consistency of the nonparametric estimators

The proposed nonparametric estimators of both the cure rate and latency are consistent under the general condition (see Laska and Meisner, 1992; Maller and Zhou, 1992; López-Cheda et al., 2017a,b)

$$\tau_0 \leq \tau_G(x), \tag{4}$$

where $\tau_0 = \sup_x \tau_0(x)$, and $\tau_0(x) = \sup\{t \geq 0 : S_0(t|x) > 0\}$ and $\tau_G(x) = \sup\{t \geq 0 : G(t|x) < 1\}$ are the right endpoints of the support of the conditional distribution of the uncures and the censoring variable, respectively.

The condition in (4) ensures $1 - p(x)$ and $S_0(t|x)$ to be consistently estimated when there is zero probability that a susceptible individual survives beyond the largest possible censoring time, $\tau_G(x)$. Since $T^1_{\max}$ converges to τ_0 in probability (see Xu and Peng, 2014), assumption (4) guarantees that, asymptotically, all times observed after the largest uncensored survival time, $T^1_{\max}$, can be assumed to correspond to cures.

Under condition (4), $S_0(\tau_G(x)|x) = 0$ and, for large n, the cure rate estimator in (1) tends to a nonparametric estimator of $S(\tau_G(x)|x) = 1 - p(x) + p(x)S_0(\tau_G(x)|x) = 1 - p(x)$. However, if there could be uncured individuals surviving beyond $\tau_G(x)$, then $S_0(\tau_G(x)|x) > 0$ and the estimator in (1) would estimate $S(\tau_G(x)|x) = 1 - p(x) + p(x)S_0(\tau_G(x)|x) > 1 - p(x)$. This might happen, for example, in a clinical trial with fixed maximum follow-up time.

These comments emphasize that care must be exercised in choosing the length of follow-up if

cures might be present since too much censoring or insufficient follow-up time could lead to erroneous conclusions. For example, if the last observation is uncensored, then, even if there is considerable late censoring, the estimated cure rate is 0. To avoid these difficulties, particularly with heavy censoring, reasonably long follow-up times and large sample sizes may be required. In this way, $S_0\left(\tau_G(x)|x\right)$ is sufficiently small for the cure rate estimator in (1) to be close enough to $1-p(x)$.

Thus, when estimating $1-p(x)$ and $S\left(t|x\right)$ for a given x with a data set, it is important to be confident that $\tau_0 \leq \tau_G(x)$. In any case, if the censoring distribution $G(t|x)$ has a heavier tail than $S_0\left(t|x\right)$, the cure rate estimates computed with the nonparametric estimator in (1) will tend to have smaller biases regardless of the value of $\tau_0(x)$ (see Xu and Peng, 2014). Maller and Zhou (1992) propose a simple nonparametric test to assess condition (4). The procedure is based on the length of the interval $(T^1_{\max}, T_{(n)}]$, i.e., the right tail of the KM estimate where it has a constant value. A long plateau with heavy censoring at the right tail of the KM curve is interpreted as evidence that follow-up time has been long enough to conclude that condition (4) holds.

Bandwidth selection

The nonparametric estimators in (1) and (3) depend on two smoothing parameters, h and b, respectively. Bootstrap-based selectors for the bandwidth h of the cure rate estimator and the bandwidth b of the latency estimator are proposed by López-Cheda et al. (2017a) and López-Cheda et al. (2017b), respectively. The bandwidths are locally chosen so that the selected bandwidths h_x and b_x depend on the point x of estimation. Using locally adaptive bandwidths instead of global ones is advantageous because they adapt to the structure of the underlying function, differentially smoothing its flat and peaky parts.

For a fixed value x, the bootstrap bandwidth of the cure estimator, h^*_x, was introduced by López-Cheda et al. (2017a) as the minimizer of the bootstrap MSE, approximated with B resamples as follows:

$$MSE^*_x(h_x) \simeq \frac{1}{B}\sum_{b=1}^{B}\left(\hat{p}^{*b}_{h_x}(x) - \hat{p}_{g_x}(x)\right)^2, \tag{5}$$

where $\hat{p}^{*b}_{h_x}(x)$ is the estimator of $p(x)$ in (1) computed with $\left\{\left(X^{*b}_i, T^{*b}_i, \delta^{*b}_i\right) : i = 1,\ldots,n\right\}$ (the bth bootstrap resample), and using the local bandwidth h_x, and $\hat{p}_{g_x}(x)$ is computed with the original sample $\{(X_i, T_i, \delta_i) : i = 1,\ldots,n\}$, and the local pilot bandwidth g_x.

With respect to the latency estimator in (3), López-Cheda et al. (2017b) propose to choose the bandwidth b_x locally with a bootstrap bandwidth selector. The bootstrap bandwidth of the latency estimator, b^*_x, is taken as the minimizer of the bootstrap mean integrated squared error (MISE):

$$MISE^*_x(b_x) \simeq \frac{1}{B}\sum_{b=1}^{B}\int_0^u \left(\hat{S}^{*b}_{0,b_x}\left(t|x\right) - \hat{S}_{0,g_x}\left(t|x\right)\right)^2 dt, \tag{6}$$

where $\hat{S}^{*b}_{0,b_x}\left(t|x\right)$ is the nonparametric estimator of $S_0\left(t|x\right)$ in (3) computed with the bth bootstrap resample and local bandwidth b_x, $\hat{S}_{0,g_x}\left(t|x\right)$ is the same estimator obtained using the original sample and a local pilot bandwidth g_x, and u is an adequately chosen upper bound of the integral.

For a fixed covariate value x, the procedure for obtaining the bootstrap bandwidth selector of h_x for $\hat{p}_{h_x}(x)$ (respectively, b_x for $\hat{S}_{0,b_x}\left(t|x\right)$) is as follows:

1. Generate B bootstrap resamples $\left\{\left(X^{*b}_i, T^{*b}_i, \delta^{*b}_i\right) : i = 1,\ldots,n\right\}$, for $b = 1,\ldots,B$.
2. Consider a search grid of bandwidths $h_l \in \{h_1,\ldots,h_L\}$. For $b = 1,\ldots,B$ and $l = 1,\ldots,L$, compute the nonparametric estimator $\hat{p}^{*b}_{h_l}(x)$ (respectively, the nonparametric latency estimator, $\hat{S}^{*b}_{0,h_l}\left(t|x\right)$) with the bth bootstrap resample and bandwidth h_l.
3. Compute the nonparametric estimator $\hat{p}_{g_x}(x)$ (respectively, the nonparametric latency estimator $\hat{S}_{0,g_x}\left(t|x\right)$) with the original sample and pilot bandwidth g_x.
4. For each bandwidth $h_l \in \{h_1,\ldots,h_L\}$, compute the Monte Carlo approximation of $MSE^*_x(h_l)$ in (5), (respectively, the Monte Carlo approximation of $MISE^*_x(h_l)$ in (6)).
5. The bootstrap bandwidth h^*_x for the cure rate estimator (respectively, b^*_x for the latency estimator) is the minimizer of the Monte Carlo approximation of $MSE^*_x(h_l)$ (respectively, $MISE^*_x(h_l)$) over the grid of bandwidths $\{h_1,\ldots,h_L\}$.

Following López-Cheda et al. (2017a) and López-Cheda et al. (2017b), the bootstrap resamples in Step 1 are generated considering the following procedure, which is equivalent to the simple weighted bootstrap proposed by Li and Datta (2001) without resampling the covariate X:

I. Generate $X_1^*, \ldots, X_n^*$ by fixing $X_i^* = X_i$, $i = 1, \ldots, n$.

II. For each i, compute the weighted empirical distribution $\hat{F}_{g_{X_i^*}}\left(t, \delta | X_i^*\right)$ with the original sample, where $\hat{F}_{g_x}\left(t, \delta | x\right) = \sum_{i=1}^{n} B_{g_x i}(x) \mathbf{1}\left(T_i \leq t, \delta_i \leq \delta\right)$ and $B_{g_x i}(x)$ is computed with a local pilot bandwidth g_x (see (7) below).

III. For each i, generate the pair $\left(T_i^*, \delta_i^*\right)$ from the weighted empirical estimator $\hat{F}_{g_{X_i^*}}\left(t, \delta | X_i^*\right)$ of the conditional distribution.

López-Cheda et al. (2017a) and López-Cheda et al. (2017b) show that the effect of the pilot bandwidth on the bootstrap bandwidth selectors of h_x and b_x is considerably low. Consequently, the same expression for the pilot bandwidth, g_x, is used in Step II of the bootstrap resampling procedure and in the approximation of the MSE_x^* in (5) for the selection of the bandwidth h_x of the cure rate estimator (respectively, in the approximation of the $MISE_x^*$ in (6) for the bandwidth b_x of the latency estimator):

$$g_x = \frac{d_k^+(x) + d_k^-(x)}{2} 100^{1/9} n^{-1/9}, \tag{7}$$

where $d_k^+(x)$ (respectively, $d_k^-(x)$) is the distance from x to the kth nearest neighbor on the right (respectively, on the left). If there are not at least k neighbors on the right (or left), we use $d_k^+(x) = d_k^-(x)$. López-Cheda et al. (2017a) show that a good choice for the parameter k is to consider $k = n/4$. The order $n^{-1/9}$ satisfies the conditions in Theorem 1 of Li and Datta (2001) and coincides with the optimal order for the pilot bandwidth obtained by Cao and González-Manteiga (1993) in the case without censoring.

When selecting locally adaptive bandwidths, the results might look a little bit spiky due to its local nature (see, e.g., Brockmann et al., 1993, on local bandwidth selection for kernel regression estimators). That could be the case for the bootstrap bandwidths for both the cure rate and latency functions. To get rid of the fluctuation of these local bandwidths, h_x and b_x can be further smoothed, for example, by computing a centered moving average of the unsmoothed vector of bandwidths as in López-Cheda et al. (2017a).

Covariate significance tests

In medical studies, it is usually important to assess whether the cure probability depends on a specific covariate, X. Noting that the cure rate can be interpreted as the regression function $E\left(\nu | X = x\right) = 1 - p(x)$, where ν is the indicator of cure, the question can be cast in the form of a hypothesis test:

$$\begin{cases} H_0 : E\left(\nu | X\right) = 1 - p \\ H_1 : E\left(\nu | X\right) = 1 - p(X) \end{cases}. \tag{8}$$

Although there are some parametric approaches to deal with this hypothesis testing problem (see Müller and Van Keilegom, 2019, among others), the only completely nonparametric method was introduced by López-Cheda et al. (2020). Their procedure is based on the test for selecting explanatory variables in nonparametric regression described by Delgado and González-Manteiga (2001). The greatest advantage of the proposed significance test for the cure rate is that although the test is completely nonparametric, no smoothing parameters are required to test (8).

The main challenge when testing (8) is that the cure indicator, ν, is only partially known due to censoring: complete observations are known to be uncured ($\nu = 0$), but censored observations might be either cured or uncured (i.e., ν is unknown). Under right censoring, all of the cured individuals and some of the uncured ones will be censored. This makes it difficult to guess whether a censored observation belongs to the cured or uncured subpopulation. López-Cheda et al. (2020) solved this situation by replacing the unknown and inestimable response variable ν in (8) by an unknown but estimable response η with the same conditional expectation as ν:

$$\eta = \frac{\nu\left(1 - \mathbf{1}(\delta = 0, T \leq \tau)\right)}{1 - G\left(\tau | X\right)}, \tag{9}$$

where τ is an unknown time beyond which a lifetime might be assumed to be cured. López-Cheda et al. (2020) propose to estimate η by replacing G and τ with suitable nonparametric estimators. The censoring distribution is estimated with the generalized PL estimator by Beran (1981) computed with the cross-validation (CV) bandwidth selector in Geerdens et al. (2017) when X is continuous and with the stratified KM estimator with the same bandwidth selector otherwise. The cure threshold, τ, is estimated as $\hat{\tau} = T_{\max}^1$, the largest uncensored observed time. The expression of η in (9) avoids the need for an estimator of the unknown cure indicator, ν, since if $\delta_i = 1$ or $(\delta_i = 0, T_i < \hat{\tau})$ then $\hat{\eta}_i = 0$,

whereas if $(\delta_i = 0, T_i \geq \hat{\tau})$ then $\hat{\eta}_i = 1/\left(1 - \hat{G}(\hat{\tau}|X_i)\right)$. It is easy to check that $E(\nu|X) = E(\eta|X)$ if the conditional censoring distribution $G(t|x)$ is independent of the cure status.

Finally, building on Delgado and González-Manteiga (2001) and using the estimated values of η in (9), the significance test proposed by López-Cheda et al. (2020) is based on the process:

$$U_n(x) = \frac{1}{n}\sum_{i=1}^{n}\left(\hat{\eta}_i - \frac{1}{n}\sum_{j=1}^{n}\hat{\eta}_j\right)\mathbf{1}(X_i \leq x). \tag{10}$$

Cramér-von Mises (CM) or Kolmogorov-Smirnov (KS) test statistics can be used:

$$\begin{aligned} CM_n &= \sum_{i=1}^{n} U_n^2(X_i), \\ KS_n &= \max_{i=1,\ldots,n} n^{1/2}|U_n(X_i)|. \end{aligned} \tag{11}$$

Note that if X is a nominal variable, it is impossible to compute the indicator function in (10). In this case, López-Cheda et al. (2020) propose to consider all the possible 'ordered' permutations of the values of X and to compute $U_n(x)$ according to the 'ordering' of each permutation. The values of the CM and KS test statistics are given by the maximum of the values CM_n and KS_n computed along with all the permutations.

The distribution of the CM and KS statistics under the null hypothesis is approximated by bootstrap, according to the following steps:

A. Obtain X_i^*, $i = 1, \ldots, n$, by randomly resampling with replacement from $\{X_1, \ldots, X_n\}$.

B. Estimate the probability of cure under H_0 as $1 - \hat{p} = \hat{S}_n^{KM}\left(T_{\max}^1\right)$, with $\hat{S}_n^{KM}$ the KM estimator of the survival function $S(t) = P(Y > t)$. For $i = 1, \ldots, n$:

   B.1. Compute $\hat{S}_{0,b}\left(t|X_i^*\right)$, a nonparametric estimator of the latency $S_0\left(t|X_i^*\right)$, with the original sample. Set $Y_i^* = \infty$ with probability $1 - \hat{p}$, and draw Y_i^* from $\hat{S}_{0,b}(t|X_i^*)$ with probability $\hat{p}$.

   B.2. Generate C_i^* from a nonparametric estimator of $G\left(t|X_i^*\right)$ with the original sample.

   B.3. Compute $T_i^* = \min\left(Y_i^*, C_i^*\right)$ and $\delta_i^* = \mathbf{1}\left(Y_i^* \leq C_i^*\right)$.

C. With the bootstrap resample $\left\{\left(X_i^*, T_i^*, \delta_i^*\right) : i = 1, \ldots, n\right\}$ compute $\hat{\eta}_i^*$ for $i = 1, \ldots, n$.

D. With $\left\{\left(\hat{\eta}_i^*, X_i^*\right) : i = 1, \ldots, n\right\}$, compute the bootstrap versions of U_n in (10) and the corresponding CM and KS statistics, CM_n^* and KS_n^*.

E. Repeat Steps A-D above B times in order to generate B values of the CM and KS statistics, $\left\{CM_n^{*1}, \ldots, CM_n^{*B}\right\}$ and $\left\{KS_n^{*1}, \ldots, KS_n^{*B}\right\}$.

F. The p-value of the CM (respectively, KS) test is approximated as the proportion of values $\left\{CM_n^{*1}, \ldots, CM_n^{*B}\right\}$ larger than CM_n (respectively, $\left\{KS_n^{*1}, \ldots, KS_n^{*B}\right\}$ larger than KS_n).

Note that nonparametric estimators of the conditional functions $S_0(t|x)$ and $G(t|x)$ are required in Step B. Following López-Cheda et al. (2020), if X is continuous, then $S_0(t|x)$ and $G(t|x)$ are estimated with the nonparametric estimator in (3) and the generalized PL estimator in Beran (1981), respectively, and with the corresponding stratified unconditional estimators otherwise.

The npcure package: structure and functionality

The **npcure** package provides several functions to model nonparametrically survival data with a possibility of cure. Table 1 contains a compact summary of the available functions. The estimators of the cure rate and latency functions, discussed in the section "Nonparametric estimation in mixture cure models", are implemented by `probcure()` and `latency()`, respectively. The functions `probcurehboot()` and `latencyhboot()` compute bootstrap bandwidths for these estimators. Another function deserving mention in this context is `beran()`, which computes the generalized PL estimator of the conditional survival function $S(t|x)$. A CV bandwidth for use with `beran()` is returned by `berancv()`. Given the computational burden of the procedures implemented by the aforementioned functions, all of them make extensive use of compiled C code. The significance test introduced in the previous section is carried out by `testcov()`, and `testmz()` performs the nonparametric test of Maller and Zhou (1992). Next, a detailed account of the usage of all these functions is provided.

The estimation functions in **npcure** are restricted to one-dimensional continuous covariates. The Epanechnikov kernel is used in the smoothing procedures. Nonparametric estimation with discrete or

Function	Description
`beran`	Computes Beran's estimator of the conditional survival function.
`berancv`	Computes the CV bandwidth for Beran's estimator of the conditional survival function.
`controlpars`	Sets the control parameters of the `latencyhboot()` and `probcurehboot()` functions.
`hpilot`	Computes pilot bandwidths for the nonparametric estimators of the cure rate and the latency.
`latency`	Computes the nonparametric estimator of the latency.
`latencyhboot`	Computes the bootstrap bandwidth for the nonparametric estimator of the latency.
`print.npcure`	Method of the generic function `print` for 'npcure' objects.
`probcure`	Computes the nonparametric estimator of the cure rate.
`probcurehboot`	Computes the bootstrap bandwidth for the nonparametric estimator of the cure rate.
`summary.npcure`	Method of the generic function `summary` for 'npcure' objects.
`testcov`	Performs covariate significance tests for the cure rate.
`testmz`	Performs the nonparametric test of Maller and Zhou (1992).

Table 1: Summary of the functions in the **npcure** package.

categorical variables could be dealt with as in other kernel smoothing procedures. A simple approach is to split the sample into a number of subsets according to the covariate values. When the size of the subsamples is not too small, valid unconditional estimates of the cure probability and latency can be computed. Another alternative is the use of special kernels that can handle any covariate types (see Racine and Li, 2004).

Several features are shared by the functions in the package. All functions return an object of S3 class 'npcure', formally a list of components. Among these components are the primary outputs of the functions, like the computed estimates for `probcure()` and `latency()`, the selected bandwidths for `probcurehboot()` and `latencyhboot()`, or the p-values of the tests for `testcov()` and `testmz()`. The covariate values, observed times, and uncensoring indicators are passed to the functions via the `x`, `t`, and `d` arguments, respectively. Typically, a set of names is passed, which are interpreted as column names of a data frame specified by the `dataset` argument. However, `dataset` may also be left as `NULL`, the default, in which case the objects named in `x`, `t`, and `d` must live in the working directory. More details on these and other arguments are given in the following.

Estimation of the cure rate

The estimation of the cure rate using the nonparametric estimator in (1) is implemented in the `probcure()` function:

```
probcure(x, t, d, dataset = NULL, x0, h, local = TRUE, conflevel = 0L,
   bootpars = if (conflevel == 0 && !missing(h)) NULL else controlpars())
```

The `x0` argument specifies the covariate values where conditional estimates of the cure rate are to be computed. The bandwidths required by the estimator are passed to the `h` argument. The `local` argument is a logical value determining whether the bandwidths are interpreted as local (`local = TRUE`) or global (`local = FALSE`) bandwidths. Notice that if `local = TRUE`, then `h` and `x0` must have the same length. Actually, the `h` argument may be missing, in which case the local bootstrap bandwidth computed by the `probcurehboot()` function is used. This last function implements the procedure for selecting the bandwidth h_x^* described in the section "Bandwidth selection", and its usage is:

```
probcurehboot(x, t, d, dataset, x0, bootpars = controlpars())
```

The `bootpars` argument controls the details of the computation of the bootstrap bandwidth (see section "Bandwidth selection"). In typical use, it is intended to receive the list returned by the `controlpars()` function. The components of this list are described in Table 2.

The function `probcure()` also allows constructing point confidence intervals (CI) for the cure rate. These CIs exploit the asymptotic normality of the estimator (Xu and Peng, 2014), using the bootstrap to obtain an estimate of the standard error of the estimated cure rate. The bootstrap resamples are generated by the same procedure described in the section "Bandwidth selection". Denoting by $z_{1-\alpha/2}$

Argument	Description
B	Number of bootstrap resamples (by default, 999).
hbound	A vector giving the minimum and maximum, respectively, of the initial grid of bandwidths as multiples of the standardized interquartile range (IQR) of the covariate values (by default, c(0.1,3)).
hl	Length of the initial grid of bandwidths (by default, 100).
hsave	A logical specifying if the grid of bandwidths is saved (by default FALSE).
nnfrac	Fraction of the sample size determining the order k of the nearest neighbor used when computing the pilot bandwidth g_x in (7) (by default, 0.25).
fpilot	Either NULL, the default, or a function name. If NULL, the pilot bandwith is computed by the package function hpilot(). If not NULL, it is the name of an alternative, user-defined function for computing the pilot.
qt	In bandwidth selection with latencyhboot(), order of the quantile of the observed times specifying the upper bound of the integral in the computation of the MISE* in (6) (by default, 0.75).
hsmooth	Order of a moving average computed to optionally smooth the selected bandwidths. By default is 1, meaning that no smoothing is done.

Table 2: Summary of the arguments of the controlpars() function.

the $1-\alpha/2$ quantile of a standard normal and by $\widehat{se}_B\left(1-\hat{p}_h(x)\right)$ the estimate of the standard error of $1-\hat{p}_h(x)$ with B bootstrap resamples, a $(1-\alpha)\,100\%$ CI for $1-p(x)$ is computed as:

$$1-\hat{p}_h(x) \mp z_{1-\frac{\alpha}{2}}\widehat{se}_B\left(1-\hat{p}_h(x)\right). \tag{12}$$

The confidence level of the CI is specified through the conflevel argument as a number between 0 and 1. With the special value 0, the default, no CI is computed. Other parameters related to the bootstrap CIs can be passed to the bootpars argument, typically via the output of the controlpars() function. These parameters relate to the number of bootstrap resamples and the computation of the pilot bandwidth, and are specified, respectively, by the B and nnfrac arguments described in Table 2.

The usage of these functions is illustrated with a simulated dataset generated from a model where the cure probability is a logistic function of the covariate:

```
library("npcure")
n <- 50
x <- runif(n, -2, 2)
y <- rweibull(n, shape = 0.5 * (x + 4), scale = 1)
c <- rexp(n, rate = 1)
p <- exp(2 * x)/(1 + exp(2 * x))
u <- runif(n)
t <- ifelse(u < p, pmin(y, c), c)
d <- ifelse(u < p, ifelse(y < c, 1, 0), 0)
data <- data.frame(x = x, t = t, d = d)
```

In the next code example, point and 95% CI estimates of the cure probability are obtained with probcure() at a grid of covariate values ranging from -1.5 to 1.5. For the estimation, the local bootstrap bandwidths previously computed by probcurehboot() are passed to the h argument. The bandwidths, which have been further smoothed with a moving average of 15 bandwidths, are contained in the hsmooth component of the output of probcurehboot(). For the bootstrap, 2000 resamples are generated.

```
x0 <- seq(-1.5, 1.5, by = 0.1)
hb <- probcurehboot(x, t, d, data, x0 = x0,
   bootpars = controlpars(B = 2000, hsmooth = 15))
q1 <- probcure(x, t, d, data, x0 = x0, h = hb$hsmooth, conflevel = 0.95,
   bootpars = controlpars(B = 2000))
q1

#> Bandwidth type: local
#>
#> Conditional cure estimate:
#>         h   x0        cure lower 95% CI upper 95% CI
```

```
#> 0.6212329 -1.5 1.000000000   0.98450759   1.00000000
#> 0.6523881 -1.4 1.000000000   0.87087244   1.00000000
#> 0.6533320 -1.3 1.000000000   0.86080078   1.00000000
#> 0.6606362 -1.2 1.000000000   0.83135572   1.00000000
#> 0.6710717 -1.1 1.000000000   0.82267310   1.00000000
#> 0.6912311 -1.0 0.972213147   0.78259082   1.00000000
#> ...
```

More compactly, the same bootstrap bandwidths would be selected and the same estimates obtained if `h` were left unset when calling `probcure()`:

```
q2 <- probcure(x, t, d, data, x0 = x0, conflevel = 0.95,
   bootpars = controlpars(B = 2000, hsmooth = 15))
```

Figure 1 shows a plot of the true cure rate function and its point and 95% CI estimates at the covariate values saved in `x0`. The plot can be reproduced by executing the next code. The components of the `q1` object accessed by the code are `x0`, keeping the vector of covariate values, `q`, containing the point estimates of the cure rate, and `conf`, a list with the lower (component `lower`) and upper (component `upper`) limits of the CIs for the cure rate.

```
plot(q1$x0, q1$q, type = "l", ylim = c(0, 1), xlab = "Covariate X",
   ylab = "Cure probability")
lines(q1$x0, q1$conf$lower, lty = 2)
lines(q1$x0, q1$conf$upper, lty = 2)
lines(q1$x0, 1 - exp(2 * q1$x0)/(1 + exp(2 * q1$x0)), col = 2)
legend("topright", c("Estimate", "95% CI limits", "True"),
   lty = c(1, 2, 1), col = c(1, 1, 2))
```

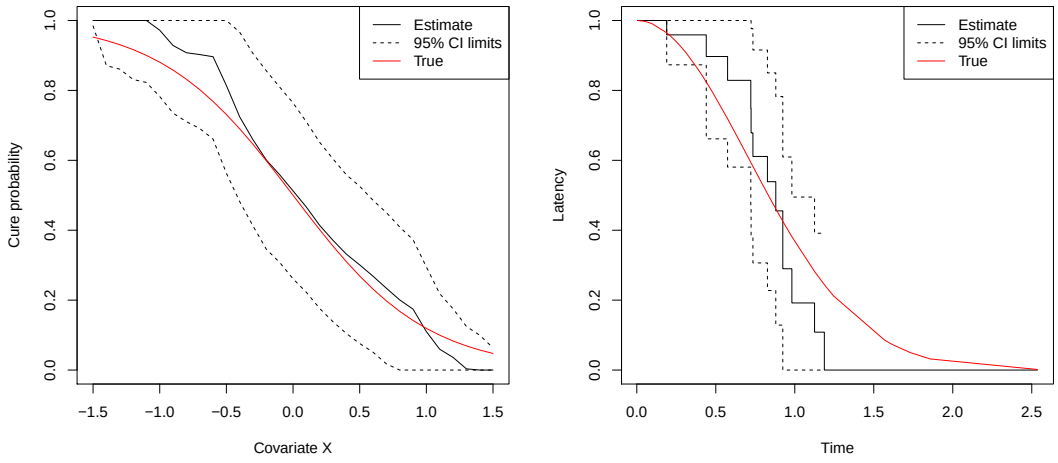


Figure 1: Left panel: estimation of the cure rate. Right panel: estimation of the latency for $x = 0$.

Estimation of the latency function

The latency estimator in (3) is implemented in the `latency()` function:

```
latency(x, t, d, dataset = NULL, x0, h, local = TRUE, testimate = NULL,
  conflevel = 0L, bootpars = if (conflevel == 0) NULL else controlpars(),
  save = TRUE)
```

The function's interface is similar to that of `probcure()`, with all the arguments, except for `testimate`, having exactly the same interpretation. The `testimate` argument determines the times t at which the function $S_0(t|x)$ is estimated. It defaults to `NULL`, which results in the latency being estimated at times given by the `t` argument.

Also, as was the case for `probcure()`, `latency()` allows getting bootstrap CIs for the latency function by specifying their level with the `conflevel` argument. These CIs also rely on the asymptotic

normality of the latency estimator $\hat{S}_{0,b}(t|x)$ in (3) (López-Cheda et al., 2017b). A $(1-\alpha)$ 100% CI for $S_{0,b}(t|x)$ is computed as:

$$\hat{S}_{0,b}(t|x) \mp z_{1-\frac{\alpha}{2}} \widehat{se}_B\left(\hat{S}_{0,b}(t|x)\right), \tag{13}$$

where $\widehat{se}_B\left(\hat{S}_{0,b}(t|x)\right)$ is a bootstrap estimate of the standard error of $\hat{S}_{0,b}(t|x)$, the bootstrap resamples being generated as described in the section "Bandwidth selection".

Also, as with `probcure()`, the user can specify a local or global bandwidth with the combined use of the `h` and `local` arguments. When `h` is left unspecified, a local bootstrap bandwidth is indirectly computed by the `latencyhboot()` function:

```
latencyhboot(x, t, d, dataset = NULL, x0, bootpars = controlpars())
```

This function provides an implementation of the bandwidth selector b_x^* introduced in the section "Bandwidth selection". It is homologous to `probcurehboot()`, with which it shares a common interface. The only noticeable difference is that now the `qt` argument of `controlpars()` (see Table 2) can be used to set u, the upper bound of the integral that must be calculated when computing the bootstrap MISE in (6).

Using the same simulated data as before, the next code illustrates the computation of point and 95% CI estimates (based on 500 bootstrap resamples) of the latency for covariate values 0 and 0.5, and with local bandwidths equal to 0.8 and 0.5, respectively. Notice that, since the `testim` argument is unset, the estimates are computed at the times `t`:

```
S0 <- latency(x, t, d, data, x0 = c(0, 0.5), h = c(0.8, 0.5),
   conflevel = 0.95, bootpars = controlpars(B = 500))
```

To estimate the latency using the bootstrap bandwidth selector, `latencyhbooot()` can be called before calling `latency()`. In the following code, the component `h` of the output of `latencyhbooot()`, where the selected local bandwidths are contained, is passed to the `h` argument of `latency()`:

```
b <- latencyhboot(x, t, d, data, x0 = c(0, 0.5))
S0 <- latency(x, t, d, data, x0 = c(0, 0.5), h = b$h, conflevel = 0.95)
S0
```

```
#> Bandwidth type: local
#>
#> Covariate (x0): 0.0 0.5
#> Bandwidth (h):  4.531978 2.527206
#>
#> Conditional latency estimate:
#>
#> x0 = 0
#>        time   latency lower 95% CI upper 95% CI
#> 0.004599127 1.0000000   1.00000000    1.0000000
#> 0.042088293 1.0000000   1.00000000    1.0000000
#> 0.042271452 1.0000000   1.00000000    1.0000000
#> 0.059671372 1.0000000   1.00000000    1.0000000
#> 0.067375891 1.0000000   1.00000000    1.0000000
#> 0.098569312 1.0000000   1.00000000    1.0000000
#> ...
#>
#> x0 = 0.5
#>        time   latency lower 95% CI upper 95% CI
#> 0.004599127 1.0000000   1.00000000    1.0000000
#> 0.042088293 1.0000000   1.00000000    1.0000000
#> 0.042271452 1.0000000   1.00000000    1.0000000
#> 0.059671372 1.0000000   1.00000000    1.0000000
#> 0.067375891 1.0000000   1.00000000    1.0000000
#> 0.098569312 1.0000000   1.00000000    1.0000000
#> ...
```

An alternative, more succinct way to proceed is to leave `h` unset, since in that case, `latencyhboot()` is indirectly called:

```
S0 <- latency(x, t, d, data, x0 = c(0, 0.5), conflevel = 0.95)
```

Figure 1 shows the estimated and true latencies for covariate value $x = 0$. Next, the code to obtain the plot is reproduced, and it is helpful in illustrating the structure of the output list returned

by `latency()`. The `testim` component has the times at which the estimates are computed. The `S` component is a list having a named item for each covariate value. Each element contains the latency estimates for a covariate value, and the name is constructed from the covariate value by prefixing it with an `x`. The `conf` component is also a named list, the names being constructed as those of the `S` component. Each one of these items contains, structured as a list, the lower (`lower` component) and upper (`upper` component) limits of the CIs. Finally, `x0` keeps the covariate values as a separate element.

```
plot(S0$testim, S0$S$x0, type = "s", xlab = "Time", ylab = "Latency",
   ylim = c(0, 1))
lines(S0$testim, S0$conf$x0$lower, type = "s", lty = 2)
lines(S0$testim, S0$conf$x0$upper, type = "s", lty = 2)
lines(S0$testim, pweibull(S0$testim, shape = 0.5 * (S0$x0[1] + 4),
   scale = 1, lower.tail = FALSE), col = 2)
legend("topright", c("Estimate", "95% CI limits", "True"),
   lty = c(1, 2, 1), col = c(1, 1, 2))
```

Significance test for the cure rate

The **npcure** package also provides an implementation of the nonparametric covariate significance tests for the cure rate discussed in the section "Covariate significance tests":

```
testcov(x, t, d, dataset = NULL, bootpars = controlpars(), save = FALSE)
```

The `x` argument is the covariate whose effect on the cure rate is to be tested. The function's output is a list whose main components are `CM` and `KS`. Each of them, in turn, is a list containing the test statistic (`stat`) and p-value (`pvalue`) of the CM and KS tests, respectively.

The result of the test carried out with our simulated data and 2500 bootstrap resamples is:

```
testcov(x, t, d, data, bootpars = controlpars(B = 2500))
```

```
#> Covariate test
#>
#> Covariate:  x
#>               test statistic p.value
#>   Cramer-von Mises 0.4537077  0.0592
#> Kolmogorov-Smirnov 1.2456568  0.0708
```

Non-numeric covariates can also be tested. For example, for `z`, a nominal covariate added to the simulated data, the result is:

```
data$z <- rep(factor(letters[1:5]), each = 10)
testcov(z, t, d, data, bootpars = controlpars(B = 2500))
```

```
#> Covariate test
#>
#> Covariate:  z
#>               test statistic p.value
#>   Cramer-von Mises 0.2513218  0.6356
#> Kolmogorov-Smirnov 0.7626470  0.5340
```

Estimation of the conditional survival function

The **npcure** package also includes the `beran()` function, which computes the generalized PL estimator of the conditional survival function, $S(t|x)$, by Beran (1981). The `beran()` function in our package may be used together with the `berancv()` function:

```
berancv(x, t, d, dataset, x0, cvpars = controlpars())
```

This function computes the local CV bandwidth selector of Geerdens et al. (2017). It can be directly called by the user, but in practical work should be more usual an indirect call from the `beran()` function, which, as said before, computes the generalized PL estimator of $S(t|x)$:

```
beran(x, t, d, dataset, x0, h, local = TRUE, testimate = NULL, conflevel = 0L,
   cvbootpars = if (conflevel == 0 && !missing(h)) NULL else controlpars())
```

The arguments of these two functions have the same meaning as their homonyms in the `latency()` and `latencyhboot()` functions, `cvpars` and `cvbootpars` playing the role of `bootpars` in these last functions. As in `latency()`, if no bandwidth is provided by the user via `h`, then the local CV bandwidth in Geerdens et al. (2017) is computed by `berancv()`.

For example, the code below computes the Beran estimator for the covariate values 0 and 0.5 using local CV bandwidths. The default behavior of `berancv()` is modified by the auxiliary function `controlpars()`. In detail, the local CV bandwidth search is performed in a grid of bandwidths, which is saved (`hsave = TRUE`) and consists of 200 bandwidths (`hl = 200`) ranging from 0.2 to 2 times the standardized IQR of the covariate (`hbound = c(0.2,2)`). Point and 95% CI estimates of the conditional survival function $S(t|x)$ are computed by `beran()` with the selected bandwidths:

```
x0  <- c(0, 0.5)
hcv <- berancv(x, t, d, data, x0 = x0,
   cvpars = controlpars(hbound = c(0.2, 2), hl = 200, hsave = TRUE))
S <- beran(x, t, d, data, x0 = x0,  h = hcv$h, conflevel = 0.95)
S
```

```
#> Bandwidth type: local
#>
#> Covariate (x0): 0.0 0.5
#> Bandwidth (h):  1.598875 1.104106
#>
#> Beran's conditional survival estimate:
#>
#> x0 = 0
#>        time  survival lower 95% CI upper 95% CI
#> 0.004599127 1.0000000    1.0000000    1.0000000
#> 0.042088293 1.0000000    1.0000000    1.0000000
#> 0.042271452 1.0000000    1.0000000    1.0000000
#> 0.059671372 1.0000000    1.0000000    1.0000000
#> 0.067375891 1.0000000    1.0000000    1.0000000
#> 0.098569312 1.0000000    1.0000000    1.0000000
#> ...
#>
#> x0 = 0.5
#>        time  survival lower 95% CI upper 95% CI
#> 0.004599127 1.0000000    1.0000000    1.0000000
#> 0.042088293 1.0000000    1.0000000    1.0000000
#> 0.042271452 1.0000000    1.0000000    1.0000000
#> 0.059671372 1.0000000    1.0000000    1.0000000
#> 0.067375891 1.0000000    1.0000000    1.0000000
#> 0.098569312 1.0000000    1.0000000    1.0000000
#> ...
```

The next code shows an equivalent way of obtaining the same estimates:

```
S <- beran(x, t, d, data, x0 = x0, conflevel = 0.95,
   cvbootpars = controlpars(hbound = c(0.2, 2), hl = 200, hsave = TRUE))
```

Figure 2 displays point and 95% CI estimates of the survival curve for covariate value 0.5. It has been obtained by executing:

```
plot(S$testim, S$S$x0.5, type = "s", xlab = "Time", ylab = "Survival",
   ylim = c(0, 1))
lines(S$testim, S$conf$x0.5$lower, type = "s", lty = 2)
lines(S$testim, S$conf$x0.5$upper, type = "s", lty = 2)
p0 <- exp(2 * x0[2])/(1 + exp(2 * x0[2]))
lines(S$testim, 1 - p0 + p0 * pweibull(S$testim,
   shape = 0.5 * (x0[2] + 4), scale = 1, lower.tail = FALSE), col = 2)
legend("topright", c("Estimate", "95% CI limits", "True"),
   lty = c(1, 2, 1), col = c(1, 1, 2))
```

Test for enough follow-up

The nonparametric estimators of the cure rate and latency functions given in (1) and (3), respectively, require assumption (4) for their consistency. In other words, the follow-up must be long enough for

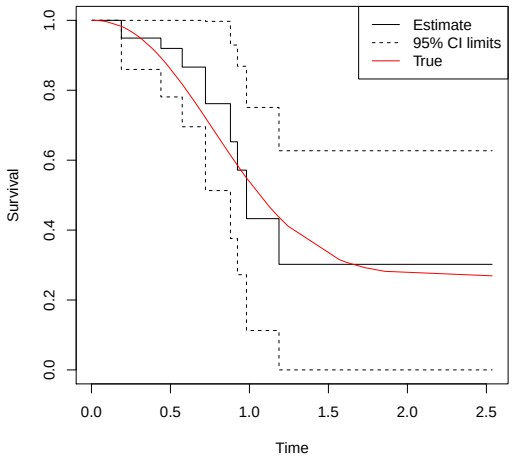


Figure 2: Beran's estimate of the conditional survival function for $x = 0.5$.

cures to happen so that the censored times after the largest uncensored observation can be assumed to correspond to cured subjects.

The procedure to test the hypothesis (4) proposed by Maller and Zhou (1992) is performed by the `testmz()` function:

```
testmz(t, d, dataset)
```

The function returns a list (with class attribute 'npcure') whose main component, containing the p-value of the test, is `pvalue`. The further component `aux` is, in turn, a list of components `statistic`, which contains the test statistic, `n`, the sample size, `delta`, giving the difference between the largest observed time $T_{(n)}$ and the largest uncensored time $T^1_{\max}$, and `interval`, which has the range between $\max(0, T^1_{\max} -$ `delta`$)$ and $T^1_{\max}$.

With our simulated data, the result of the test is:

```
testmz(t, d, data)
```

```
#> Maller-Zhou test
#>
#> statistic  n      p.value
#>        43 50 2.024892e-43
```

Example

To illustrate the nonparametric modeling of the mixture cure model with the **npcure** package, we consider the bone marrow transplantation data in Klein and Moeschberger (2005), available as the `bmt` dataset of the R package **KMsurv** (Klein et al., 2012). The data comes from a multi-center study carried out between 1984 and 1989, involving 137 patients with acute myelocytic leukemia (AML) or acute lymphoblastic leukemia (ALL), aged from 7 to 52. Bone marrow transplant (BMT) is the standard treatment for acute leukemia. Transplantation can be considered a failure when leukemia recurs or the patient dies. Consequently, the failure time is defined as the time (days) to relapse or death. The variables collecting this information are:

- `t2` Disease-free survival time in days (time to relapse, death, or end of study)
- `d3` Disease-free survival indicator (`1`: Dead or relapsed, `0`: Alive and disease-free)

The probability of cure after BMT is high, especially if BMT is performed while the patient remains in the chronic phase (Devergie et al., 1987). Recovery after BMT is a complex process depending on a large set of risk factors, whose status is coded by the following variables:

ta	Time to acute graft-versus-host disease (GVHD).
tc	Time to chronic GVHD.
tp	Time to return of platelets to normal levels.
z1	Patient age (years).
z2	Donor age (years).
z7	Waiting time to transplant (days).
group	Disease group (1: ALL, 2: AML low risk, 3: AML high risk).
da	Acute GVHD indicator (1: Developed, 0: Never developed).
dc	Chronic GVHD indicator (1: Developed, 0: Never developed).
dp	Platelet recovery indicator (1: Returned to normal, 0: Never returned to normal).
z3	Patient gender (1: Male, 0: Female).
z4	Donor gender (1: Male, 0: Female).
z5	Patient cytomegalovirus (CMV) status (1: Positive, 0: Negative).
z6	Donor CMV status (1: Positive, 0: Negative).
z8	FAB (1: FAB grade 4 or 5 and AML, 0: Otherwise).
z9	Hospital (1: Ohio State University, 2: Alferd, 3: St. Vincent, 4: Hahnemann).
z10	Methotrexate (MTX) used for prophylaxis of GVHD (1: Yes, 0: No).

Before applying the estimation methods of the **npcure** package, it should be checked whether the follow-up time was long enough to make it sure that condition (4) holds. This can be subjectively assessed by visualizing a plot of the KM estimate of the unconditional survival function, $S(t)$. The estimated survival curve in Figure 3 suggests the existence of a non-zero asymptote at the right tail. The test of Maller and Zhou (1992) confirms that the follow-up period is adequate to ensure the validity of the nonparametric estimation procedures available in the package:

```
data("bmt", package = "KMsurv")
testmz(t2, d3, bmt)

#> Maller-Zhou test
#>
#> statistic   n      p.value
#>        11 137 1.047242e-05
```

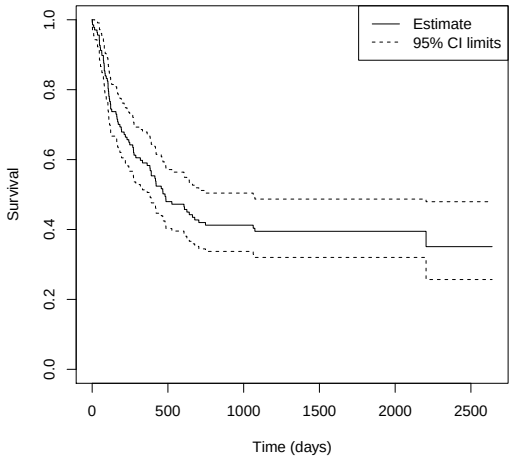


Figure 3: Estimated disease-free survival.

Estimation of the probability of cure

We start by estimating the cure probability as a function of age (z1) and waiting time to transplant (z7), respectively. Cure probabilities are estimated at a grid of 100 values between the 5th and 95th quantiles of the values of z1 and z7. The code for z1 is (for z7, it is similar):

```
x0 <- seq(quantile(bmt$z1, 0.05), quantile(bmt$z1, 0.95), length.out = 100)
q.age <- probcure(z1, t2, d3, bmt, x0 = x0, conflevel = 0.95,
   bootpars = controlpars(hsmooth = 10))
```

Both estimated cure rates are displayed in Figure 4, where a kernel estimate of the covariate density has been added for reference:

```
par(mar = c(5, 4, 4, 5) + 0.1)
plot(q.age$x0, q.age$q, type = "l", ylim = c(0, 1),
   xlab = "Patient age (years)", ylab = "Cure probability")
lines(q.age$x0, q.age$conf$lower, lty = 2)
lines(q.age$x0, q.age$conf$upper, lty = 2)
par(new = TRUE)
d.age <- density(bmt$z1)
plot(d.age, xaxt = "n", yaxt = "n", xlab = "", ylab = "", col = 2,
   main = "", zero.line = FALSE)
mtext("Density", side = 4, col = 2, line = 3)
axis(4, ylim = c(0, max(d.age$y)), col = 2, col.axis = 2)
legend("topright", c("Estimate", "95% CI limits", "Covariate density"),
   lty = c(1, 2, 1), col = c(1, 1, 2), cex = 0.8)
```

The cure probability seems to be nearly constant or, at most, to decrease slightly with patient age and as the waiting time to transplant increases.

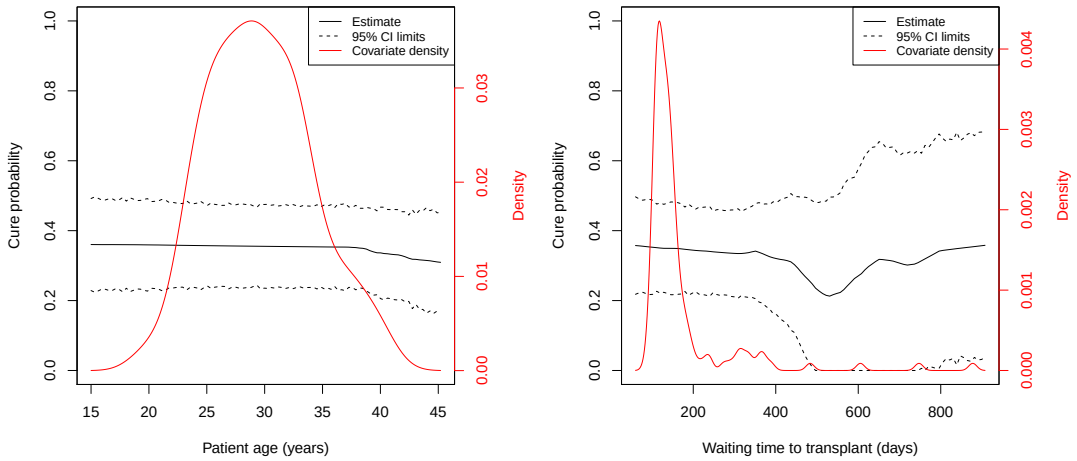


Figure 4: Estimation of the cure probability conditional on age (left panel) and waiting time to transplant (right panel). Nonparametric estimates of the covariate density are plotted for reference.

Testing the effect of one covariate on the probability of cure

The significance of the effect of patient age (`z1`) and waiting time to transplant (`z7`) on the probability of cure can be tested with the `testcov()` function:

```
testcov(z1, t2, d3, bmt,  bootpars = controlpars(B = 2500))
```

```
#> Covariate test
#>
#> Covariate:  z1
#>                test statistic p.value
#>   Cramer-von Mises 0.1103200  0.8204
#> Kolmogorov-Smirnov 0.7308477  0.7900
```

```
testcov(z7, t2, d3, bmt,  bootpars = controlpars(B = 2500))
```

```
#> Covariate test
#>
#> Covariate:  z7
#>                test statistic p.value
#>   Cramer-von Mises 0.7921912  0.0968
#> Kolmogorov-Smirnov 1.6116129  0.1008
```

The effect of age on the cure probability is not statistically significant with neither the CM nor the KS tests ($p_{CM} = 0.820$ and $p_{KS} = 0.790$, where the subscripts identify the p-value in an obvious way). As for the effect of waiting time to transplant, it reaches a borderline significance ($p_{CM} = 0.097$ and $p_{KS} = 0.101$).

Cure probability can also be compared between groups defined by a categorical covariate. We illustrate this case by considering gender (`z3`) and the use of MTX for prophylaxis of GVHD (`z10`). For improving readability, we first label the groups:

```
bmt$z3 <- factor(bmt$z3, labels = c("Male", "Female"))
bmt$z10 <- factor(bmt$z10, labels = c("MTX", "No MTX"))
summary(bmt[, c("z3", "z10")])
```

```
#>      z3         z10
#> Male  :57   MTX   :97
#> Female:80   No MTX:40
```

The estimated survival functions are displayed in Figure 5. The code for gender (`z3`) is:

```
library("survival")
Sgender <- survfit(Surv(t2, d3) ~ z3, data = bmt)
Sgender
```

```
#> Call: survfit(formula = Surv(t2, d3) ~ z3, data = bmt)
#>
#>           n events median 0.95LCL 0.95UCL
#> z3=Male   57     36    318     172      NA
#> z3=Female 80     47    606     418      NA
```

```
plot(Sgender, col = 1:2, mark.time = FALSE, xlab = "Time (days)",
   ylab = "Disease-free survival")
legend("topright", legend = c("Male", "Female"), title = "Gender",
   lty = 1, col = 1:2)
```

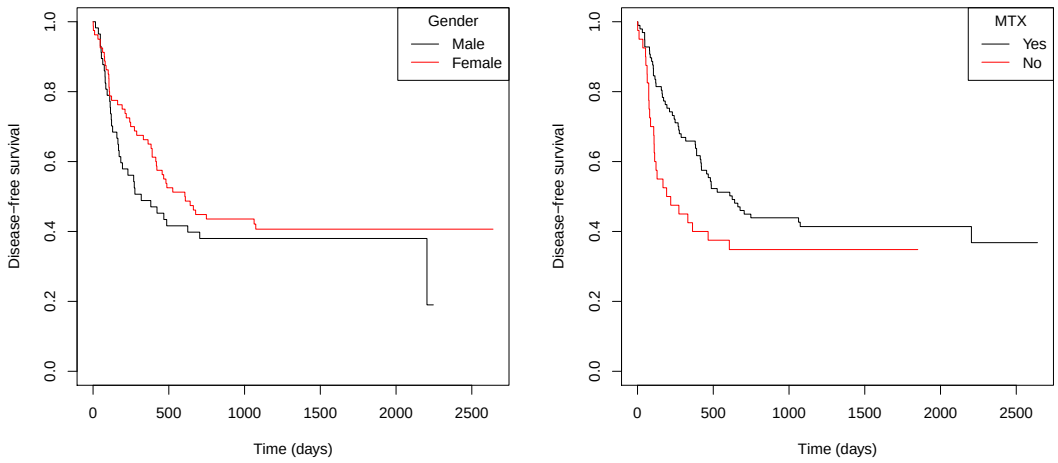


Figure 5: Survival curves of patients conditional on gender (left panel) and use of MTX for prophylaxis of GVHD (right panel).

The estimated probability of cure for each group defined by gender (`z3`) is obtained by computing for each stratum the unconditional cure rate estimator of Laska and Meisner (1992). This estimator of the probability of cure is the value of the KM curve at $T^1_{\max}$ (i.e., it is the minimum of the KM estimate):

```
qgender <- c(min(Sgender[1]$surv), min(Sgender[2]$surv))
qgender
```

```
#> [1] 0.1899671 0.4065833
```

The estimated probability of cure is 19.0% for males and 40.7% for females. The cure probabilities according to the use or not of MTX as GVHD prophylactic (`z10`) are:

```
Smtx <- survfit(Surv(t2, d3) ~ z10, data = bmt)
qmtx <- c(min(Smtx[1]$surv), min(Smtx[2]$surv))
qmtx

#> [1] 0.3679977 0.3482143
```

The cure rate of patients treated with MTX is estimated to be 36.8%, slightly higher than 34.8%, the estimate for patients not treated with MTX.

The effect of these two binary variables on the cure probability is tested with the `testcov()` function similarly as it was done with continuous covariates:

```
testcov(z3, t2, d3, bmt, bootpars = controlpars(B = 2500))

#> Covariate test
#>
#> Covariate:  z3
#>                test statistic p.value
#>   Cramer-von Mises 0.5947305  0.0900
#> Kolmogorov-Smirnov 1.1955919  0.0892

testcov(z10, t2, d3, bmt, bootpars = controlpars(B = 2500))

#> Covariate test
#>
#> Covariate:  z10
#>               test statistic p.value
#>   Cramer-von Mises  1.018441  0.0692
#> Kolmogorov-Smirnov  1.199340  0.0668
```

The differences in the probability of cure between males and females, and between patients with and without MTX treatment are not statistically significant, although a borderline effect is evidenced ($p_{CM} = 0.090$ and $p_{KS} = 0.089$ for gender, $p_{CM} = 0.069$ and $p_{KS} = 0.067$ for MTX).

Estimation of the latency function

The survival of the uncured patients (latency) is estimated for patient age (`z1`) 25 and 40 years as follows:

```
S0 <- latency(z1, t2, d3, bmt, x0 = c(25, 40), conflevel = 0.95,
   bootpars = controlpars(B = 500))
```

Figure 6 displays the survival functions for the two ages, obtained by executing:

```
plot(S0$testim, S0$S$x25, type = "s", ylim = c(0, 1),
   xlab = "Time (days)", ylab = "Latency")
lines(S0$testim, S0$conf$x25$lower, type = "s", lty = 2)
lines(S0$testim, S0$conf$x25$upper, type = "s", lty = 2)
lines(S0$testim, S0$S$x40, type = "s", col = 2)
lines(S0$testim, S0$conf$x40$lower, type = "s", lty = 2, col = 2)
lines(S0$testim, S0$conf$x40$upper, type = "s", lty = 2, col = 2)
legend("topright", c("Age 25: Estimate", "Age 25: 95% CI limits",
   "Age 40: Estimate", "Age 40: 95% CI limits"), lty = 1:2,
   col = c(1, 1, 2, 2))
```

An increased survival of younger patients can be observed, but the survival advantage vanishes after approximately 6 years.

Summary

This paper introduces the **npcure** package. It provides an R implementation of a completely nonparametric approach for estimation in mixture cure models, along with a nonparametric covariate significance test for the cure probability. Moreover, the generalized PL estimator of the conditional survival function with a CV bandwidth selection function is included. Furthermore, the theory underlying the implemented methods, presented in Xu and Peng (2014), López-Cheda et al. (2017a), and López-Cheda et al. (2017b), has been compiled.

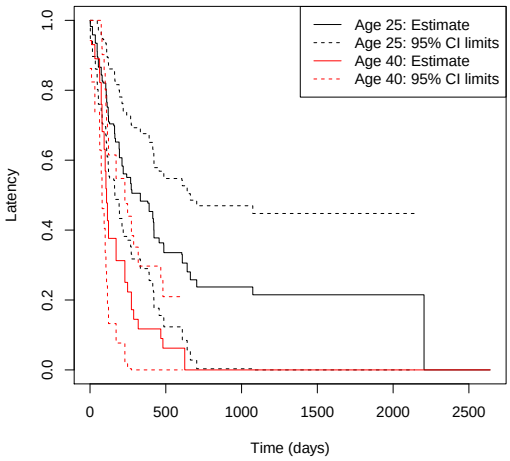


Figure 6: Latency curves of uncured patients 25 and 40 years old.

The **npcure** package has some limitations. Firstly, it only handles right-censored survival times. Left-censored data, truncation, or interval-censored data have not been considered in this approach, and it remains an open problem to be dealt with in the future. Secondly, a conditional estimation can be performed when only one covariate is involved. The same restriction applies to the implemented covariate significance test for the cure rate. An important extension would be the development of estimation and test procedures for the cure rate and latency functions when they depend on a set of covariates. A major challenge is the way the covariates are handled. In that case, the analysis of a large number of covariates would suffer from the curse of dimensionality. Dimension reduction techniques would be required, which leads to a demanding approach that has not been addressed yet, and we leave for further research.

There is an interesting issue that remains an open problem to be dealt with in future versions of the package. Traditional cure rate models implicitly assume that there is no additional information on the cure status of the patients. So, the cure indicator is modeled as a latent variable. However, examples contradicting this assumption can be found. For instance, in some clinical settings, subjects who are followed up beyond a threshold period without experiencing the event can be considered as cured. In other cases, complementary diagnostic tests providing further information about a patient's cure status may be available. We aim to develop improved non-parametric methods of estimation and hypothesis testing that take into account this additional information.

Acknowledgments

The first author's research was sponsored by the Beatriz Galindo Junior Spanish Grant from Ministerio de Ciencia, Innovación y Universidades (MICINN) with reference BGP18/00154. All the authors acknowledge partial support by the MICINN Grant MTM2017-82724-R (EU ERDF support included), and by Xunta de Galicia (Centro Singular de Investigación de Galicia accreditation ED431G/01 2016-2019 and Grupos de Referencia Competitiva CN2012/130 and ED431C2016-015) and the European Union (European Regional Development Fund - ERDF).

Bibliography

J. Amdahl. *flexsurvcure: Flexible Parametric Cure Models*, 2017. URL https://CRAN.R-project.org/package=flexsurvcure. R package version 0.0.2. [p21]

M. Amico and I. Van Keilegom. Cure models in survival analysis. *Annual Review of Statistics and Its Application*, 5:311–342, 2018. URL https://doi.org/10.1146/annurev-statistics-031017-100101. [p21]

R. Beran. *Nonparametric Regression with Randomly Censored Survival Data*. University of California, Berkeley, 1981. [p22, 23, 25, 26, 31]

A. Bertrand, C. Legrand, and I. Van Keilegom. *miCoPTCM: Promotion Time Cure Model with Mis-Measured Covariates*, 2020. URL https://CRAN.R-project.org/package=miCoPTCM. R package version 1.1. [p22]

J. W. Boag. Maximum likelihood estimates of the proportion of patients cured by cancer therapy. *Journal of the Royal Statistical Society B*, 11:15–53, 1949. URL https://doi.org/10.1111/j.2517-6161.1949.tb00020.x. [p21]

J. Brettas. *intercure: Cure Rate Estimators for Interval Censored Data*, 2016. URL https://CRAN.R-project.org/package=intercure. R package version 0.1.0. [p22]

M. Brockmann, T. Gasser, and E. Herrmann. Locally adaptive bandwidth choice for kernel regression estimators. *Journal of the American Statistical Association*, 88:1302–1309, 1993. URL https://doi.org/10.2307/2291270. [p25]

C. Cai, Y. Zou, Y. Peng, and J. Zhang. *smcure: Fit Semiparametric Mixture Cure Models*, 2012. URL https://CRAN.R-project.org/package=smcure. R package version 2.0. [p21]

C. Cai, S. Wang, W. Lu, and J. Zhang. *NPHMC: Sample Size Calculation for the Proportional Hazards Mixture Cure Model*, 2013. URL https://CRAN.R-project.org/package=NPHMC. R package version 2.2. [p21]

R. Cao and W. González-Manteiga. Bootstrap methods in regression smoothing. *Journal of Nonparametric Statistics*, 2:379–388, 1993. URL https://doi.org/10.1080/10485259308832566. [p25]

D. Dabrowska. Variable bandwidth conditional Kaplan-Meier estimate. *Scandinavian Journal of Statistics*, 19:351–361, 1992. [p22]

M. A. Delgado and W. González-Manteiga. Significance testing in nonparametric regression based on the bootstrap. *Annals of Statistics*, 29:1469–1507, 2001. URL https://doi.org/10.1214/aos/1013203462. [p25, 26]

J. W. Denham, E. Denham, K. B. Dear, and G. V. Hudson. The follicular non-Hodgkin's lymphomas - I. the possibility of cure. *European Journal of Cancer*, 32:470–479, 1996. URL https://doi.org/10.1016/0959-8049(95)00607-9. [p21]

A. Devergie, E. Gluckman, F. Varrin, J. L. Huret, J. Meletis, H. D. Castro, D. Bombail, E. Vilmer, R. Traineau, and M. Boiron. La greffe de moelle osseuse allogénique dans la leucémie myéloïde chronique. *Nouvelle Revue Française d'Hématologie*, 29:69–72, 1987. [p33]

V. T. Farewell. The use of mixture models for the analysis of survival data with long-term survivors. *Biometrics*, 38:1041–1046, 1982. URL https://doi.org/10.2307/2529885. [p21]

V. T. Farewell. Mixture models in survival analysis: Are they worth the risk? *Canadian Journal of Statistics*, 14:257–262, 1986. URL https://doi.org/10.2307/3314804. [p21]

A. Gannoun, J. Saracco, and K. Yu. Comparison of kernel estimators of conditional distribution function and quantile regression under censoring. *Statistical Modelling*, 7:329–344, 2007. URL https://doi.org/10.1177/1471082X0700700404. [p22]

C. Geerdens, E. F. Acar, and P. Janssen. Conditional copula models for right-censored clustered event time data. *Biostatistics*, 19:247–262, 2017. URL https://doi.org/10.1093/biostatistics/kxx034. [p22, 25, 31, 32]

T. A. Gerds. *prodlim: Product-Limit Estimation for Censored Event History Analysis*, 2018. URL https://CRAN.R-project.org/package=prodlim. R package version 2018.04.18. [p22]

X. Han, Y. Zhang, and Y. Shao. *rcure: Robust Cure Models for Survival Analysis*, 2017. URL https://CRAN.R-project.org/package=rcure. R package version 0.1.0. [p22]

M. C. Iglesias-Pérez. Comparación de dos selectores de la ventana en la estimación de la distribución condicional con censura. In SGAPEIO, editor, *Proceedings of the IX Congreso Galego de Estatística e Investigación de Operacións*. Sociedade Galega para a Promoción da Estatística e Investigación de Operacións, 2009. URL http://sidor.uvigo.es/ixsgapeio/resumenes/81_29_paper.pdf. [p22]

J. D. Kalbfleisch and R. L. Prentice. *The Statistical Analysis of Failure Time Data*. Wiley, New York, 2nd edition, 2002. ISBN 978-0-471-36357-6. [p21]

E. L. Kaplan and P. Meier. Nonparametric estimation from incomplete observations. *Journal of the American Statistical Association*, 53:458–481, 1958. URL https://doi.org/10.2307/2281868. [p22]

J. P. Klein and M. L. Moeschberger. *Survival Analysis: Techniques for Censored and Truncated Data*. Springer-Verlag, New York, 2nd edition, 2005. ISBN 978-0-387-21645-4. [p33]

J. P. Klein, M. L. Moeschberger, and J. Yan. *KMsurv: Data sets from Klein and Moeschberger (1997), Survival Analysis*, 2012. URL https://CRAN.R-project.org/package=KMsurv. R package version 0.1-5. [p33]

A. Y. C. Kuk and C. H. Chen. A mixture model combining logistic regression with proportional hazards regression. *Biometrika*, 79:531–541, 1992. URL https://doi.org/10.1093/biomet/79.3.531. [p21]

E. M. Laska and M. J. G. Meisner. Nonparametric estimation and testing in a cure model. *Biometrics*, 48:1223–1234, 1992. URL https://doi.org/10.2307/2532714. [p22, 23, 36]

C. Li and J. M. G. Taylor. A semi-parametric accelerated failure time cure model. *Statistics in Medicine*, 21:3235–3247, 2002. URL https://doi.org/10.1002/sim.1260. [p21]

G. Li and S. Datta. A bootstrap approach to nonparametric regression for right censored data. *Annals of the Institute of Statistical Mathematics*, 53:708–729, 2001. URL https://doi.org/10.1023/A:1014644700806. [p24, 25]

A. López-Cheda, R. Cao, M. A. Jácome, and I. Van Keilegom. Nonparametric incidence estimation and bootstrap bandwidth selection in mixture cure models. *Computational Statistics & Data Analysis*, 105:144–165, 2017a. URL https://doi.org/10.1016/j.csda.2016.08.002. [p22, 23, 24, 25, 37]

A. López-Cheda, M. A. Jácome, and R. Cao. Nonparametric latency estimation for mixture cure models. *TEST*, 26:353–376, 2017b. URL https://doi.org/10.1007/s11749-016-0515-1. [p22, 23, 24, 25, 30, 37]

A. López-Cheda, M. A. Jácome, I. Van Keilegom, and R. Cao. Nonparametric covariate hypothesis tests for the cure rate in mixture cure models. *Statistics in Medicine*, 39:2291–2307, 2020. URL https://doi.org/10.1002/sim.8530. [p25, 26]

I. López-de-Ullibarri, A. López-Cheda, and M. A. Jácome. *npcure: Nonparametric Estimation in Mixture Cure Models*, 2020. URL https://CRAN.R-project.org/package=npcure. R package version 0.1-5. [p22]

R. A. Maller and S. Zhou. Estimating the proportion of immunes in a censored sample. *Biometrika*, 79:731–739, 1992. URL https://doi.org/10.1093/biomet/79.4.731. [p23, 24, 26, 27, 33, 34]

R. A. Maller and S. Zhou. *Survival Analysis with Long-Term Survivors*. Wiley, Chichester, U. K., 1996. URL https://doi.org/10.1002/cbm.318. [p21]

L. Meira-Machado and M. Sestelo. condsurv: An R package for the estimation of the conditional survival function for ordered multivariate failure time data. *The R Journal*, 8(2):460–473, 2016. URL https://doi.org/10.32614/RJ-2016-059. [p22]

L. Meira-Machado, M. Sestelo, and G. Soutinho. *survidm: Inference and Prediction in an Illness-Death Model*, 2019. URL https://CRAN.R-project.org/package=survidm. R package version 1.2.0. [p22]

U. U. Müller and I. Van Keilegom. Goodness-of-fit tests for the cure rate in a mixture cure model. *Biometrika*, 106:211–227, 2019. URL https://doi.org/10.1093/biomet/asy058. [p21, 25]

Y. Niu and Y. Peng. *geecure: Marginal Proportional Hazards Mixture Cure Models with Generalized Estimating Equations*, 2018. URL https://CRAN.R-project.org/package=geecure. R package version 1.0-6. [p22]

Y. Peng. *mixcure: Mixture Cure Models*, 2020. URL https://CRAN.R-project.org/package=mixcure. R package version 2.0. [p22]

Y. Peng and K. B. Dear. A nonparametric mixture model for cure rate estimation. *Biometrics*, 56:237–243, 2000. URL https://doi.org/10.1111/j.0006-341X.2000.00237.x. [p21]

Y. Peng, K. B. Dear, and J. W. Denham. A generalized F mixture model for cure rate estimation. *Statistics in Medicine*, 17:813–830, 1998. URL https://doi.org/10.1002/(SICI)1097-0258(19980430)17:8<813::AID-SIM775>3.0.CO;2-%23. [p21]

J. Racine and Q. Li. Nonparametric estimation of regression functions with both categorical and continuous data. *Journal of Econometrics*, 119:99–130, 2004. URL https://doi.org/10.1016/S0304-4076(03)00157-X. [p27]

J. Xu and Y. Peng. Nonparametric cure rate estimation with covariates. *Canadian Journal of Statistics*, 42:1–17, 2014. URL https://doi.org/10.1002/cjs.11197. [p22, 23, 24, 27, 37]

J. Zhou, J. Zhang, and W. Lu. Computationally efficient estimation for the generalized odds rate mixture cure model with interval-censored data. *Journal of Computational and Graphical Statistics*, 27: 48–58, 2017. URL https://doi.org/10.1080/10618600.2017.1349665. [p22]

Ana López-Cheda
Research Group MODES, CITIC, Departamento de Matemáticas, Facultade de Informática, Universidade da Coruña
CITIC, Campus de Elviña s/n, A Coruña 15071
Spain
(ORCiD: 0000-0002-3618-3246)
ana.lopez.cheda@udc.es

M. Amalia Jácome
Research Group MODES, CITIC, Departamento de Matemáticas, Facultade de Ciencias, Universidade da Coruña
Rúa da Fraga s/n, A Zapateira, A Coruña 15071
Spain
(ORCiD: 0000-0001-7000-9623)
maria.amalia.jacome@udc.es

Ignacio López-de-Ullibarri
Research Group MODES, Departamento de Matemáticas, Escuela Universitaria Politécnica, Universidade da Coruña
15405, Ferrol, A Coruña
Spain
(ORCiD: 0000-0002-3438-6621)
ignacio.lopezdeullibarri@udc.es

A Method for Deriving Information from Running R Code

by Mark P.J. van der Loo

Abstract It is often useful to tap information from a running R script. Obvious use cases include monitoring the consumption of resources (time, memory) and logging. Perhaps less obvious cases include tracking changes in R objects or collecting the output of unit tests. In this paper, we demonstrate an approach that abstracts the collection and processing of such secondary information from the running R script. Our approach is based on a combination of three elements. The first element is to build a customized way to evaluate code. The second is labeled *local masking* and it involves temporarily masking a user-facing function so an alternative version of it is called. The third element we label *local side effect*. This refers to the fact that the masking function exports information to the secondary information flow without altering a global state. The result is a method for building systems in pure R that lets users create and control secondary flows of information with minimal impact on their workflow and no global side effects.

Introduction

The R language provides a convenient language to read, manipulate, and write data in the form of scripts. As with any other scripted language, an R script gives a description of data manipulation activities, one after the other, when read from top to bottom. Alternatively, we can think of an R script as a one-dimensional visualization of data flowing from one processing step to the next, where intermediate variables or pipe operators carry data from one treatment to the next.

We run into limitations of this one-dimensional view when we want to produce data flows that are somehow 'orthogonal' to the flow of the data being treated. For example, we may wish to follow the state of a variable while a script is being executed, report on progress (logging), or keep track of resource consumption. Indeed, the sequential (one-dimensional) nature of a script forces one to introduce extra expressions between the data processing code.

As an example, consider a code fragment where the variable `x` is manipulated.

```
x[x > threshold] <- threshold
x[is.na(x)]  <- median(x, na.rm=TRUE)
```

In the first statement, every value above a certain threshold is replaced with a fixed value, and next, missing values are replaced with the median of the completed cases. It is interesting to know how an aggregate of interest, say the mean of `x`, evolves as it gets processed. The instinctive way to do this is to edit the code by adding statements to the script that collect the desired information.

```
meanx <- mean(x, na.rm=TRUE)
x[x > threshold] <- threshold
meanx <- c(meanx, mean(x, na.rm=TRUE))
x[is.na(x)]  <- median(x, na.rm=TRUE)
meanx <- c(meanx, mean(x, na.rm=TRUE))
```

This solution clutters the script by inserting expressions that are not necessary for its main purpose. Moreover, the tracking statements are repetitive, which validates some form of abstraction.

A more general picture of what we would like to achieve is given in Figure 1. The 'primary data flow' is developed by a user as a script. In the previous example, this concerns processing `x`. When the script runs, some kind of logging information, which we label the 'secondary data flow' is derived implicitly by an abstraction layer.

Creating an abstraction layer means that concerns between primary and secondary data flows are separated as much as possible. In particular, we want to prevent the abstraction layer from inspecting or altering the user code that describes the primary data flow. Furthermore, we would like the user to have some control over the secondary flow from within the script, for example, to start, stop, or parameterize the secondary flow. This should be done with minimum editing of the original user code, and it should not rely on global side effects. This means that neither the user nor the abstraction layer for the secondary data flow should have to manipulate or read global variables, options, or other environmental settings to convey information from one flow to the other. Finally, we want to treat the availability of a secondary data flow as a normal situation. This means we wish to avoid using signaling conditions (e.g., warnings or errors) to convey information between the flows unless there is an actual exceptional condition such as an error.

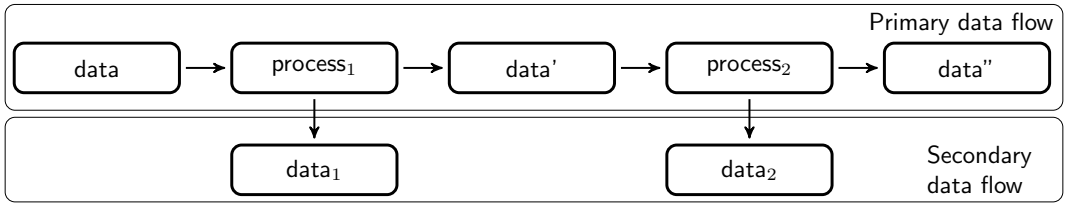


Figure 1: Primary and secondary data flows in an R script. The primary flow follows the execution of an R script, while in the background a secondary data flow (e.g. logging information) is created.

Prior art

There are several packages that generate a secondary data flow from a running script. One straightforward application concerns logging messages that report on the status of a running script. To create a logging message, users edit their code by inserting logging expressions where desired. Logging expressions are functions calls that help to build expressions, for example, by automatically adding a timestamp. Configuration options usually include a measure of logging verbosity and setting an output channel that controls where logging data will be sent. Changing these settings relies on communication from the main script to the functionality that controls the flow of logging data. In **logger** (Daróczi, 2021), this is done by manipulating a variable stored in the package namespace using special helper functions. The **logging** package (Frasca, 2019) also uses an environment within the namespace of the package to manage option settings, while **futile.logger** (Rowe, 2016) implements a custom global option settings manager that is somewhat comparable to R's own `options()` function.

Packages **bench** (Hester, 2020b) and **microbenchmark** (Mersmann, 2019) provide time profiling of single R expressions. The **bench** package also includes memory profiling. Their purpose is not to derive a secondary data flow from a running production script as in Figure 1 but to compare the performance of R expressions. Both packages export a function that accepts a sequence of expressions to profile. These functions take control of expression execution and insert time and/or memory measurements where necessary. Options, such as the number of times each expression is executed, are passed directly to the respective function.

Unit testing frameworks provide another source of secondary data flows. Here, an R script is used to prepare, set up, and compare test data, while the results of comparisons are tapped and reported. Testing frameworks are provided by **testthat** (Wickham, 2011), **RUnit**, (Burger et al., 2018), **testit** Xie (2021), **unitizer** (Gaslam, 2021), and **tinytest** (van der Loo, 2020). The first three packages (**testthat**, **RUnit**, and **testit**) all export assertion functions that generate condition signals to convey information about test results. Packages **RUnit** and **testit** use `sys.source()` to run a file containing unit test assertions and exit on the first error while **testthat** uses `eval()` to run expressions, capture conditions, and test results and reports afterward. The **unitizer** framework is different because it implements an interactive prompt to run tests and explore their results. Rather than providing explicit assertions, **unitizer** stores results of all expressions that return a visible result and compares their output at subsequent runs. Interestingly, **unitizer** allows for optional monitoring of the testing environment. This includes environment variables, options, and more. This is done by manipulating the code of (base) R functions that manage these settings and masking the original functions temporarily. These masking functions then provide parts of the secondary data flow (changes in the environment). Finally, **tinytest** is based on the approach that is the topic of this paper, and it will be discussed as an application below.

Finally, we note the **covr** package of Hester (2020a). This package is used to keep track of which expressions of an R package are run (covered) by package tests or examples. In this case, the primary data flow is a test script executing code (functions, methods) stored in another script, usually in the context of a package. The secondary flow consists of counts of how often each expression in the source is executed. The package works by parsing and altering the code in the source file, inserting expressions that increase appropriate counters. These counters are stored in a variable that is part of the package's namespace.

Summarizing, we find that in logging packages, the secondary data flow is invoked explicitly by users while configuration settings are communicated by manipulating a global state that may or may not be directly accessible by the user. For benchmarking packages, the expressions are passed explicitly to an 'expression runner' that monitors the effect on memory and passage of time. In most test packages, the secondary flow is invoked explicitly using special assertions that throw condition signals. Test files are run using functionality that captures and administrates signals where necessary. Two of the discussed packages explicitly manipulate existing code before running it to create a secondary data flow. The **covr** package does this to update expression counters and the

unitizer package to monitor changes in the global state.

Contribution of this paper

The purpose of this paper is to first provide some insight into the problem of managing multiple data flows, independent of specific applications. In the following section, we discuss managing a secondary data stream from the point of view of changing the way in which expressions are combined and executed by R.

Next, we highlight two programming patterns that allow one to derive a secondary data stream, both in non-interactive (while executing a file) and in interactive circumstances. The methods discussed here do not require explicit inspection or modification of the code that describes the primary data flow. It is also not necessary to invoke signaling conditions to transport information from or to the secondary data stream.

We also demonstrate a combination of techniques that allow users to parameterize the secondary flow without resorting to global variables, global options, or variables within a package's namespace. We call this technique 'local masking' with 'local side effects'. It is based on temporarily and locally masking a user-facing function with a function that does exactly the same except for a side effect that passes information to the secondary data flow.

As examples, we discuss two applications where these techniques have been implemented. The first is the **lumberjack** package (van der Loo, 2021), which allows for tracking changes in R objects as they are manipulated expression by expression. The second is **tinytest** (van der Loo, 2020), a compact and extensible unit testing framework.

Finally, we discuss some advantages and limitations to the techniques proposed.

Concepts

In this section we give a high-level overview of the problem of adding a second data flow to an existing one, and general way to think about a solution. The general approach was inspired by a discussion of Milewski (2018) and is related to what is sometimes called a *bind operator* in functional programming.

Consider as an example the following two expressions, labeled e_1 and e_2.

```
e1:  x <- 10
e2:  y <- 2*x
```

We would like to implement some kind of monitoring as these expressions are evaluated. For this purpose, it is useful to think of e_1 and e_2 as functions that accept a set of key-value pairs, possibly alter the set's contents, and return it. In R this set of key-value pairs is an `environment`, and usually, it is the global environment (the user's workspace). Starting with an empty environment $\{\}$ we get:

$$e_1(\{\}) = \{(\texttt{"x"}, \texttt{10})\}$$
$$e_2(e_1(\{\})) = \{(\texttt{"x"}, \texttt{10}), (\texttt{"y"}, \texttt{20})\}$$

In this representation, we can write the result of executing the above script in terms of the function composition operator $\circ$:

$$e_2(e_1(\{\})) = (e_2 \circ e_1)(\{\}).$$

And in general, we can express the final state $\mathcal{U}$ of any environment after executing a sequence of expressions $e_1, e_2, \cdots, e_k$ as:

$$\mathcal{U} = (e_k \circ e_{k-1} \circ \cdots \circ e_1)(\{\}), \tag{1}$$

where we assumed without loss of generality that we start with an empty environment. We will refer to the sequence $e_1 \ldots e_k$ as the 'primary expressions' since they define a user's main data flow.

We now wish to introduce some kind of logging. For example, we want to count the number of evaluated expressions, not counting the expressions that will perform the count. The naive way to do this is to introduce a new expression, say n:

```
n:   if (!exists("N")) N <- 1 else N <- N + 1
```

And we insert this into the original sequence of expressions. This amounts to the cumbersome solution:

$$\mathcal{U} \cup \{(\texttt{"N"}, k)\} = (n \circ e_k \circ n \circ e_{k-1} \circ n \circ \cdots n \circ e_1)(\{\}), \tag{2}$$

where the number of executed expressions is stored in `N`. We shall refer to `n` as a 'secondary expression', as it does not contribute to the user's primary data flow.

The above procedure can be simplified if we define a new function composition operator $\circ_n$ as follows:

$$a \circ_n b = a \circ n \circ b.$$

One may verify the associativity property $a \circ_n (b \circ_n c) = (a \circ_n b) \circ_n c$ for expressions a, b, and c, so $\circ_n$ can, indeed, be interpreted as a new function composition operator. Using this operator we get

$$\mathcal{U} \cup \{(\texttt{"N"}, k-1)\} = (e_k \circ_n e_{k-1} \circ_n \cdots \circ_n e_1)(\{\}), \tag{3}$$

which gives the same result as Equation 2 up to a constant.

If we are able to alter function composition, then this mechanism can be used to track all sorts of useful information during the execution of $e_1, \ldots, e_k$. For example, a simple profiler is set up by timing the expressions and adding the following expression to the function composition operator.

```
s: if (!exists("S")) S <- Sys.time() else S <- c(S, Sys.time())
```

After running $e_k \circ_s \cdots \circ_s e_1$, `diff(S)` gives the timings of individual statements. A simple memory profiler is defined as follows.

```
m: if (!exists("M")) M <- sum(memory.profile()) else M <- c(M, sum(memory.profile()))
```

After running $e_k \circ_m \cdots \circ_m e_1$, `M` gives the amount of memory used by R after each expression.

We can also track changes in data, but it requires that the composition operator knows the name of the R object that is being tracked. As an example, consider the following primary expressions.

```
e1:  x <- rnorm(10)
e2:  x[x<0] <- 0
e3:  print(x)
```

We can define the following expression for our modified function composition operator.

```
v:  {
      if (!exists("V")){
        V <- logical(0)
        x0 <- x
      }
      if (identical(x0,x)) V <- c(V, FALSE)
      else V <- c(V, TRUE)
      x0 <- x
    }
```

After running $e_3 \circ_v e_2 \circ_v e_1$, the variable `V` equals `c(TRUE,FALSE)`, indicating that e_2 changed `x`, and e_3 did not.

These examples demonstrate that redefining function composition yields a powerful method for extracting logging information with (almost) no intrusion on the user's common workflow. The simple model shown here does have some obvious setbacks: first, the expressions inserted by the composition operator manipulate the same environment as the user expressions. The user- and secondary expressions can therefore interfere with each other's results. Second, there is no direct control from the primary sequence over the secondary sequence: the user has no explicit control over starting, stopping, or parametrizing the secondary data stream. We demonstrate in the next section how these setbacks can be avoided by evaluating secondary expressions in a separate environment and by using techniques we call 'local masking' and 'local side-effect'.

Creating a secondary data flow with R

R executes expressions one by one in a read-evaluate-print loop (REPL). In order to tap information from this running loop, it is necessary to catch the user's expressions and interweave them with our own expressions. One way to do this is to develop an alternative to R's native `source()` function. Recall that `source()` reads an R script and executes all expressions in the global environment. Applications include non-interactive sessions or interactive sessions with repetitive tasks such as running test scripts while developing functions. A second way to intervene with a user's code is to develop a special 'forward pipe' operator, akin to R's `|>` pipe, the **magrittr** pipe of Bache and Wickham (2014), or

the 'dot-pipe' of Mount and Zumel (2018). Since a user inserts a pipe between expressions, it is an obvious place to insert code that generates a secondary data flow.

In the following two subsections we will develop both approaches. As a running example, we will implement a secondary data stream that counts expressions.

Build your own `source()`

The `source()` function reads an R script and executes all expressions in the global environment. A simple variant of `source()` that counts expressions as they get evaluated can be built using `parse()` and `eval()`.

```
run <- function(file){
  expressions <- parse(file)
  runtime <- new.env(parent=.GlobalEnv)

  n <- 0
  for (e in expressions){
    eval(e, envir=runtime)
    n <- n + 1
  }
  message(sprintf("Counted %d expressions",n))
  runtime
}
```

Here, `parse()` reads the R file and returns a list of expressions (technically, an object of class 'expression'). The `eval()` function executes the expression while all variables created by or needed for execution are sought in a newly created environment called `runtime`. We make sure that variables and functions in the global environment are found by setting the parent of `runtime` equal to `.GlobalEnv`. Now, given a file `"script.R"`.

```
# contents of script.R
x <- 10
y <- 2*x
```

An interactive session would look like this.

```
> e <- run("script.R")
Counted 2 expressions
> e$x
[1] 10
```

So, contrary to the default behavior of `source()`, variables are assigned in a new environment. This difference in behavior can be avoided by evaluating expressions in `.GlobalEnv`. However, for the next step, it is important to have a separate runtime environment.

We now wish to give the user some control over the secondary data stream. In particular, we want the user to be able to choose when `run()` starts counting expressions. Recall that we demand that this is done by direct communication to `run()`. This means that side-effects such as setting a special variable in the global environment or a global option is out of the question. Furthermore, we want to avoid code inspection: the `run()` function should be unaware of what expressions it is running exactly. We start by writing a function for the user that returns `TRUE`.

```
start_counting <- function() TRUE
```

Our task is to capture this output from `run()` when `start_counting()` is called. We do this by masking this function with another function that does exactly the same, except that it also copies the output value to a place where `run()` can find it. To achieve this, we use the following helper function.

```
capture <- function(fun, envir){
  function(...){
    out <- fun(...)
    envir$counting <- out
    out
  }
}
```

This function accepts a function (`fun`) and an environment (`envir`). It returns a function that first executes `fun(...)`, copies its output value to `envir`, and then returns the output to the user. In an interactive session, we would see the following.

```
> store <- new.env()
> f <- capture(start_counting, store)
> f()
[1] TRUE
> store$counting
[1] TRUE
```

Observe that our call to `f()` returns `TRUE` as expected but also exported a copy of `TRUE` into `store`. The reason this works is that an R function 'remembers' where it is created. The function `f()` was created inside `capture()`, and the variable `envir` is present there. We say that this 'capturing' version of `start_counting` has a *local side-effect*: it writes outside of its own scope but the place where it writes is controlled.

We now need to make sure that `run()` executes the captured version of `start_counting()`. This is done by locally masking the user-facing version of `start_counting()`. That is, we make sure that the captured version is found by `eval()` and not the original version. A new version of `run()` now looks as follows.

```
run <- function(file){
  expressions <- parse(file)
  store <- new.env()
  runtime <- new.env(parent=.GlobalEnv)
  runtime$start_counting <- capture(start_counting, store)
  n <- 0
  for (e in expressions){
    eval(e, envir=runtime)
    if ( isTRUE(store$counting) ) n <- n + 1
  }
  message(sprintf("Counted %d expressions",n))
  runtime
}
```

Now, consider the following code, stored in `script1.R`.

```
# contents of script1.R
x <- 10
start_counting()
y <- 2*x
```

In an interactive session, we would see this.

```
> e <- run("script1.R")
Counted 1 expressions
> e$x
[1] 10
> e$y
[1] 20
```

Let us go through the most important parts of the new `run()` function. After parsing the R file, a new environment is created that will store the output of calls to `start_counting()`.

```
store  <- new.env()
```

The runtime environment is created as before, but now we add the capturing version of `start_counting()`.

```
runtime <- new.env(parent=.GlobalEnv)
runtime$start_counting <- capture(start_counting, store)
```

This ensures that when the user calls `start_counting()`, the capturing version is executed. We call this technique *local masking* since the `start_counting()` function is only masked during the execution of `run()`. The captured version of `start_counting()`as a side effect stores its output in `store`. We call this a 'local side-effect' because `store` is never seen by the user: it is created inside `run()` and destroyed when `run()` is finished.

Finally, all expressions are executed in the runtime environment and counted conditional on the value of `store$counting`.

```
for (e in expressions){
  eval(e, envir=runtime)
  if ( isTRUE(store$counting) ) n <- n + 1
}
```

Summarizing, with this construction, we are able to create a file runner akin to `source()` that can gather and communicate useful process metadata while executing a script. Moreover, the user of the script can convey information directly to the file runner, while it runs, without relying on global side-effects. This is achieved by first creating a user-facing function that returns the information to be sent to the file runner. The file runner locally masks the user-facing version with a version that copies the output to an environment local to the file runner before returning the output to the user.

The approach just described can be generalized to more realistic use cases. All examples mentioned in the 'Context' section —time or memory profiling, or logging changes in data, merely need some extra administration. Furthermore, the current example emits the secondary data flow as a '`message`'. In practical use cases, it may make more sense to write the output to a file connection or database or the make the secondary data stream output of the file runner. In the Applications section, both applications are discussed.

Build your own pipe operator

Pipe operators have become a popular tool for R users over the last years, and R currently has a pipe operator (`|>`) built-in. This pipe operator is intended as a form of 'syntactic sugar' that, in some cases, makes code a little easier to write. A pipe operator behaves somewhat like a left-to-right 'expression composition operator'. This, in the sense that a sequence of expressions that are joined by a pipe operator are interpreted by R's parser as a single expression. Pipe operators also offer an opportunity to derive information from a running sequence of expressions.

It is possible to implement a basic pipe operator as follows.

```
`%p>%` <- function(lhs, rhs) rhs(lhs)
```

Here, the `rhs` (right-hand side) argument must be a single-argument function, which is applied to `lhs`. In an interactive session we could see this.

```
> 3 %p>% sin %p>% cos
[1] 0.9900591
```

To build our expression counter, we need to have a place to store the counter value hidden from the user. In contrast to the implementation of the file runner in the previous section, each use of `%p>%` is disconnected from the other, and there seems to be no shared space to increase the counter at each call. The solution is to let the secondary data flow travel with the primary flow by adding an attribute to the data. We create two user-facing functions that start or stop logging as follows.

```
start_counting <- function(data){
  attr(data, "n") <- 0
  data
}
end_counting <- function(data){
  message(sprintf("Counted %d expressions", attr(data,"n")-1))
  attr(data, "n") <- NULL
  data
}
```

Here, the first function attaches a counter to the data and initializes it to zero. The second function reports its value, decreased by one, so the stop function itself is not included in the count. We also alter the pipe operator to increase the counter if it exists.

```
`%p>%` <- function(lhs, rhs){
  if ( !is.null(attr(lhs,"n")) ){
    attr(lhs,"n") <- attr(lhs,"n") + 1
  }
  rhs(lhs)
}
```

In an interactive session, we could now see the following.

```
> out <- 3 %p>%
+  start_counting %p>%
+    sin %p>%
+    cos %p>%
+  end_counting
Counted 2 expressions
```

```
> out
[1] 0.9900591
```

Summarizing, for small interactive tasks, a secondary data flow can be added to the primary one by using a special kind of pipe operator. Communication between the user and the secondary data flow is implemented by adding or altering attributes attached to the R object.

Generalizations of this technique come with a few caveats. First, the current pipe operator only allows right-hand side expressions that accept a single argument. Extension to a more general case involves inspection and manipulation of the right-hand side's abstract syntax tree and is out of scope for the current work. Second, the current implementation relies on the right-hand side expressions to preserve attributes. A general implementation will have to test that the output of rhs(lhs) still has the logging attribute attached (if there was any) and re-attach it if necessary.

Application 1: tracking changes in data

The **lumberjack** package (van der Loo, 2021) implements a logging framework to track changes in R objects as they get processed. The package implements both a pipe operator, denoted `%L>%`, and a file runner called `run_file()`. The main communication devices for the user are two functions called `start_log()` and `dump_log()`.

We will first demonstrate working with the **lumberjack** pipe operator. The function `start_log()` accepts an R object and a logger object. It attaches the logger to the R object and returns the augmented R object. A logger is a reference object[1] that exposes at least an `$add()` method and a `$dump()` method. If a logger is present, the pipe operator stores a copy of the left-hand side. Next, it executes the expression on the right-hand side with the left-hand side as an argument and stores the output. It then calls the `add()` method of the logger with the input and output so that the logger can compute and store the difference. The `dump_log()` function accepts an R object, calls the `$dump()` method on the attached logger (if there is any), removes the logger from the object and returns the object. An interactive session could look as follows.

```
> library(lumberjack)
> out <- women %L>%
>   start_log(simple$new()) %L>%
>   transform(height = height * 2.54) %L>%
>   identity() %L>%
>   dump_log()
Dumped a log at /home/mark/simple.csv
> read.csv("simple.csv")
  step               time                         expression changed
1    1 2019-08-09 11:29:06 transform(height = height * 2.54)    TRUE
2    2 2019-08-09 11:29:06                         identity()   FALSE
```

Here, `simple$new()` creates a logger object that registers whether an R object has changed or not. There are other loggers that compute more involved differences between in- and output. The `$dump()` method of the logger writes the logging output to a csv file.

For larger scripts, a file runner called `run_file()` is available in **lumberjack**. As an example, consider the following script. It converts columns of the built-in `women` data set to SI units (meters and kilogram) and then computes the body-mass index of each case.

```
# contents of script2.R
start_log(women, simple$new())
women$height <- women$height * 2.54/100
women$weight <- women$weight * 0.453592
women$bmi    <- women$weight/(women$height)^2
```

In an interactive session, we can run the script and access both the logging information and retrieve the output of the script.

```
> e <- run_file("script2.R")
Dumped a log at /home/mark/women_simple.csv
> read.csv("women_simple.csv")
  step               time                                  expression changed
1    1 2019-08-09 13:11:25            start_log(women, simple$new())   FALSE
```

[1] A native R Reference Class, an 'R6' object (Chang, 2020), or any other reference type object implementing the proper API.

```
2    2 2019-08-09 13:11:25    women$height <- women$height * 2.54/100    TRUE
3    3 2019-08-09 13:11:25    women$weight <- women$weight * 0.453592    TRUE
4    4 2019-08-09 13:11:25 women$bmi <- women$weight/(women$height)^2    TRUE
> head(e$women,3)
  height   weight      bmi
1 1.4732 52.16308 24.03476
2 1.4986 53.07026 23.63087
3 1.5240 54.43104 23.43563
```

The **lumberjack** file runner locally masks `start_log()` with a function that stores the logger and the name of the tracked R object in a local environment. A copy of the tracked object is stored locally as well. Expressions in the script are executed one by one. After each expression, the object in the runtime environment is compared with the stored object. If it has changed, the `$add()` method of the logger is called, and a copy of the changed object is stored. After all expressions have been executed, the `$dump()` method is called, so the user does not have to do this explicitly.

A user can add multiple loggers for each R object and track multiple objects. It is also possible to dump specific logs for specific objects during the script. All communication necessary for these operations runs via the mechanism explained in the 'build your own `source()`' section.

Application 2: unit testing

The **tinytest** package (van der Loo, 2020) implements a unit testing framework. Its core function is a file runner that uses local masking and local side effects to capture the output of assertions that are inserted explicitly by the user. As an example, we create tests for the following function.

```
# contents of bmi.R
bmi <- function(weight, height) weight/(height^2)
```

A simple **tinytest** test file could look like this.

```
# contents of test_script.R
data(women)
women$height <- women$height * 2.54/100
women$weight <- women$weight * 0.453592
BMI    <- with(women, bmi(weight,height) )

expect_true( all(BMI >= 10) )
expect_true( all(BMI <= 30) )
```

The first four lines prepare some data, while the last two lines check whether the prepared data meets our expectations. In an interactive session, we can run the test file after loading the `bmi()` function.

```
> source("bmi.R")
> library(tinytest)
> out <- run_test_file('test_script.R')
Running test_script.R.................    2 tests OK
> print(out, passes=TRUE)
----- PASSED      : test_script.R<7--7>
 call| expect_true(all(BMI >= 10))
----- PASSED      : test_script.R<8--8>
 call| expect_true(all(BMI <= 30))
```

In this application, the file runner locally masks the `expect_*()` functions and captures their result through a local side effect. As we are only interested in the test results, the output of all other expressions is discarded.

Compared to the basic version described in the 'build your own `source()`' section, this file runner keeps some extra administration, such as the line numbers of each masked expression. These can be extracted from the output of `parse()`. The package comes with a number of assertions in the form of `expect_*()` functions. It is possible to extend **tinytest** by registering new assertions. These are then automatically masked by the file runner. The only requirement on the new assertions is that they return an object of the same type as the built-in assertions (an object of class '`tinytest`').

Discussion

The techniques demonstrated here have two major advantages. First, it allows for a clean and side-effect free separation between the primary and secondary data flows. As a result, the secondary data flow is composed with the primary data flow. In other words: a user that wants to add a secondary data flow to an existing script does not have to edit any existing code. Instead, it is only necessary to add a bit of code to specify and initialize the secondary stream, which is a big advantage for maintainability. Second, the current mechanisms avoid the use of condition signals. This also leads to code that is easier to understand and navigate because all code associated with the secondary flow can be limited to the scope of a single function (here: either a file runner or a pipe operator). Since the secondary data flow is not treated as an unusual condition (exception), the exception signaling channel is free for transmitting truly unusual conditions such as errors and warnings.

There are also some limitations inherent to these techniques. Although the code for the secondary data flow is easy to compose with code for the primary data flow, it is not as easy to compose different secondary data flows. For example, one can use only one file runner to run an R script and only a single pipe operator to combine two expressions.

A second limitation is that this approach does not recurse into the primary expressions. For example, the expression counters we developed only count user-defined expressions. They can not count expressions that are called by functions called by the user. This means that something like a code coverage tool such as **covr** is out of scope.

A third and related limitation is that the resolution of expressions may be too low for certain applications. For example in R, 'if' is an expression (it returns a value when evaluated) rather than a statement (like for). This means that parse() interprets a block such as

```
if ( x > 0 ){
 x <- 10
 y <- 2*x
}
```

as a single expression. If higher resolution is needed, this requires explicit manipulation of the user code.

Finally, the local masking mechanism excludes the use of the namespace resolution operator. For example, in **lumberjack**, it is not possible to use lumberjack::start_log() since, in that case, the user-facing function from the package is executed and not the masked function with the desired local side-effect.

Conclusion

In this paper we demonstrated a set of techniques that allow one to add a secondary data flow to an existing user-defined R script. The core idea is that we manipulate way expressions are combined before they are executed. In practice, we use R's parse() and eval() to add secondary data stream to user code, or build a special 'pipe' operator. Local masking and local side effects allow a user to control the secondary data flow without global side-effects. The result is a clean separation of concerns between the primary and secondary data flow, that does not rely on condition handling, is void of global side-effects, and that is implemented in pure R.

Mark P.J. van der Loo
Statistics Netherlands
PO-BOX 24500, 2490HA Den Haag
The Netherlands
https://orcid.org/0000-0002-9807-4686
https://www.markvanderloo.eu
m.vanderloo@cbs.nl

Bibliography

S. M. Bache and H. Wickham. *magrittr: A Forward-Pipe Operator for R*, 2014. URL https://CRAN.R-project.org/package=magrittr. R package version 1.5. [p45]

M. Burger, K. Juenemann, and T. Koenig. *RUnit: R Unit Test Framework*, 2018. URL https://CRAN.R-project.org/package=RUnit. R package version 0.4.32. [p43]

W. Chang. *R6: Encapsulated Classes with Reference Semantics*, 2020. URL https://CRAN.R-project.org/package=R6. R package version 2.5.0. [p49]

G. Daróczi. *logger: A Lightweight, Modern and Flexible Logging Utility*, 2021. URL https://CRAN.R-project.org/package=logger. R package version 0.2.0. [p43]

M. Frasca. *logging: R Logging Package*, 2019. URL https://CRAN.R-project.org/package=logging. R package version 0.10-108. [p43]

B. Gaslam. *unitizer: Interactive R Unit Tests*, 2021. URL https://CRAN.R-project.org/package=unitizer. R package version 1.4.14. [p43]

J. Hester. *covr: Test Coverage for Packages*, 2020a. URL https://CRAN.R-project.org/package=covr. R package version 3.5.1. [p43]

J. Hester. *bench: High Precision Timing of R Expressions*, 2020b. URL https://CRAN.R-project.org/package=bench. R package version 1.1.1. [p43]

O. Mersmann. *microbenchmark: Accurate Timing Functions*, 2019. URL https://CRAN.R-project.org/package=microbenchmark. R package version 1.4-7. [p43]

B. Milewski. *Category Theory for Programmers*. Blurb, Incorporated, 2018. ISBN 9780464825081. See also the online lectures: https://youtu.be/I8LbkfSSR58. [p44]

J. Mount and N. Zumel. Dot-Pipe: an S3 Extensible Pipe for R. *The R Journal*, 10(2):309–316, 2018. doi: 10.32614/RJ-2018-042. URL https://doi.org/10.32614/RJ-2018-042. [p46]

B. L. Y. Rowe. *futile.logger: A Logging Utility for R*, 2016. URL https://CRAN.R-project.org/package=futile.logger. R package version 1.4.3. [p43]

M. van der Loo. *tinytest: Lightweight but Feature Complete Unit Testing Framework*, 2020. URL https://github.com/markvanderloo/tinytest. R package version 1.2.4. [p43, 44, 50]

M. P. J. van der Loo. Monitoring Data in R with the lumberjack Package. *Journal of Statistical Software*, 98:1–13, 2021. [p44, 49]

H. Wickham. testthat: Get started with testing. *The R Journal*, 3:5–10, 2011. URL https://journal.r-project.org/archive/2011-1/RJournal_2011-1_Wickham.pdf. [p43]

Y. Xie. *testit: A Simple Package for Testing R Packages*, 2021. URL https://CRAN.R-project.org/package=testit. R package version 0.13. [p43]

JMcmprsk: An R Package for Joint Modelling of Longitudinal and Survival Data with Competing Risks

by Hong Wang, Ning Li, Shanpeng Li, and Gang Li

Abstract In this paper, we describe an R package named **JMcmprsk**, for joint modelling of longitudinal and survival data with competing risks. The package in its current version implements two joint models of longitudinal and survival data proposed to handle competing risks survival data together with continuous and ordinal longitudinal outcomes respectively (Elashoff et al., 2008; Li et al., 2010). The corresponding R implementations are further illustrated with real examples. The package also provides simulation functions to simulate datasets for joint modelling with continuous or ordinal outcomes under the competing risks scenario, which provide useful tools to validate and evaluate new joint modelling methods.

Introduction

Joint modeling of longitudinal and survival data has drawn a lot of attention over the past two decades. Much of the research has been focused on data with a single event time and a single type of failure, usually under the assumption of independent censoring of event times (Tsiatis and Davidian, 2004). However, in some situations interest lies with competing risks data, where there is more than one possible cause of an event or where the censoring is informative (Williamson et al., 2008). Typically, a standard linear mixed model or its extensions are used for the longitudinal submodel. Cause-specific hazards model with either unspecified or spline baseline hazards are studied for the competing risk submodels. Various types of random effects are assumed to account for the association between these submodels.

Despite various theoretical and methodological developments (Hickey et al., 2018b; Papageorgiou et al., 2019), there are still limited software packages to deal with specific problems in the analysis of follow-up data in clinical studies. To our knowledge, currently, there are three related CRAN R packages, namely **JM** (Rizopoulos, 2012), **joineR** (Williamson et al., 2008), and **lcmm** (Proust-Lima et al., 2017), which support the modeling of longitudinal and survival data with competing risks.

The **JM** package provides support for competing risks via the "CompRisk" option in the `jointModel()` function. In **JM**, a linear mixed-effects submodel is modeled for the longitudinal part and a relative risk submodel is assumed for each competing event. In the current version (1.4-8), only the piecewise proportional hazards model, where the log baseline hazard is approximated using B-splines, is supported for the survival component. The **joineR** package fits the joint model (Williamson et al., 2008) for joint models of longitudinal data and competing risks using the `joint()` function. In their model, the time-to-event data is modeled using a cause-specific Cox proportional hazards regression model with time-varying covariates. The longitudinal outcome is modeled using a linear mixed effects model. The association is captured by a zero-mean shared latent Gaussian process. Parameters in the model are estimated using an Expectation Maximization(EM) algorithm. The **lcmm** package implements the support for competing risks joint modeling in the `Jointlcmm()` function. Radically different from the above two R packages, the **lcmm** package uses a less well-known framework called the joint latent class model (Proust-Lima et al., 2014), which assumes that dependency between the longitudinal markers and the survival risk can be captured by a latent class structure entirely. However, the **lcmm** package is mainly designed for prediction purpose and may not be suited to evaluate specific assumptions regarding the characteristics of the marker trajectory that are the most influential on the event risk (Proust-Lima et al., 2014).

In all these packages, a time-independent shared random effects vector is usually assumed in modeling the longitudinal and survival data. However, they are not capable of fitting more flexible models with separate random effects in these submodels (Elashoff et al., 2008; Li et al., 2010). In many biomedical applications, sometimes, it is necessary to have a model which takes into account longitudinal ordinal outcomes for the longitudinal part. Yet, due to the complex nature of joint modeling, most of the available software does not support longitudinal ordinal variables (Armero et al., 2016; Ferrer, 2017). We thus decided to fill this gap and implemented a joint model which supports ordinal disease markers based on our previous work (Li et al., 2010).

Both **JM** and **joineR** packages depend heavily on the R **nlme** and **survival** packages. In **JM**, the linear mixed-effects submodel and the survival submodel are first fitted using `lme()` and `coxph()` R function in these packages before a joint modeling process. In **joineR**, `lme()` and `coxph()` functions

are applied to obtain initial values for parameters in the joint model, which are further estimated by an EM algorithm. The major advantage of using available packages such as **survival** and **nlme** lies that joint modeling R packages can be built quickly with adequate efficiency as most of these base R packages have been optimized for speed. However, if required functionality is not available in these packages, as is the case of Elashoff et al. (2008) and Li et al. (2010), implementing new joint modeling methods is a non-trivial task.

Compared with **JM** and **joineR** packages, the **JMcmprsk** package introduced here can be regarded as a "stand-alone" R package, which does not required initial estimates for the linear mixed effects model or survival submodel to compute parameters of the joint model in question. In particular, the **JMcmprsk** package is built within the **Rcpp** (Eddelbuettel et al., 2011) and **GSL**(The GNU Scientific Library)(Galassi et al., 2002) framework, which make R functions have access to a wide range of fast numerical routines such as Monte Carlo integration, numerical integration and differentiation.

Joint Models with Competing Risks

A joint model for competing risk data consists of two linked components: the longitudinal submodel, which takes care of repeatedly measured information and the survival submodel, which deals with multiple failure times. The combination of different longitudinal and survival components leads to a variety of joint models (Hickey et al., 2018a).

In the current version of **JMcmprsk**, we have implemented two joint models for competing risk data, namely joint modeling with continuous longitudinal outcomes (Elashoff et al., 2008), and joint modeling with ordinal longitudinal outcomes (Li et al., 2010). Both models have adopted a cause-specific Cox submodel with a frailty term for multiple survival endpoints. The difference between these two models lies in the longitudinal part. The former model applies a linear mixed submodel for the continuous longitudinal outcome, while the latter model includes a partial proportional odds submodel for the ordinal longitudinal outcome.

Different from previous approaches (Rizopoulos, 2012; Williamson et al., 2008), we assume a flexible separate random effects structure for the longitudinal submodel and the survival submodel. Furthermore, the association between both submodels is modeled by the assumption that the random effects in two submodels jointly have a multivariate normal distribution.

Model 1: Joint modeling with continuous longitudinal outcomes

Let $Y_i(t)$ be the longitudinal outcome measured at time t for subject i, $i = 1, 2, \cdots, n$ and n is the total number of subjects in study. Let $C_i = (T_i, D_i)$ denote the competing risks data on subject i, where T_i is the failure time or censoring time, and D_i takes value in $\{0, 1, \cdots, g\}$, with $D_i = 0$ indicating a censored event and $D_i = k$ showing that subject i fails from the kth type of failure, where $k = 1, \cdots, g$.

The joint model is specified in terms of the following two linked submodels:

$$\begin{aligned} Y_i(t) &= X_i^{(1)}(t)^\top \beta + \tilde{X}_i^{(1)}(t)^\top b_i + \epsilon_i(t), \\ \lambda_k(t) &= \lambda_{0k}(t) \exp(X_i^{(2)}(t)^\top \gamma_k + \nu_k u_i), \ \text{ for } \ k = 1, \cdots, g, \end{aligned}$$

where $X_i^{(1)}(t)$, $X_i^{(2)}(t)$ denote the covariates for the fixed-effects β and γ_k, $\tilde{X}_i^{(1)}(t)$ denotes the covariates for the random-effects b_i and $\epsilon_i(t) \sim N(0, \sigma^2)$ for all $t \geq 0$. The parameter ν_1 is set to 1 to ensure identifiability. We assume that b_i is independent of $\epsilon_i(t)$ and that $\epsilon_i(t_1)$ is independent of $\epsilon_i(t_2)$ for any $t_1 \neq t_2$. We further assume the random effects b_i and u_i jointly have a multivariate normal distribution, denoted by $\theta_i \sim N(0, \Sigma)$, where $\Sigma = (\Sigma_b, \Sigma_{bu}^\top; \Sigma_{bu}, \sigma_u)$.

Denote Ψ as the unknown parameters from the joint models. We propose to obtain the maximum likelihood estimate of Ψ through an EM algorithm. The complete data likelihood is

$$\begin{aligned} & L(\Psi; Y, C, \theta) \\ & \propto \Pi_{i=1}^n \Big[\Pi_{j=1}^{n_i} \frac{1}{\sqrt{2\pi\sigma^2}} \exp(-\frac{1}{2\sigma^2}(Y_{ij} - X_i^{(1)}(t_{ij})^\top \beta - \tilde{X}_i^{(1)}(t_{ij})^\top b_i)^2) \Big] \\ & \times \Pi_{k=1}^g \lambda_k(T_i)^{I(D_i=k)} \exp\Big\{ - \int_0^{T_i} \sum_{k=1}^g \lambda_k(t) dt \Big\} \\ & \times \frac{1}{\sqrt{(2\pi)^d |\Sigma|}} \exp(-\frac{1}{2} \theta_i^\top \Sigma^{-1} \theta_i). \end{aligned}$$

In the E-step, we need to calculate the expected value of all the functions of θ. Since the integral over the random effects does not have a closed-form solution, an iterative numerical method has to be employed.

In **JMcmprsk**, the integral over time is approximated using a Gauss-Kronrod quadrature and the computation of the integral over the individual random effects is achieved using a Gauss-Hermite quadrature. The quadrature approximates the integral using a weighted sum of function values at specified points within the domain of integration; the Gaussian quadrature is based on the use of polynomial functions. A standard option here is the Gaussian quadratic rules. In the M-step, Ψ is updated by maximizing the functions obtained from the E-step.

Model 2: Joint modeling with ordinal longitudinal outcomes

Let Y_{ij} denote the jth response measured on subject i, where $i = 1, \cdots, n$, $j = 1, \cdots, n_i$, and Y_{ij} takes values in $\{1, \cdots, K\}$. The competing risks failure times on subject i is (T_i, D_i), and the notations have the same meaning as in Model 1.

We propose the following partial proportional odds model for Y_{ij}

$$P(Y_{ij} \leq k | X_{ij}, \tilde{X}_{ij}, W_{ij}, b_i) = \frac{1}{1 + \exp(-\theta_k - X_{ij}\beta - \tilde{X}_{ij}\alpha_k - W_{ij}^\top b_i)},$$

where $k = 1, \cdots, K-1$, X_{ij} and $\tilde{X}_{ij}$ are $p \times 1$ and $s \times 1$ vectors of covariates for the fixed-effect β and α_k, with $\alpha_1 = 0$, and $\tilde{X}_{ij}$ is a subset of X_{ij} for which the proportional odds assumption may not be satisfied. The $q \times 1$ vector b_i represents random effects of the associated covariates W_{ij}.

The distribution of the competing risks failure times (T_i, D_i) are assumed to take the form of the following cause-specific hazards frailty model:

$$\lambda_d(t|Z_i(t), u_i) \quad = \quad \lambda_{0d}(t) \exp(Z_i(t)^\top \gamma_d + \nu_d u_i), \text{ for } d = 1, \cdots, g,$$

where the $l \times 1$ vector γ_d and ν_d are the cause-specific coefficients for the covariates $Z_i(t)$ and the random effects u_i, respectively.

The parameter ν_1 is set to 1 to ensure identifiability. We assume the random effects b_i and u_i jointly have a multivariate normal distribution, denoted by $a_i \sim N(0, \Sigma)$.

Denote Ψ as the unknown parameters from the joint models. We propose to obtain the maximum likelihood estimate of Ψ through an EM algorithm. The complete data likelihood is

$$\begin{aligned}
&L(\Psi; Y, C, a) \\
&\propto \Pi_{i=1}^{n} \left[\Pi_{j=1}^{n_i} \Pi_{k=1}^{K} \{\pi_{ij}(k) - \pi_{ij}(k-1)\}^{I(Y_{ij}=k)} \right] \\
&\times \Pi_{d=1}^{g} \lambda_d(T_i)^{I(D_i=d)} \exp\left\{ - \int_0^{T_i} \sum_{k=1}^{d} \lambda_d(t) dt \right\} \\
&\times \frac{1}{\sqrt{(2\pi)^{q+1}|\Sigma|}} \exp(-\frac{1}{2} a_i^\top \Sigma^{-1} a_i).
\end{aligned}$$

where $\pi_{ij}(k)$ stands for the probability that $Y_{ij} \leq k$ given the covariates and the random effects. The implementation of EM algorithm in this case is similar to the procedure of Model 1.

Package structure and functionality

The R package **JMcmprsk** implements the above two joint models on the basis of R package **Rcpp** (Eddelbuettel et al., 2011) and **GSL** library(Galassi et al., 2002) and is hosted at CRAN. After setting the GSL environment by following the instructions in the INSTALL file from the package, we can issue the following command in the R console to install the package:

```
> install.packages("JMcmprsk")
```

There are two major functions included in the **JMcmprsk** package: the function that fits continuous outcomes `jmc()` and the function that fits ordinal outcomes `jmo()`.

jmc() function

As an illustrative example of `jmc()`, we consider Scleroderma Lung Study (Tashkin et al., 2006), a double-blinded, randomized clinical trial to evaluate the effectiveness of oral cyclophosphamide (CYC) versus placebo in the treatment of lung disease due to scleroderma. This study consists of 158 patients and the primary outcome is forced vital capacity (FVC, as % predicted) determined at 3-month intervals from the baseline. The event of interest is the time-to-treatment failure or death. We consider two covariates, baseline %FVC (FVC_0) and baseline lung fibrosis (FIB_0) and two risks, informative and noninformative. The model setups are as follows:

$$\begin{aligned} \%FVC_{ij} &= \beta_0 + \beta_1 t_{ij} + \beta_2 FVC_{0i} + \beta_3 FIB_{0i} + \beta_4 CYC_i \\ &\quad + \beta_5 FVC_{0i} \times CYC_i + \beta_6 FIB_{0i} \times CYC_i + \beta_7 t_{ij} \times CYC_i + b_i t_{ij} + \epsilon, \end{aligned}$$

and the cause-specific hazards frailty models are

$$\begin{aligned} \lambda_1(t) &= \lambda_{01}(t) \exp(\gamma_{11} FVC_{0i} + \gamma_{12} FIB_{0i} + \gamma_{13} CYC_i + \gamma_{14} FVC_{0i} \times CYC_i + \gamma_{15} FIB_{0i} \times CYC_i + u_i) \\ \lambda_2(t) &= \lambda_{02}(t) \exp(\gamma_{21} FVC_{0i} + \gamma_{22} FIB_{0i} + \gamma_{23} CYC_i + \gamma_{24} FVC_{0i} \times CYC_i + \gamma_{25} FIB_{0i} \times CYC_i + \nu_2 u_i), \end{aligned}$$

We first load the package and the data.

```
library(JMcmprsk)
set.seed(123)
data(lung)
yread <- lung[, c(1,2:11)]
cread <- unique(lung[, c(1, 12, 13, 6:10)])
```

The number of rows in "yread" is the total number of measurements for all subjects in the study. For "cread", the survival/censoring time is included in the first column, and the failure type coded as 0 (censored events), 1 (risk 1), or 2 (risk 2) is given in the second column. Two competing risks are assumed.

Then, "yread" and "cread" are used as the longitudinal and survival input data for the model specified by the function `jmc()` as shown below:

```
jmcfit <- jmc(long_data = yread, surv_data = cread, out = "FVC",
           FE = c("time", "FVC0", "FIB0", "CYC", "FVC0.CYC",
                  "FIB0.CYC", "time.CYC"),
           RE = "linear", ID = "ID",cate = NULL, intcpt = 0,
           quad.points = 20, quiet = TRUE, do.trace = FALSE)
```

where `out` is the name of the outcome variable in the longitudinal sub-model, `FE` the list of covariates for the fixed effects in the longitudinal sub-model, `RE` the types/vector of random effects in the longitudinal sub-model, `ID` the column name of subject id, `cate` the list of categorical variables for the fixed effects in the longitudinal sub-model, `intcpt` the indicator of random intercept coded as 1 (yes, default) or 0(no). The option `quiet` is used to print the progress of function, the default is TRUE (no printing).

A concise summary of the results can be obtained using `jmcfit` as shown below:

```
>jmcfit
Call:
 jmc(long_data = yread, surv_data = cread, out = "FVC",
FE = c("time", "FVC0", "FIB0", "CYC", "FVC0.CYC", "FIB0.CYC", "time.CYC"),
RE = "linear", ID = "ID", cate = NULL, intcpt = 0, quad.points = 20, quiet = FALSE)

Data Summary:
Number of observations: 715
Number of groups: 140

Proportion of competing risks:
Risk 1 : 10 %
Risk 2 : 22.86 %

Numerical intergration:
Method: standard Guass-Hermite quadrature
Number of quadrature points:  20

Model Type: joint modeling of longitudinal continuous and competing risks data
```

```
Model summary:
Longitudinal process: linear mixed effects model
Event process: cause-specific Cox proportional hazard model with unspecified baseline hazard

Loglikelihood:  -3799.044

Longitudinal sub-model fixed effects:  FVC ~ time + FVC0 + FIB0 + CYC + FVC0.CYC + FIB0.CYC + time.CYC

                 Estimate   Std. Error       95% CI            Pr(>|Z|)
Longitudinal:
 Fixed effects:
  intercept      66.0415      0.7541      ( 64.5634, 67.5196)      0.0000
  time           -0.0616      0.0790      (-0.2165, 0.0932)      0.4353
  FVC0           0.9017       0.0365      ( 0.8302, 0.9732)      0.0000
  FIB0           -1.7780      0.5605      (-2.8767,-0.6793)      0.0015
  CYC            0.0150       0.9678      (-1.8819, 1.9119)      0.9876
  FVC0.CYC       0.1380       0.0650      ( 0.0106, 0.2654)      0.0338
  FIB0.CYC       1.7088       0.7643      ( 0.2109, 3.2068)      0.0254
  time.CYC       0.1278       0.1102      (-0.0883, 0.3438)      0.2464
Random effects:
  sigma^2         22.7366       0.6575  ( 21.4478, 24.0253)      0.0000

Survival sub-model fixed effects:  Surv(surv, failure_type) ~ FVC0 + FIB0 + CYC + FVC0.CYC + FIB0.CYC

                 Estimate   Std. Error       95% CI           Pr(>|Z|)
Survival:
 Fixed effects:
  FVC0_1         0.0187       0.0326      (-0.0452, 0.0826)      0.5660
  FIB0_1         0.1803       0.3521      (-0.5098, 0.8705)      0.6086
  CYC_1          -0.6872      0.7653      (-2.1872, 0.8128)      0.3692
  FVC0.CYC_1     -0.0517      0.0746      (-0.1979, 0.0945)      0.4880
  FIB0.CYC_1     -0.4665      1.1099      (-2.6419, 1.7089)      0.6743
  FVC0_2         -0.0677      0.0271      (-0.1208,-0.0147)      0.0123
  FIB0_2         0.1965       0.3290      (-0.4484, 0.8414)      0.5503
  CYC_2          0.3137       0.4665      (-0.6007, 1.2280)      0.5013
  FVC0.CYC_2     0.1051       0.0410      ( 0.0248, 0.1854)      0.0103
  FIB0.CYC_2     0.1239       0.4120      (-0.6836, 0.9314)      0.7636

Association parameter:
  v2             1.9949       2.3093      (-2.5314, 6.5212)      0.3877

Random effects:
  sigma_b11      0.2215       0.0294      ( 0.1638, 0.2792)      0.0000
  sigma_u        0.0501       0.0898      (-0.1259, 0.2260)      0.5772
 Covariance:
  sigma_b1u      -0.0997      0.0797      (-0.2560, 0.0565)      0.2109
```

The resulting table contains three parts, the fixed effects in longitudinal model, survival model and random effects. It gives the estimated parameters in the first column, the standard error in the second column, and 95% confidence interval and p-value for these parameters in the third and fourth columns. In our example, there is only one random effect. If there is more than one random effect, the output will include $sigma_b11, sigma_b12, sigma_b22, sigma_b1u, sigma_b2u$, and so on.

The supporting function `coef()` can be used to extract the coefficients of the longitudinal/survival process by specifying the argument `coeff`, where"beta" and "gamma" denotes the longitudinal and survival submodel fixed effects, respectively.

```
beta <- coef(jmcfit, coeff = "beta")
>beta
  intercept      time.1        FVC0        FIB0         CYC    FVC0.CYC    FIB0.CYC
66.04146267 -0.06164756  0.90166283 -1.77799172  0.01503104  0.13798885  1.70883750
   time.CYC
 0.12776670

gamma <- coef(jmcfit, coeff = "gamma")
```

```
>gamma
            FVC0      FIB0        CYC    FVC0.CYC   FIB0.CYC
[1,]  0.01871359 0.1803249 -0.6872099 -0.05172157 -0.4664724
[2,] -0.06772664 0.1965190  0.3136709  0.10509986  0.1239203
```

The supporting function `summary()` can be used to extract the point estimate, the standard error, 95%CI, and p-values of the coefficients of both sub-models with the option `coeff` to specify which submodel fixed effects one would like to extract, and `digits`, the number of digits to be printed out. We proceed below to extract the fixed effects for both submodels:

```
>summary(jmcfit, coeff = "longitudinal", digits = 4)
  Longitudinal    coef     SE 95%Lower 95%Upper p-values
1    intercept  66.0415 0.7541  64.5634  67.5196   0.0000
2         time  -0.0616 0.0790  -0.2165   0.0932   0.4353
3         FVC0   0.9017 0.0365   0.8302   0.9732   0.0000
4         FIB0  -1.7780 0.5605  -2.8767  -0.6793   0.0015
5          CYC   0.0150 0.9678  -1.8819   1.9119   0.9876
6     FVC0.CYC   0.1380 0.0650   0.0106   0.2654   0.0338
7     FIB0.CYC   1.7088 0.7643   0.2109   3.2068   0.0254
8     time.CYC   0.1278 0.1102  -0.0883   0.3438   0.2464

>summary(jmcfit, coeff = "survival", digits = 4)
     Survival    coef exp(coef) SE(coef) 95%Lower 95%Upper p-values
1      FVC0_1  0.0187    1.0189   0.0326  -0.0452   0.0826   0.5660
2      FIB0_1  0.1803    1.1976   0.3521  -0.5098   0.8705   0.6086
3       CYC_1 -0.6872    0.5030   0.7653  -2.1872   0.8128   0.3692
4  FVC0.CYC_1 -0.0517    0.9496   0.0746  -0.1979   0.0945   0.4880
5  FIB0.CYC_1 -0.4665    0.6272   1.1099  -2.6419   1.7089   0.6743
6      FVC0_2 -0.0677    0.9345   0.0271  -0.1208  -0.0147   0.0123
7      FIB0_2  0.1965    1.2172   0.3290  -0.4484   0.8414   0.5503
8       CYC_2  0.3137    1.3684   0.4665  -0.6007   1.2280   0.5013
9  FVC0.CYC_2  0.1051    1.1108   0.0410   0.0248   0.1854   0.0103
10 FIB0.CYC_2  0.1239    1.1319   0.4120  -0.6836   0.9314   0.7636
```

We proceed to test the global hypothesis for the longitudinal and the survival submodels using `linearTest()`.

```
>linearTest(jmcfit, coeff="beta")
             Chisq df Pr(>|Chi|)
L*beta=Cb 1072.307  7 0.0000
>linearTest(jmcfit, coeff="gamma")
              Chisq df Pr(>|Chi|)
L*gamma=Cg 11.06558 10 0.3524
```

The results suggest that the hypothesis $\beta_1 = \beta_2 = \cdots = \beta_7 = 0$ is rejected, and the hypothesis $\gamma_{11} = \gamma_{12} = \cdots = \gamma_{15} = \gamma_{21} = \gamma_{22} = \cdots = \gamma_{25} = 0$ is not rejected at the significance level of 0.05.

`linearTest()` can also be used to test any linear hypothesis about the coefficients for each submodel. For example, if one wants to test $H_0 : \beta_1 = \beta_2$ in the longitudinal submodel, then we start with a linear contrast `Lb` and pass it to `linearTest()`.

```
Lb <- matrix(c(1, -1, 0, 0, 0, 0, 0), ncol = length(beta)-1, nrow = 1)
>linearTest(jmcfit, coeff="beta", Lb = Lb)
             Chisq df Pr(>|Chi|)
L*beta=Cb 124.8179  1 0.0000
```

Note that we do not include intercept for linear hypotheses testing. It is seen that the hypothesis $\beta_1 = \beta_2$ is rejected at level 0.05 in the above example.

Similarly, a linear hypotheses testing can also be done in the survival submodel using `linearTest()`. For example, if we want to test $H_0 : \gamma_{11} = \gamma_{21}$, then we start with another linear contrast `Lg` and pass it to `linearTest()`.

```
Lg <- matrix(c(1, 0, 0, 0, 0, -1, 0, 0, 0, 0), ncol = length(gamma), nrow = 1)
>linearTest(jmcfit, coeff="gamma", Lg = Lg)
              Chisq df Pr(>|Chi|)
L*gamma=Cg 4.301511  1 0.0381
```

It is seen that the hypothesis $\gamma_{11} = \gamma_{21}$ is rejected at level 0.05.

For categorical variables, `jmc()` function will create the appropriate dummy variables automatically as needed within the function. The reference group in a categorical variable is specified as the one that comes first alphabetically. Below is another example:

First, we add two categorical variables "sex" and "race" to the longitudinal data set "yread", in which "sex" is coded as "Female" or "Male", and race is coded as "Asian", "White", "Black", or "Hispanic".

```
#make up two categorical variables and add them into yread
set.seed(123)
sex <- sample(c("Female", "Male"), nrow(cread), replace = TRUE)
race <- sample(c("White", "Black", "Asian", "Hispanic"),
               nrow(cread), replace = TRUE)
ID <- cread$ID
cate_var <- data.frame(ID, sex, race)
if (require(dplyr)) {
  yread <- dplyr::left_join(yread, cate_var, by = "ID")
}
```

Second, we rerun the model with the two added categorical variables.

```
# run jmc function again for yread file with two added categorical variables
res2 <- jmc(long_data = yread, surv_data = cread,
            out = "FVC", cate = c("sex", "race"),
            FE = c("time", "FVC0", "FIB0", "CYC", "FVC0.CYC",
                   "FIB0.CYC", "time.CYC"),
            RE = "time", ID = "ID", intcpt = 0,
            quad.points = 20, quiet = FALSE)
res2
```

We can obtain the estimated coefficients of the longitudinal process using `coef()`.

```
> coef(res2, coeff = "beta")
  intercept        time        FVC0        FIB0         CYC    FVC0.CYC    FIB0.CYC    time.CYC
67.05760799 -0.07340060  0.91105151 -1.75007966  0.02269507  0.13045588  1.58807248  0.15876200
       Male       Black    Hispanic       White
-0.77110697 -0.94635182 -0.45873814 -1.19910638
```

jmo() function

The implementation of `jmo()` is very similar to that of `jmc()`. As an illustrative example, we use the data from (rt PA Stroke Study, 1995). In this study, 624 patients are included, and the patients treated with rt-PA were compared with those given placebo to look for an improvement from baseline in the score on the modified Rankin scale, an ordinal measure of the degree of disability with categories ranging from no symptoms, no significant disability to severe disability or death, which means in this example, Y_{ij} takes $K = 4$ ordinal values. The following covariates are considered: treatment group (rt-PA or placebo), modified Rankin scale prior stroke onset, time since randomization (dummy variables for 3, 6 and 12 months), and the three subtypes of acute stroke (small vessel occlusive disease, large vessel atherosclerosis or cardioembolic stroke, and unknown reasons). Similarly, we also consider the informative and noninformative risks. The model setups are as follows:

$$\begin{aligned} P(Y_{ij} \le k) &= [1 + \exp(-\theta_k - (\beta_1 Group + \beta_2 \text{Modified Rankin scale prior onset} + \beta_3 time3 \\ &\quad + \beta_4 time6 + \beta_5 time12 + \beta_6 \text{Small vessel} + \beta_7 \text{Large vessel or cardioembolic stroke} \\ &\quad + \beta_8 \text{Small vessel*group} + \beta_9 \text{Large vessel or cardioembolic stroke*group}) \\ &\quad - (\alpha_{k1} \text{Small vessel} + \alpha_{k2} \text{Large vessel or cardioembolic stroke}) - b_i)]^{-1}, \end{aligned}$$

where $k = 1, \cdots, K-1$.

$$\begin{aligned} \lambda_1(t) &= \lambda_{01}(t) \exp(\gamma_{11} Group + \gamma_{12} \text{Modified Rankin scale prior onset} \\ &\quad + \gamma_{13} \text{Small vessel} + \gamma_{14} \text{Large vessel or cardioembolic stroke} \\ &\quad + \gamma_{15} \text{Small vessel*group} + \gamma_{16} \text{Large vessel or cardioembolic stroke*group} + u_i) \\ \lambda_2(t) &= \lambda_{02}(t) \exp(\gamma_{21} Group + \gamma_{22} \text{Modified Rankin scale prior onset} \\ &\quad + \gamma_{23} \text{Small vessel} + \gamma_{24} \text{Large vessel or cardioembolic stroke} \\ &\quad + \gamma_{25} \text{Small vessel*group} + \gamma_{26} \text{Large vessel or cardioembolic stroke*group} + \nu_2 u_i) \end{aligned}$$

We first load the package and the data.

```
library(JMcmprsk)
set.seed(123)
data(ninds)
yread <- ninds[, c(1, 2:14)]
cread <- ninds[, c(1, 15, 16, 6, 10:14)]
cread <- unique(cread)
```

and the other arrangements are the same with those in `jmc()`,

```
jmofit <- jmo(yread, cread, out = "Y",
              FE = c("group", "time3", "time6", "time12", "mrkprior",
                     "smlves", "lvORcs", "smlves.group", "lvORcs.group"),
              cate = NULL,RE = "intercept", NP = c("smlves", "lvORcs"),
              ID = "ID",intcpt = 1, quad.points = 20,
              max.iter = 1000, quiet = FALSE, do.trace = FALSE)
```

where `NP` is the list of non-proportional odds covariates and `FE` the list of proportional odds covariates.

To see a concise summary of the result, we can type:

```
>jmofit
Call:
 jmo(long_data = yread, surv_data = cread, out = "Y",
FE = c("group", "time3", "time6", "time12", "mrkprior", "smlves", "lvORcs", "smlves.group", "lvORcs.group"),
RE = "intercept", NP = c("smlves", "lvORcs"), ID = "ID", cate = NULL, intcpt = 1,
quad.points = 20, max.iter = 1000, quiet = FALSE, do.trace = FALSE)

Data Summary:
Number of observations: 1906
Number of groups: 587

Proportion of competing risks:
Risk 1 : 32.88 %
Risk 2 : 4.26 %

Numerical intergration:
Method: Standard Guass-Hermite quadrature
Number of quadrature points:  20

Model Type: joint modeling of longitudinal ordinal and competing risks data

Model summary:
Longitudinal process: partial proportional odds model
Event process: cause-specific Cox proportional hazard model with unspecified baseline hazard

Loglikelihood:  -2292.271

Longitudinal sub-model proportional odds:  Y ~ group + time3 + time6 + time12 + mrkprior + smlves +
lvORcs + smlves.group + lvORcs.group
Longitudinal sub-model non-proportional odds: smlves_NP + lvORcs_NP

                  Estimate   Std. Error       95% CI            Pr(>|Z|)
Longitudinal:
 Fixed effects:
  proportional odds:
  group          1.6053       0.1905      ( 1.2319, 1.9786)      0.0000
  time3          2.5132       0.1934      ( 2.1341, 2.8923)      0.0000
  time6          2.6980       0.1962      ( 2.3134, 3.0825)      0.0000
  time12         2.9415       0.2004      ( 2.5486, 3.3344)      0.0000
  mrkprior      -2.1815       0.2167      (-2.6063,-1.7567)      0.0000
  smlves         6.4358       0.4228      ( 5.6072, 7.2644)      0.0000
  lvORcs        -1.2907       0.2861      (-1.8515,-0.7300)      0.0000
  smlves.group   0.4903       0.7498      (-0.9793, 1.9598)      0.5132
  lvORcs.group  -3.2277       0.4210      (-4.0528,-2.4026)      0.0000
```

```
  Non-proportional odds:
  smlves_NP_2    0.2725       0.4485      (-0.6066, 1.1515)      0.5435
  lvORcs_NP_2    -0.4528      0.2466      (-0.9362, 0.0305)      0.0663
  smlves_NP_3    1.7844       1.0613      (-0.2958, 3.8645)      0.0927
  lvORcs_NP_3    -0.1364      0.4309      (-0.9809, 0.7081)      0.7516
  Logit-specific intercepts:
  theta1         -6.2336      0.1722      (-6.5712,-5.8960)      0.0000
  theta2         -4.1911      0.1561      (-4.4971,-3.8851)      0.0000
  theta3         3.9806       0.1896      ( 3.6091, 4.3522)      0.0000

Survival sub-model fixed effects:  Surv(surv, comprisk) ~ group + mrkprior + smlves + lvORcs +
smlves.group + lvORcs.group

                 Estimate   Std. Error       95% CI            Pr(>|Z|)
Survival:
 Fixed effects:
  group_1        -0.4630      0.2434      (-0.9400, 0.0140)      0.0571
  mrkprior_1     0.5874       0.1371      ( 0.3187, 0.8560)      0.0000
  smlves_1       -2.5570      0.7223      (-3.9728,-1.1413)      0.0004
  lvORcs_1       0.5992       0.2485      ( 0.1120, 1.0863)      0.0159
  smlves.group_1 -0.4990      1.4257      (-3.2934, 2.2955)      0.7264
  lvORcs.group_1 1.1675       0.4692      ( 0.2479, 2.0871)      0.0128
  group_2        0.2087       0.4834      (-0.7388, 1.1562)      0.6659
  mrkprior_2     0.0616       0.4277      (-0.7766, 0.8998)      0.8854
  smlves_2       0.7758       0.6217      (-0.4428, 1.9943)      0.2121
  lvORcs_2       -0.3256      0.5120      (-1.3291, 0.6778)      0.5247
  smlves.group_2 -0.0437      1.1573      (-2.3120, 2.2245)      0.9699
  lvORcs.group_2 0.0991       1.0718      (-2.0015, 2.1998)      0.9263

Association prameter:
  v2             0.0101       0.1595      (-0.3025, 0.3227)      0.9496

Random effects:
  sigma_b11      55.6404       5.6560      ( 44.5547, 66.7261)      0.0000
  sigma_u        6.6598       1.7196      ( 3.2894, 10.0303)      0.0001
 Covariance:
  sigma_b1u      -19.2452      0.7730      (-20.7602,-17.7302)      0.0000
```

The usage of function `coef()` is similar to those in Model 1. More specifically, `coef()` can extract the coefficients of non-proportional odds fixed effects and logit-specific intercepts. For example,

```
alpha <- coef(jmofit, coeff = "alpha")
>alpha
     smlves_NP  lvORcs_NP
[1,] 0.2724605 -0.4528214
[2,] 1.7843743 -0.1363731

theta <- coef(jmofit, coeff = "theta")
> theta
[1] -6.233618 -4.191114  3.980638
```

The usage of function `summary()` is the same as in Model 1. It extracts the point estimate, standard error, 95%CI, and p-values of the coefficients of both submodels as demonstrated below:

```
> summary(jmofit, coeff = "longitudinal")
   Longitudinal    coef     SE 95%Lower 95%Upper p-values
1         group  1.6053 0.1905   1.2319   1.9786   0.0000
2         time3  2.5132 0.1934   2.1341   2.8923   0.0000
3         time6  2.6980 0.1962   2.3134   3.0825   0.0000
4        time12  2.9415 0.2004   2.5486   3.3344   0.0000
5      mrkprior -2.1815 0.2167  -2.6063  -1.7567   0.0000
6        smlves  6.4358 0.4228   5.6072   7.2644   0.0000
7        lvORcs -1.2907 0.2861  -1.8515  -0.7300   0.0000
8  smlves.group  0.4903 0.7498  -0.9793   1.9598   0.5132
9  lvORcs.group -3.2277 0.4210  -4.0528  -2.4026   0.0000
```

```
10  smlves_NP_2  0.2725 0.4485  -0.6066   1.1515   0.5435
11  lvORcs_NP_2 -0.4528 0.2466  -0.9362   0.0305   0.0663
12  smlves_NP_3  1.7844 1.0613  -0.2958   3.8645   0.0927
13  lvORcs_NP_3 -0.1364 0.4309  -0.9809   0.7081   0.7516
14       theta1 -6.2336 0.1722  -6.5712  -5.8960   0.0000
15       theta2 -4.1911 0.1561  -4.4971  -3.8851   0.0000
16       theta3  3.9806 0.1896   3.6091   4.3522   0.0000

> summary(jmofit, coeff = "survival")
         Survival    coef exp(coef) SE(coef) 95%Lower 95%Upper p-values
1         group_1 -0.4630    0.6294   0.2434  -0.9400   0.0140   0.0571
2      mrkprior_1  0.5874    1.7993   0.1371   0.3187   0.8560   0.0000
3        smlves_1 -2.5570    0.0775   0.7223  -3.9728  -1.1413   0.0004
4        lvORcs_1  0.5992    1.8206   0.2485   0.1120   1.0863   0.0159
5  smlves.group_1 -0.4990    0.6072   1.4257  -3.2934   2.2955   0.7264
6  lvORcs.group_1  1.1675    3.2140   0.4692   0.2479   2.0871   0.0128
7         group_2  0.2087    1.2321   0.4834  -0.7388   1.1562   0.6659
8      mrkprior_2  0.0616    1.0636   0.4277  -0.7766   0.8998   0.8854
9        smlves_2  0.7758    2.1722   0.6217  -0.4428   1.9943   0.2121
10       lvORcs_2 -0.3256    0.7221   0.5120  -1.3291   0.6778   0.5247
11 smlves.group_2 -0.0437    0.9572   1.1573  -2.3120   2.2245   0.9699
12 lvORcs.group_2  0.0991    1.1042   1.0718  -2.0015   2.1998   0.9263
```

Analogous to `jmcfit`, `linearTest()` can be used to the global hypothesis for the longitudinal and the survival submodels.

```
> linearTest(jmofit,coeff="beta")
             Chisq df Pr(>|Chi|)
L*beta=Cb 1096.991  9 0.0000
> linearTest(jmofit,coeff="gamma")
              Chisq df Pr(>|Chi|)
L*gamma=Cg 47.15038 12 0.0000
> linearTest(jmofit,coeff="alpha")
              Chisq df Pr(>|Chi|)
L*alpha=Ca 8.776262  4 0.0669
```

According to the p-values, the hypothesis $\beta_1 = \beta_2 = \cdots = \beta_9 = 0$ is rejected, $\gamma_{11} = \gamma_{12} = \cdots = \gamma_{16} = \gamma_{21} = \gamma_{22} = \cdots = \gamma_{26} = 0$ is rejected, but $\alpha_{11} = \alpha_{12} = \alpha_{21} = \alpha_{22} = 0$ is not rejected at the significance level of 0.05.

Similarly, `linearTest()` can be used to test a linear hypothesis for non-proportional odds fixed effects in the longitudinal submodel. For example, if we want to test $H_0 : \alpha_{11} = \alpha_{21}$, then we can simply type:

```
La <- matrix(c(1, 0, -1, 0), ncol = length(alpha), nrow = 1)
> linearTest(jmofit, coeff = "alpha", La = La)
              Chisq df Pr(>|Chi|)
L*alpha=Ca 1.929563  1 0.1648
```

It is seen that the hypothesis $\alpha_{11} = \alpha_{21}$ is not rejected at level 0.05.

Likewise, `jmo()` function allows for categorical variables. Moreover, categorical variables are allowed for setting up non-proportional odds covariates. As an illustration, here we consider the "sex" and "race" variables and use them as two of the non-proportional odds covariates. Below is another example:

```
#Create two categorical variables and add them into yread
ID <- cread$ID
set.seed(123)
sex <- sample(c("Female", "Male"), nrow(cread), replace = TRUE)
race <- sample(c("White", "Black", "Asian", "Hispanic"), nrow(cread), replace = TRUE)
cate_var <- data.frame(ID, sex, race)
if (require(dplyr)) {
   yread <- dplyr::left_join(yread, cate_var, by = "ID")
   }

res2 <- jmo(yread, cread, out = "Y",
```

```
            FE = c("group", "time3", "time6", "time12", "mrkprior",
                           "smlves", "lvORcs", "smlves.group", "lvORcs.group"), cate = c("race", "sex"),
            RE = "intercept", NP = c("smlves", "lvORcs", "race", "sex"), ID = "ID",intcpt = 1,
            quad.points = 20, max.iter = 10000, quiet = FALSE, do.trace = FALSE)
res2
Call:
 jmo(long_data = yread, surv_data = cread, out = "Y",
FE = c("group", "time3", "time6", "time12", "mrkprior", "smlves", "lvORcs", "smlves.group", "lvORcs.group"),
RE = "intercept", NP = c("smlves", "lvORcs", "race", "sex"), ID = "ID", cate = c("race", "sex"),
intcpt = 1, quad.points = 20, max.iter = 10000, quiet = FALSE, do.trace = FALSE)

Data Summary:
Number of observations: 1906
Number of groups: 587

Proportion of competing risks:
Risk 1 : 32.88 %
Risk 2 : 4.26 %

Numerical intergration:
Method: Standard Guass-Hermite quadrature
Number of quadrature points:  20

Model Type: joint modeling of longitudinal ordinal and competing risks data

Model summary:
Longitudinal process: partial proportional odds model
Event process: cause-specific Cox proportional hazard model with unspecified baseline hazard

Loglikelihood:  -2271.831

Longitudinal sub-model proportional odds:  Y ~ group + time3 + time6 + time12 + mrkprior + smlves +
lvORcs + smlves.group + lvORcs.group + Black + Hispanic + White + Male
Longitudinal sub-model non-proportional odds: smlves_NP + lvORcs_NP + Black_NP + Hispanic_NP +
White_NP + Male_NP

                  Estimate   Std. Error       95% CI            Pr(>|Z|)
Longitudinal:
 Fixed effects:
  proportional odds:
  group          1.1430       0.1989      ( 0.7532, 1.5328)      0.0000
  time3          2.4607       0.1963      ( 2.0758, 2.8455)      0.0000
  time6          2.6310       0.1986      ( 2.2416, 3.0203)      0.0000
  time12         2.8717       0.2111      ( 2.4579, 3.2854)      0.0000
  mrkprior      -2.3329       0.1855      (-2.6965,-1.9693)      0.0000
  smlves         3.9941       0.4413      ( 3.1292, 4.8589)      0.0000
  lvORcs        -0.9469       0.3219      (-1.5778,-0.3160)      0.0033
  smlves.group  -4.3940       0.7560      (-5.8758,-2.9123)      0.0000
  lvORcs.group  -3.6954       0.4768      (-4.6299,-2.7608)      0.0000
  Black          0.8235       0.3162      ( 0.2038, 1.4433)      0.0092
  Hispanic      -0.0218       0.3289      (-0.6665, 0.6229)      0.9471
  White          0.0523       0.3457      (-0.6253, 0.7299)      0.8797
  Male          -0.3528       0.2323      (-0.8080, 0.1025)      0.1288
  Non-proportional odds:
  smlves_NP_2    0.3314       0.4310      (-0.5133, 1.1761)      0.4419
  lvORcs_NP_2   -0.3148       0.2696      (-0.8432, 0.2136)      0.2429
  Black_NP_2     0.3781       0.2936      (-0.1973, 0.9535)      0.1978
  Hispanic_NP_2 -0.0303       0.3176      (-0.6528, 0.5923)      0.9241
  White_NP_2    -0.3802       0.3034      (-0.9748, 0.2144)      0.2102
  Male_NP_2      0.0531       0.2221      (-0.3822, 0.4884)      0.8110
  smlves_NP_3    2.2743       1.0748      ( 0.1677, 4.3809)      0.0343
  lvORcs_NP_3    0.0033       0.4632      (-0.9045, 0.9111)      0.9943
  Black_NP_3    -0.2274       0.5419      (-1.2896, 0.8349)      0.6748
  Hispanic_NP_3 -0.5070       0.5087      (-1.5040, 0.4901)      0.3190
```

```
  White_NP_3     0.4205       0.5722      (-0.7010, 1.5420)      0.4624
  Male_NP_3      -0.8489      0.3911      (-1.6155,-0.0824)      0.0300
  Logit-specific intercepts:
  theta1         -6.0565      0.2868      (-6.6186,-5.4945)      0.0000
  theta2         -4.0881      0.2379      (-4.5545,-3.6217)      0.0000
  theta3         4.1340       0.3437      ( 3.4602, 4.8077)      0.0000

Survival sub-model fixed effects:  Surv(surv, comprisk) ~ group + mrkprior + smlves + lvORcs +
smlves.group + lvORcs.group

                  Estimate   Std. Error       95% CI            Pr(>|Z|)
Survival:
 Fixed effects:
  group_1        -0.2815      0.2545      (-0.7802, 0.2173)      0.2687
  mrkprior_1     0.6404       0.1549      ( 0.3367, 0.9440)      0.0000
  smlves_1       -1.8107      0.8252      (-3.4280,-0.1934)      0.0282
  lvORcs_1       0.4894       0.2450      ( 0.0092, 0.9696)      0.0458
  smlves.group_1 1.2608       1.6390      (-1.9517, 4.4733)      0.4417
  lvORcs.group_1 1.4503       0.4901      ( 0.4898, 2.4108)      0.0031
  group_2        0.2073       0.4831      (-0.7396, 1.1542)      0.6678
  mrkprior_2     0.0617       0.4343      (-0.7896, 0.9129)      0.8871
  smlves_2       0.7871       0.6026      (-0.3940, 1.9683)      0.1915
  lvORcs_2       -0.3266      0.5085      (-1.3233, 0.6701)      0.5207
  smlves.group_2 -0.0374      1.1600      (-2.3110, 2.2362)      0.9743
  lvORcs.group_2 0.0952       1.0591      (-1.9807, 2.1711)      0.9284

Association prameter:
  v2             0.0036       0.1577      (-0.3056, 0.3128)      0.9818

Random effects:
  sigma_b11      49.0241       5.0606      ( 39.1053, 58.9430)      0.0000
  sigma_u        6.3475       1.5884      ( 3.2343, 9.4607)      0.0001
 Covariance:
  sigma_b1u      -17.6331      0.7415      (-19.0864,-16.1797)      0.0000

coef(res2, coeff = "beta")
       group        time3        time6       time12     mrkprior       smlves       lvORcs
  1.14302264   2.46065107   2.63095850   2.87165209  -2.33288371   3.99407491  -0.94689649
smlves.group lvORcs.group        Black     Hispanic        White         Male
 -4.39403193  -3.69535020   0.82353645  -0.02181286   0.05232005  -0.35276916
```

Older versions of jmc() and jmo()

In the previous versions of **JMcmprsk**, both the previous `jmc()` and `jmo()` functions require the longitudinal input data "yfile" to be in a specific format regarding the order of the outcome variable and the random and fixed effects covariates. It also requires users to create an additional "mfile" for the longitudinal data. At the suggestions of the reviewers, in the most recent version, we focus and develop user-friendly versions of these functions.

However, for both package consistency and user's convenience, we still keep older versions of these functions in the package, and rename these functions to `jmc_0()` and `jmo_0()`, respectively. Supporting functions of `jmo()` and `jmc()`, such as `coef()`, `summary()`, `linearTest()`, also apply to `jmc_0()` and `jmo_0()` functions.

Here, we show the usage of `jmc_0()` with some simulated data and the "lung" data used in presenting `jmc()` functions.

If the data are provided as files, the function `jmc_0()` has the following usage:

```
library(JMcmprsk)
yfile=system.file("extdata", "jmcsimy.txt", package = "JMcmprsk")
cfile=system.file("extdata", "jmcsimc.txt", package = "JMcmprsk")
mfile=system.file("extdata", "jmcsimm.txt", package = "JMcmprsk")
jmc_0fit = jmc_0(p=4, yfile, cfile, mfile, point=20, do.trace = FALSE)
```

with `p` the dimension of fixed effects (including the intercept) in `yfile`, the option `point` is the number of points used to approximate the integral in the E-step, default is 20, and `do.trace` is used to

control whether the program prints the iteration details. Additionally, the option `type_file` controls the type of data inputs.

If data frames or matrices are provided as inputs, we set the above `type_file` option as `type_file = FALSE` in the `jmc_0()` function:

```
library(JMcmprsk)
data(lung)
lungY <- lung[, c(2:11)]
lungC <- unique(lung[, c(1, 12, 13, 6:10)])
lungC <- lungC[, -1]
## return a vector file with the number of repeated measurements as lungM
lungM <- data.frame(table(lung$ID))
lungM <- as.data.frame(lungM[, 2])
jmc_0fit2=jmc_0(p=8, lungY, lungC, lungM, point=20, do.trace = FALSE, type_file = FALSE)
```

Computational Complexity

To understand the computational complexity of both `jmc()` and `jmo()` models, we carried out a variety of simulations with different sample size and different proportions of events. However, there was no clear trend observed between the proportions of events and running times. Hence, only one event distribution with different sample sizes are given here for illustration purpose. According to Figures 1 and 2, we can easily see that the run time grows much faster as sample size increases, which implies that the computational complexity does not follow a linear order. In this case, it will limit joint models to handling large and even moderate sample size data. To make the joint modeling more scalable, it is necessary to carry out a novel algorithm to reduce its computational complexity to a linear order.

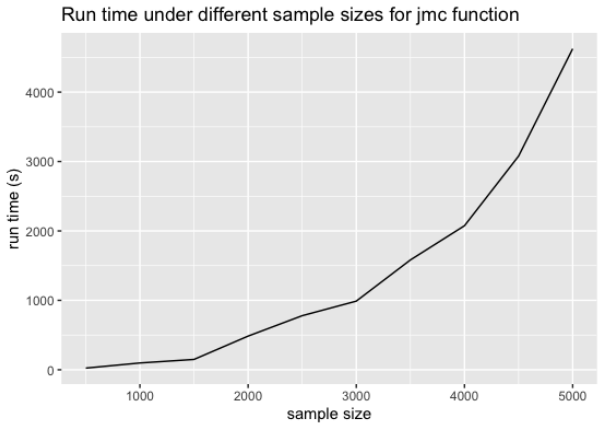


Figure 1: Run time comparison under different sample sizes for `jmc()` function (from 500 to 5000). Data setup: $p = 4$, $n_q = 6$, 10.4% censoring, 51.4% risk 1, and 38.2% risk 2. The run time under each sample size was based on one random sample.

Data Simulation

A simulation can generate datasets with exact ground truth for evaluation. Hence, the simulation of longitudinal and survival data with multiple failures associated with random effects is an important measure to assess the performance of joint modeling approaches dealing with competing risks. In **JMcmprsk**, simulation tools are based on the data models proposed in Elashoff et al. (2008) and Li et al. (2010), which can be used for testing joint models with continuous and ordinal longitudinal outcomes, respectively.

The main function for simulation data continuous longitudinal outcomes and survival data with multiple event outcomes is called `SimDataC()`, which has the following usage:

```
SimDataC(k_val, p1_val, p1a_val, p2_val, g_val, truebeta, truegamma,
  randeffect, yfn, cfn, mfn)
```

We briefly explain some of the important options. `k_val` denotes the number of subjects in study; `p1_val` and `p1a_val` denote the dimension of fixed effects and random effects in longitudinal measurements, respectively; `p2_val` and `g_val` denotes the dimension of fixed effects and number to

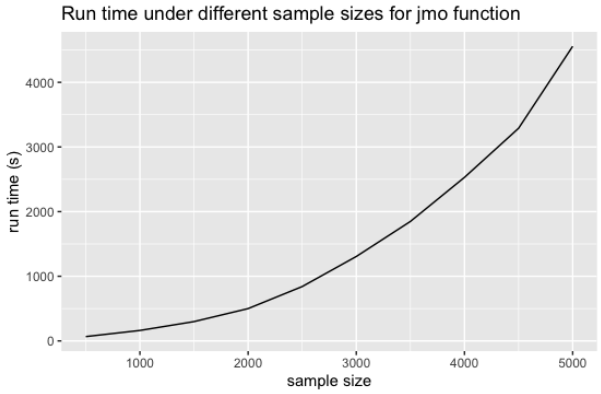


Figure 2: Run time comparison under different sample sizes for `jmo()` function (from 500 to 5000). Data setup: $p = 4$, $n_q = 10$, 22.4% censoring, 57.2% risk 1, and 20.4% risk 2. The run time under each sample size was based on one random sample.

competing risks in survival data; `truebeta` and `truegamma` represent the true values of fixe effects in the longitudinal and the survival submodels, respectively. `randeffect` sets the true values for random effects in longitudinal and competing risks parts, namely in the order of σ,σ_b,ν_2, and σ_u.

The following example generates the datasets used in simulation study in Elashoff et al. (2008):

```
require(JMcmprsk)
set.seed(123)
yfn="jmcsimy1.txt";
cfn="jmcsimc1.txt";
mfn="jmcsimm1.txt";
k_val=200;p1_val=4;p1a_val=1; p2_val=2;g_val=2;
truebeta=c(10,-1,1.5,0.6);truegamma=c(0.8,-1,0.5,-1); randeffect=c(5,0.5,0.5,0.5);
#writing files
SimDataC(k_val, p1_val, p1a_val, p2_val, g_val,truebeta,
         truegamma, randeffect, yfn,  cfn,  mfn)
```

The output of function `SimDataC()` contains additional censoring rate information and newly generated files names for further usage.

```
$`censoring_rate`
[1] 0.21
$rate1
[1] 0.45
$rate2
[1] 0.34
$yfn
[1] "jmcsimy1.txt"
$cfn
[1] "jmcsimc1.txt"
$mfn
[1] "jmcsimm1.txt"
```

The main function for data simulation with ordinal longitudinal outcomes and survival data with multiple event outcomes is called `SimDataO()`, the usage of which is very similar to `SimDataC()`:

```
SimDataO(k_val, p1_val, p1a_val, p2_val, g_val, truebeta, truetheta,
  truegamma, randeffect, yfn, cfn, mfn)
```

All options have the same meanings as in `SimDataC()`, while `SimDataO()` has one more option `truetheta`, which sets the true values of the non-proportional odds longitudinal coefficients subset.

The following example generates the datasets used in simulation study in Li et al. (2010):

```
require(JMcmprsk)
set.seed(123)
yfn="jmosimy1.txt";
cfn="jmosimc1.txt";
```

```
mfn="jmosimm1.txt";
 k_val=500;p1_val=3;p1a_val=1; p2_val=2;g_val=2;
truebeta=c(-1,1.5,0.8);truetheta=c(-0.5,1);truegamma=c(0.8,-1,0.5,-1); randeffect=c(1,0.5,0.5);
#writing files
SimDataO(k_val, p1_val, p1a_val, p2_val, g_val,
      truebeta, truetheta, truegamma, randeffect, yfn,  cfn,  mfn)
```

The output of the above function is

```
$`censoring_rate`
[1] 0.218
$rate1
[1] 0.414
$rate2
[1] 0.368
$yfn
[1] "jmosimy1.txt"
$cfn
[1] "jmosimc1.txt"
$mfn
[1] "jmosimm1.txt"
```

Conclusions and Future Work

In this paper, we have illustrated the capabilities of package **JMcmprsk** for fitting joint models of time-to-event data with competing risks for two types of longitudinal data. We also present simulation tools to generate joint model datasets under different settings. Several extensions of **JMcmprsk** package are planned to further expand on what is currently available. First, as the integral over the random effects becomes computationally burdensome in the case of high dimensionality, Laplace approximations or other Gauss-Hermite quadrature rules would be applied to the E-M step to speed up the computation procedure. Second, with the increasing need for predictive tools for personalized medicine, dynamic predictions for the aforementioned joint models will be added. Third, other new joint models such as joint analysis for bivariate longitudinal ordinal outcomes will be included.

Acknowledgements

We thank the reviewers for their insightful and constructive comments that led to significant improvements in our paper. The research of Hong Wang was partly supported by the National Social Science Foundation of China (17BTJ019). The research of Gang Li was partly supported by the National Institute of Health Grants P30 CA-16042, UL1TR000124-02, and P01AT003960.

Bibliography

C. Armero, C. Forné, M. Rué, A. Forte, H. Perpiñán, G. Gómez, and M. Baré. Bayesian joint ordinal and survival modeling for breast cancer risk assessment. *Statistics in medicine*, 35(28):5267–5282, 2016. [p53]

D. Eddelbuettel, R. François, J. Allaire, K. Ushey, Q. Kou, N. Russel, J. Chambers, and D. Bates. Rcpp: Seamless r and c++ integration. *Journal of Statistical Software*, 40(8):1–18, 2011. [p54, 55]

R. M. Elashoff, G. Li, and N. Li. A joint model for longitudinal measurements and survival data in the presence of multiple failure types. *Biometrics*, 64(3):762–771, 2008. [p53, 54, 65, 66]

P. L. Ferrer. *Joint modelling and prediction of several risks of cancer progression from repeated measurements of biomarkers*. PhD thesis, University of Bordeaux, 2017. [p53]

M. Galassi, J. Davies, J. Theiler, B. Gough, G. Jungman, P. Alken, M. Booth, F. Rossi, and R. Ulerich. *GNU scientific library*. Network Theory Limited, 2002. [p54, 55]

G. L. Hickey, P. Philipson, A. Jorgensen, and R. Kolamunnage-Dona. A comparison of joint models for longitudinal and competing risks data, with application to an epilepsy drug randomized controlled trial. *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, 181(4):1105–1123, 2018a. [p54]

G. L. Hickey, P. Philipson, A. Jorgensen, and R. Kolamunnage-Dona. Joint models of longitudinal and time-to-event data with more than one event time outcome: a review. *The international journal of biostatistics*, 14(1), 2018b. [p53]

N. Li, R. M. Elashoff, G. Li, and J. Saver. Joint modeling of longitudinal ordinal data and competing risks survival times and analysis of the ninds rt-pa stroke trial. *Statistics in medicine*, 29(5):546–557, 2010. [p53, 54, 65, 66]

G. Papageorgiou, K. Mauff, A. Tomer, and D. Rizopoulos. An overview of joint modeling of time-to-event and longitudinal outcomes. *Annual review of statistics and its application*, 6:223–240, 2019. [p53]

C. Proust-Lima, M. Séne, J. M. Taylor, and H. Jacqmin-Gadda. Joint latent class models for longitudinal and time-to-event data: A review. *Statistical methods in medical research*, 23(1):74–90, 2014. [p53]

C. Proust-Lima, V. Philipps, and B. Liquet. Estimation of extended mixed models using latent classes and latent processes: The r package lcmm. *Journal of Statistical Software*, 78(1):1–56, 2017. [p53]

D. Rizopoulos. *Joint models for longitudinal and time-to-event data: With applications in R*. Chapman and Hall/CRC, 2012. [p53, 54]

N. rt PA Stroke Study. Tissue plasminogen activator for acute ischemic stroke. *New England Journal of Medicine*, 333(24):1581–1588, 1995. [p59]

D. P. Tashkin, R. Elashoff, P. J. Clements, J. Goldin, M. D. Roth, D. E. Furst, E. Arriola, R. Silver, C. Strange, M. Bolster, et al. Cyclophosphamide versus placebo in scleroderma lung disease. *New England Journal of Medicine*, 354(25):2655–2666, 2006. [p56]

A. A. Tsiatis and M. Davidian. Joint modeling of longitudinal and time-to-event data: an overview. *Statistica Sinica*, pages 809–834, 2004. [p53]

P. Williamson, R. Kolamunnage-Dona, P. Philipson, and A. Marson. Joint modelling of longitudinal and competing risks data. *Statistics in medicine*, 27(30):6426–6438, 2008. [p53, 54]

Hong Wang
School of Mathematics & Statistics,Central South University
Changsha, Hunan Provinice,410075
China
wh@csu.edu.cn

Ning Li
UCLA Biomathematics
Los Angeles, CA 90095-1772
USA
nli@biomath.ucla.edu

Shanpeng Li
Department of Biostatistics,UCLA School of Public Health
Los Angeles, CA 90095-1772
USA
lishanpeng0913@ucla.edu

*Gang Li**
Department of Biostatistics,UCLA School of Public Health
Los Angeles, CA 90095-1772
USA
(Corresponding Author)
vli@ucla.edu

Wide-to-tall Data Reshaping Using Regular Expressions and the nc Package

by Toby Dylan Hocking

Abstract Regular expressions are powerful tools for extracting tables from non-tabular text data. Capturing regular expressions that describe the information to extract from column names can be especially useful when reshaping a data table from wide (few rows with many regularly named columns) to tall (fewer columns with more rows). We present the R package **nc** (short for named capture), which provides functions for wide-to-tall data reshaping using regular expressions. We describe the main new ideas of **nc**, and provide detailed comparisons with related R packages (**stats**, **utils**, **data.table**, **tidyr**, **tidyfast**, **tidyfst**, **reshape2**, **cdata**).

Introduction

Regular expressions are powerful tools for text processing that are available in many programming languages, including R. A regular expression *pattern* or *regex* defines a set of *matches* in a *subject* string. For some example subjects, consider the column names of the famous iris data set in R: `Species`, `Sepal.Length`, `Petal.Width`, etc. Some example patterns: a dot between square brackets `[.]` matches a period, a dot by itself `.` matches any non-newline character, and a dot followed by a star `.*` matches zero or more non-newline characters. Therefore the pattern `.*[.].*` matches zero or more non-newline characters, followed by a period, followed by zero or more non-newline characters. It would match `Sepal.Length` and `Petal.Width`, but it would not match `Species`. For a more detailed discussion of regular expressions, we refer the reader to `help(regex)` in R or the book of Friedl (2002).

The focus of this article is patterns with capture groups, which are typically defined using parentheses. For example, the pattern `(.*)[.](.*)` results in the same matches as the pattern in the previous paragraph, and it additionally allows the user to capture and extract the substrings by group index (e.g., group 1 matches `Sepal`, group 2 matches `Length`).

Named capture groups allow extracting the substring by name rather than by index. Using names rather than indices is preferable in order to create more readable regular expressions (names document the purpose of each sub-pattern) and to create more readable R code (it is easier to understand the intent of named references than numbered references). For example, the pattern `(?<part>.*)[.](?<dimension>.*)` documents that the flower part appears before the measurement dimension; the `part` group matches `Sepal` and the `dimension` group matches `Length`.

Recently, Hocking (2019a) proposes a new syntax for defining named capture groups in R code. Using this new syntax, named capture groups are specified using named arguments in R, which results in code that is easier to read and modify than capture groups defined in string literals. For example, the pattern in the previous paragraph can be written as `part = ".*", "[.]", dimension = ".*"`. Sub-patterns can be grouped for clarity and/or re-used using lists, and numeric data may be extracted with user-provided type conversion functions.

The main thesis of this article is that regular expressions can greatly simplify the code required to specify wide-to-tall data reshaping operations (when the input columns adhere to a regular naming convention). For one such operation, the input is a "wide" table with many columns, and the desired output is a "tall" table with more rows, and some of the input columns are converted into a smaller number of output columns (Figure 1). To clarify the discussion, we first define three terms that we will use to refer to the different types of columns involved in this conversion:

Reshape columns contain the data which is present in the same amount but in different shapes in the input and output. There are equivalent terms used in different R packages: `varying` in `utils::reshape`, `measure.vars` in `melt` (**data.table**, **reshape2**), etc.

Copy columns contain data in the input which are each copied to multiple rows in the output (`id.vars` in `melt`).

Capture columns are only present in the output, and contain data which come from matching a capturing regex pattern to the input reshape column names.

For example, the wide iris data (W in Figure 1) have four numeric columns to reshape: `Sepal.Length`, `Sepal.Width`, `Petal.Length`, `Petal.Width`. For some purposes (e.g., displaying a histogram of each reshape input column using facets in **ggplot2**), the desired reshaping operation results in a table with a single reshape output column (S in Figure 1), two copied columns, and two columns captured from the names of the reshaped input columns. For other purposes (e.g., scatterplot to compare sepal and

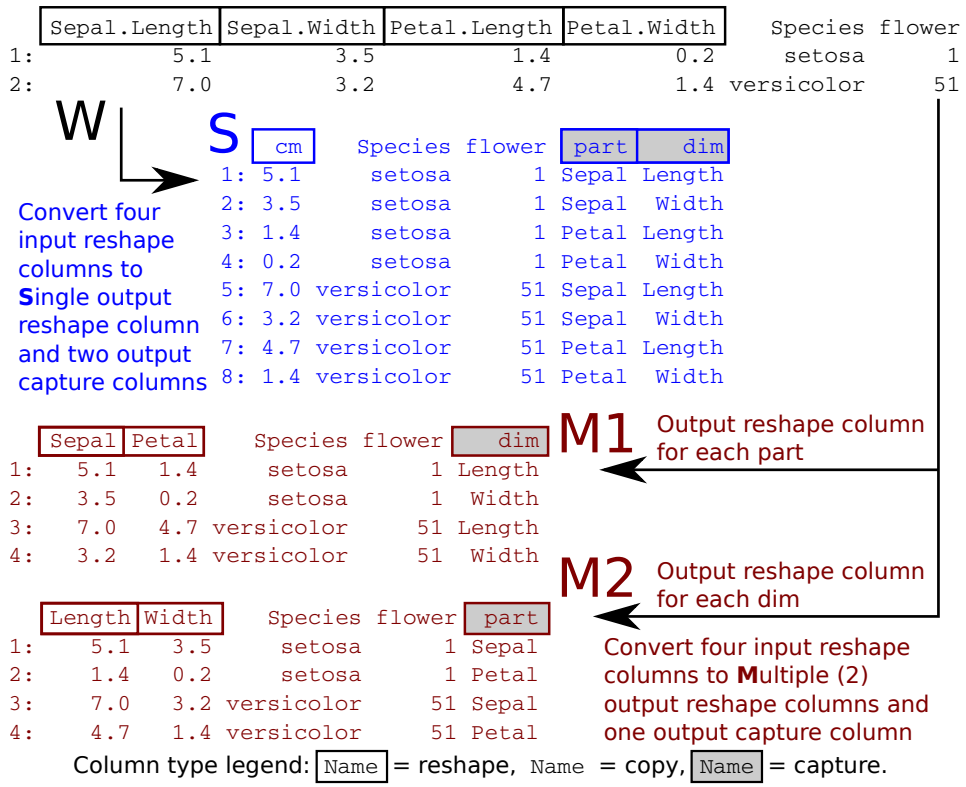


Figure 1: Two rows of the iris data set (W, black) are considered as the input to a wide-to-tall reshape operation. Four input reshape columns are converted to either a single output reshape column (S, blue) or multiple (2) output reshape columns (M1, M2, red). Other output columns are either copied from the non-reshaped input data, or captured from the names of the reshaped input columns.

petal sizes) the desired reshaping operation results in a table with multiple reshape output columns (M1 with `Sepal` and `Petal` columns in Figure 1), two copied columns, and one column captured from the names of the reshaped input columns.

In this article, our original contribution is the R package **nc** which provides a new implementation of the previously proposed named capture regex syntax of Hocking (2019a), in addition to several new functions that perform wide-to-tall data reshaping using regular expressions. The main new idea is to use a single named capture regular expression for defining both (1) the subset of reshape input columns to convert and (2) the additional capture output columns. We will show that this results in a simple, powerful, non-repetitive syntax for wide-to-tall data reshaping. A secondary contribution of this article is a detailed comparison of current R functions for wide-to-tall data reshaping in terms of syntax, computation times, and functionality (Table 1). Note that in this article, we do not discuss tall-to-wide data reshaping, because regular expressions are not useful in that case.

The organization of this article is as follows. The rest of this introduction provides an overview of current R packages for regular expressions and data reshaping. The second section describes the proposed functions of the **nc** package, and then the third section provides detailed comparisons with other R packages. The article concludes with a summary and discussion of possible future work.

Related work

There are many R functions which can extract tables from non-tabular text using regular expressions. Recommended R package functions include `base::regexpr` and `base::gregexpr` as well as `utils::strcapture`. CRAN packages which provide various functions for text processing using regular expressions include **namedCapture** (Hocking, 2019b), **rematch2** (Csárdi, 2017), **rex** (Ushey et al., 2017), **stringr** (Wickham, 2018), **stringi** (Gagolewski, 2018), **tidyr** (Wickham and Henry, 2018), and **re2r** (Wenfeng, 2017). We refer the reader to our previous research paper for a detailed comparison of these packages (Hocking, 2019a).

For reshaping data from wide (one row with many columns) to tall (one column with many rows), there are several different R functions that provide similar functionality. Each function supports a different set of features (Table 1); each feature/column is explained in detail below:

pkg::function	single	multiple	regex	na.rm	types	list
nc::capture_melt_multiple	no	yes	capture	yes	any	yes
nc::capture_melt_single	yes	no	capture	yes	any	yes
tidyr::pivot_longer	yes	yes	capture	yes	any	yes
stats::reshape	yes	if sorted	capture	no	some	no
data.table::melt, patterns	yes	if sorted	match	yes	no	yes
tidyfst::longer_dt	yes	no	match	yes	no	yes
tidyr::gather	yes	no	no	yes	some	yes
tidyfast::dt_pivot_longer	yes	no	no	yes	no	yes
cdata::rowrecs_to_blocks	yes	yes	no	no	no	yes
cdata::unpivot_to_blocks	yes	no	no	no	no	yes
reshape2::melt	yes	no	no	yes	no	no
utils::stack	yes	no	no	no	no	no

Table 1: Reshaping functions in R support various features: "single" for converting input columns into a single output column; "multiple" for converting input columns (either "if sorted" in a regular order, or "yes" for any order) into multiple output columns of possibly different types; "regex" for regular expressions to "match" input column names or to "capture" and create new output column names; "na.rm" for removal of missing values; "types" for converting input column names to non-character output columns; "list" for output of list columns.

single refers to support for converting input reshape columns of the same type to a single reshape output column.

multiple refers to support for converting input reshape columns of possibly different types to multiple output reshape columns; "if sorted" means that conversion works correctly only if the input reshape columns are sorted in a regular order, e.g., `Sepal.Length`, `Sepal.Width`, `Petal.Length`, `Petal.Width`; "yes" means that conversion works correctly even if they are not sorted, e.g., `Sepal.Length`, `Sepal.Width`, `Petal.Width`, `Petal.Length`.

regex refers to support for regular expressions; "match" means a pattern is used to match the input column names; "capture" means that the specified pattern is used to create new output capture columns — this is especially useful when the names consist of several distinct pieces of information, e.g., `Sepal.Length`; "no" means that regular expressions are not directly supported (although `base::grep` can always be used).

na.rm refers to support for removing missing values.

types refers to support for converting captured text to numeric output columns.

list refers to support for output of list columns.

Recommended R package functions include `stats::reshape` and `utils::stack` for reshaping data from wide to tall. Of the features listed in Table 1, `utils::stack` only supports output with a single reshape column, whereas `stats::reshape` supports the following features. For data with regular input column names (output column, separator, time value), regular expressions can be used to specify the separator (e.g., in `Sepal.Length`, `Sepal` is output column, dot is separator, `Length` is time value). Multiple output columns are supported, but incorrect output may be computed if input columns are not sorted in a regular order. The time value is output to a capture column named `time` by default. Automatic type conversion is performed on time values when possible, but custom type conversion functions are not supported. There is neither support for missing value removal nor list column output.

The **tidyr** package provides two functions for reshaping data from wide to tall format: `gather` and `pivot_longer`. The older `gather` function only supports converting input reshape columns to a single output reshape column (not multiple). The input reshape columns to convert may not be directly specified using regular expressions; instead, R expressions such as `x:y` can be used to indicate all columns starting from `x` and ending with `y`. It does support limited type conversion; if the `convert = TRUE` argument is specified, the `utils::type.convert` function is used to convert the input column names to numeric, integer, or logical. In contrast, the newer `pivot_longer` also supports multiple output reshape columns (even if input reshape columns are unsorted) and regular expressions for specifying output capture columns (but to specify input reshape columns with a regex, `grep` must be used). Arbitrary type conversion is also supported in `pivot_longer`, via the `names_transform` argument, which should be a named list of conversion functions. Both functions support list columns and removing missing values, although different arguments are used (`na.rm` for `gather`, `values_drop_na` for `pivot_longer`).

The **reshape2** and **data.table** packages each provide a `melt` function for converting data from wide to tall (Wickham, 2007; Dowle and Srinivasan, 2019). The older **reshape2** version only supports converting input reshape columns to a single output reshape column, whereas the newer **data.table** version also supports multiple output reshape columns. Regular expressions are not supported in **reshape2**, but can be used with `data.table::patterns` to match input column names to convert (although the output can be incorrect if columns are not sorted in a regular order). Neither function supports type conversion, and both functions support removing missing values from the output using the `na.rm` argument. List column output is supported in **data.table** but not **reshape2**. The **tidyfast** (Barrett, 2020) and **tidyfst** (Huang and Zhao, 2020) packages provide reshaping functions that use `data.table::melt` internally (but do not support multiple output reshape columns).

The **cdata** package provides several functions for data reshaping, including `rowrecs_to_blocks` and `unpivot_to_blocks`, which can convert data from wide to tall (Mount and Zumel, 2019). The simpler of the two functions is `unpivot_to_blocks`, which supports a single output reshape column (interface similar to `reshape2::melt/tidyr::gather`). The user of `rowrecs_to_blocks` must provide a control table that describes how the input should be reshaped into the output. It, therefore, supports multiple output reshape columns for possibly unsorted input columns. Both functions support list column output, but other features from Table 1 are not supported (regular expressions, missing value removal, type conversion).

Basic features for wide-to-tall data reshaping using regular expressions

The **nc** package provides new regular expression functionality based on the syntax recently proposed by Hocking (2019a). During the rest of the article, we give only a brief overview of this syntax; for a more detailed review, please read the **nc** package vignettes. In this section, we show how new **nc** functions can be used to reshape wide data (with many columns) to tall data (with fewer columns, and more rows). We begin by considering the two data visualization problems which were mentioned in the introduction and which involve the familiar iris data set.

Single reshape output column

First, suppose we would like to visualize the univariate distribution of each numeric variable. One way would be to use a histogram of each numeric variable, with row facets for the flower part and column facets for the measurement dimension. Our desired output, therefore, needs a single column with all of the reshaped numeric data to plot (Figure 1, W→S).

We can perform this operation using `nc::capture_melt_single`, which inputs a data frame and a pattern which should match the names of the input columns to reshape. Any input columns with names that do not match the pattern are considered copy columns; the output also contains a capture column for each group specified in the pattern:

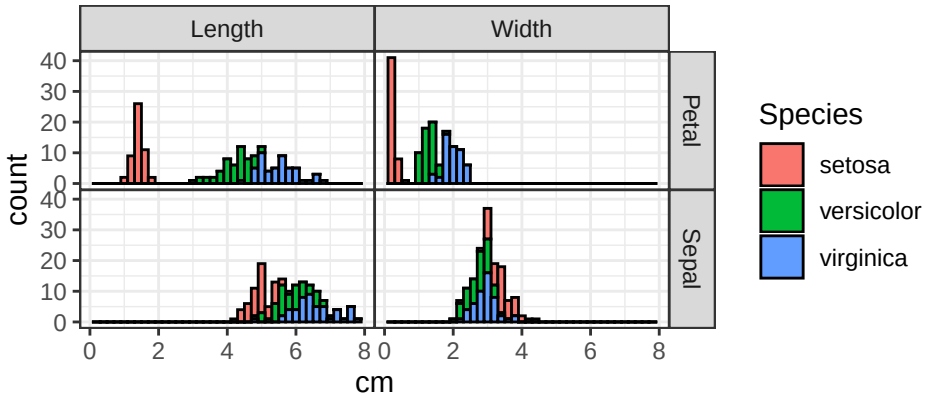
```
> (iris.tall.single <- nc::capture_melt_single(
+   iris, part = ".*", "[.]", dim = ".*", value.name = "cm"))

       Species  part    dim  cm
  1:    setosa Sepal Length 5.1
  2:    setosa Sepal Length 4.9
  3:    setosa Sepal Length 4.7
  4:    setosa Sepal Length 4.6
  5:    setosa Sepal Length 5.0
 ---
596: virginica Petal  Width 2.3
597: virginica Petal  Width 1.9
598: virginica Petal  Width 2.0
599: virginica Petal  Width 2.3
600: virginica Petal  Width 1.8
```

The code above can be read as follows. The first argument, `iris` specifies the wide input to reshape (a data frame or data table). The next three arguments (`part = ".*", "[.]", dim = ".*"`) specify the regex. Internally **nc** generates a capture group for each named argument, so the generated regex pattern is `(.*)[.](.*)` in this example. The `value.name` argument is not considered part of the regex and instead specifies the name of the output reshape column.

The output above is a data table (a data frame subclass with special methods with reference semantics) because `data.table::melt` is used internally for the reshape operation. The output data

table consists of one copy column (`Species`), two capture columns (`part`, `dim`), and a single reshape column (`cm`). These data can be used to create the desired histogram with **ggplot2** via:

```
> library(ggplot2)
> ggplot(iris.tall.single) + facet_grid(part ~ dim) +
+   theme_bw() + theme(panel.spacing = grid::unit(0, "lines")) +
+   geom_histogram(aes(cm, fill = Species), color = "black", bins = 40)
```



For comparison, we show how the same reshape operation can be accomplished with the **data.table** package:

```
> iris.pattern <- "(.*)[.](.*)"
> iris.wide <- data.table::as.data.table(iris)
> iris.tall <- data.table::melt(
+   iris.wide, measure = patterns(iris.pattern), value.name = "cm")
> iris.tall[, `:=`(part = sub(iris.pattern, "\\1", variable),
+                  dim  = sub(iris.pattern, "\\2", variable))][]

       Species     variable  cm  part    dim
  1:    setosa Sepal.Length 5.1 Sepal Length
  2:    setosa Sepal.Length 4.9 Sepal Length
  3:    setosa Sepal.Length 4.7 Sepal Length
  4:    setosa Sepal.Length 4.6 Sepal Length
  5:    setosa Sepal.Length 5.0 Sepal Length
 ---
596: virginica  Petal.Width 2.3 Petal  Width
597: virginica  Petal.Width 1.9 Petal  Width
598: virginica  Petal.Width 2.0 Petal  Width
599: virginica  Petal.Width 2.3 Petal  Width
600: virginica  Petal.Width 1.8 Petal  Width
```

The code above uses `data.table::melt` with `patterns` which takes a regex used to specify the four columns to reshape. The `part` and `dim` capture columns must be created during a post-processing step. In this case, the **nc** code is substantially simpler because the named capture regular expression was used to specify both the input columns to reshape and the capture columns to output.

Finally we show how the same reshape operation could be done using the **tidyr** package:

```
> tidyr::pivot_longer(iris, matches(iris.pattern), values_to = "cm",
+   names_to=c("part", "dim"), names_pattern=iris.pattern)

# A tibble: 600 x 4
   Species part  dim       cm
   <fct>   <chr> <chr>  <dbl>
 1 setosa  Sepal Length   5.1
 2 setosa  Sepal Width    3.5
 3 setosa  Petal Length   1.4
 4 setosa  Petal Width    0.2
 5 setosa  Sepal Length   4.9
 6 setosa  Sepal Width    3
 7 setosa  Petal Length   1.4
 8 setosa  Petal Width    0.2
```

```
 9 setosa  Sepal Length   4.7
10 setosa  Sepal Width    3.2
# . . .  with 590 more rows
```

The code above is almost as simple as the corresponding **nc** code, but with one key difference. The output capture column names are defined in the `names_to` argument, which is far away from the definition of the groups in `iris.pattern`. In this simple example with two groups in the regex this separation of related concepts is not a huge problem, but the **nc** syntax should be preferred for more complex patterns (with more groups) in order to keep the group names and sub-patterns closer and easier to maintain/read in the code.

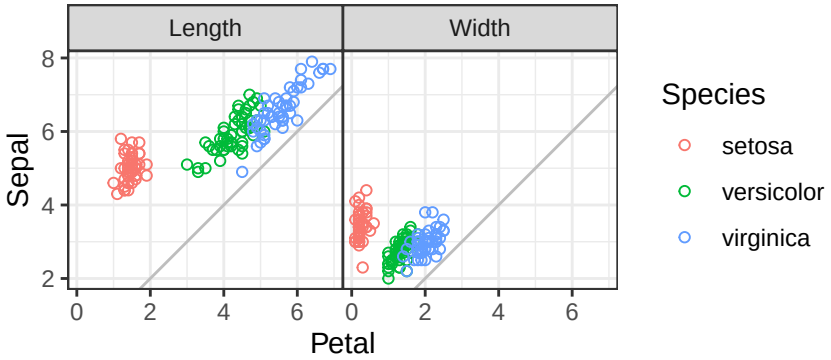
Multiple reshape output columns

For the second data reshaping task, suppose we want to determine whether or not sepals are larger than petals for each measurement dimension and species. We could use a scatterplot of sepal versus petal, with a facet for measurement dimension. We, therefore, need a data table with two reshape output columns: a `Sepal` column to plot against a `Petal` column (Figure 1, W→M1). We can perform this operation using another function, `nc::capture_melt_multiple`, which inputs a data frame and a pattern which must contain the special `column` group and at least one other named group:

```
> (iris.parts <- nc::capture_melt_multiple(iris, column = ".*", "[.]", dim = ".*"))

       Species    dim Petal Sepal
  1:    setosa Length   1.4   5.1
  2:    setosa Length   1.4   4.9
  3:    setosa Length   1.3   4.7
  4:    setosa Length   1.5   4.6
  5:    setosa Length   1.4   5.0
 ---
296: virginica  Width   2.3   3.0
297: virginica  Width   1.9   2.5
298: virginica  Width   2.0   3.0
299: virginica  Width   2.3   3.4
300: virginica  Width   1.8   3.0
```

Again, any input columns with names that do not match the pattern are considered copy columns (`Species` in the example above). Each unique value captured in the special `column` group becomes the name of an output reshape column (`Petal`, `Sepal`); other groups are used to create output capture columns (`dim`). These data can be used to create the scatterplot using **ggplot2** via:

```
> ggplot(iris.parts) + facet_grid(. ~ dim) +
+   theme_bw() + theme(panel.spacing = grid::unit(0, "lines")) +
+   coord_equal() + geom_abline(slope = 1, intercept = 0, color = "grey") +
+   geom_point(aes(Petal, Sepal, color = Species), shape = 1)
```



For comparison, we show how to output a data table with multiple reshape output columns using the **data.table** and **tidyr** packages:

```
> iris.multiple <- data.table::melt(
+   iris.wide, measure = patterns(Petal="Petal", Sepal="Sepal"))
> iris.multiple[, dim := c("Length", "Width")[variable] ]
```

```
       Species variable Petal Sepal    dim
  1:    setosa        1   1.4   5.1 Length
  2:    setosa        1   1.4   4.9 Length
  3:    setosa        1   1.3   4.7 Length
  4:    setosa        1   1.5   4.6 Length
  5:    setosa        1   1.4   5.0 Length
 ---
296: virginica        2   2.3   3.0  Width
297: virginica        2   1.9   2.5  Width
298: virginica        2   2.0   3.0  Width
299: virginica        2   2.3   3.4  Width
300: virginica        2   1.8   3.0  Width
```

```
> tidyr::pivot_longer(iris, matches(iris.pattern), values_to = "cm",
+   names_to=c(".value", "dim"), names_pattern=iris.pattern)
```

```
# A tibble: 300 x 4
   Species dim    Sepal Petal
   <fct>   <chr>  <dbl> <dbl>
 1 setosa  Length   5.1   1.4
 2 setosa  Width    3.5   0.2
 3 setosa  Length   4.9   1.4
 4 setosa  Width    3     0.2
 5 setosa  Length   4.7   1.3
 6 setosa  Width    3.2   0.2
 7 setosa  Length   4.6   1.5
 8 setosa  Width    3.1   0.2
 9 setosa  Length   5     1.4
10 setosa  Width    3.6   0.2
# . . .  with 290 more rows
```

The code above computes equivalent results but suffers from the same drawbacks as discussed in the previous section (repetition, separation of pattern and group names).

To conclude this section, **nc** provides two new functions for data reshaping using regular expressions. Both functions input a data frame to reshape and a pattern to match with the column names. For `nc::capture_melt_single`, all matching input columns are reshaped in the output to a single column which is named using the `value.name` argument. For `nc::capture_melt_multiple` the output is multiple reshape columns with names defined by the values captured in the special `column` group. Values from other groups are stored in capture columns in the output. Both functions support the output of numeric capture columns via user-specified type conversion functions, as we will see in the next section.

Comparisons which highlight differences with other packages

In this section, we compare the new data reshaping functions in the **nc** package with similar functions in other packages. We aim to demonstrate that the new **nc** syntax is often more convenient and less repetitive without sacrificing speed.

Building a complex pattern from smaller sub-patterns

In terms of functionality for wide-to-tall data reshaping, the most similar package to **nc** is **tidyr** (Table 1). One advantage of **nc** is that complex patterns may be defined in terms of simpler sub-patterns, which can include group names and type conversion functions. Integrating these three pieces results in a syntax that is easy to read as well; it is more difficult to build and read complex patterns using **tidyr** syntax, which requires specifying regex pattern strings, group names, and types as separate arguments. For example, consider a data set from the World Health Organization (WHO):

```
> data(who, package = "tidyr")
> set.seed(1);sample(names(who), 10)
```

```
 [1] "newrel_f3544" "year"         "new_ep_m65"   "country"      "new_ep_m1524"
 [6] "new_sn_m4554" "new_ep_f3544" "new_sp_f2534" "new_sp_f65"   "newrel_m4554"
```

```
>
```

Each reshape column name starts with new and has three distinct pieces of information: diagnosis type (e.g., ep, rel), gender (m or f), and age range (e.g., 1524, 4554). We extract all three pieces of information below and include a function for converting gender to a factor with levels in a specific (non-default) order:

```
> nc.who.sub.pattern <- list(
+   "new_?", diagnosis = ".*", "_",
+   gender = ".", function(mf)factor(mf, c("m", "f")))
> nc.who.ages <- nc::capture_melt_single(who, nc.who.sub.pattern, ages = ".*")
> print(nc.who.ages[1:2], class = TRUE)

       country   iso2   iso3  year diagnosis gender   ages value
        <char> <char> <char> <int>    <char> <fctr> <char> <int>
1: Afghanistan     AF    AFG  1997        sp      m    014     0
2: Afghanistan     AF    AFG  1998        sp      m    014    30
```

First, note that nc.who.sub.pattern is a sub-pattern list variable that we have used as the first part of the pattern in the call to nc::capture_melt_single above (and we will use that sub-pattern again below). Sub-pattern lists may contain regex character strings (patterns to match), functions (for converting the previous capture group), or other sub-pattern lists. The reshaped output is a data table with gender converted to a factor — this can also be done using tidyr::pivot_longer:

```
> tidyr.who.sub.names <- c("diagnosis", "gender")               #L0
> tidyr.who.sub.pattern <- "new_?(.*)_(.)"                      #L1
> tidyr.who.pattern <- paste0(tidyr.who.sub.pattern, "(.*)")    #L2
> tidyr::pivot_longer(                                         #L3
+   who, cols = matches(tidyr.who.pattern),                    #L4
+   names_to = c(tidyr.who.sub.names, "ages"),                 #L5
+   names_ptypes = list(gender = factor(levels = c("m", "f"))), #L6
+   names_pattern = tidyr.who.pattern)[1:2,]                   #L7

# A tibble: 2 x 8
  country     iso2  iso3   year diagnosis gender ages  value
  <chr>       <chr> <chr> <int> <chr>     <fct>  <chr> <int>
1 Afghanistan AF    AFG    1980 sp        m      014      NA
2 Afghanistan AF    AFG    1980 sp        m      1524     NA
```

In the code above, we first define a sub-pattern variable for the diagnosis and gender capture groups, as we did using **nc**. One difference is that the **tidyr** sub-pattern variable is a string with un-named capture groups, whereas the **nc** sub-pattern variable is a list which includes capture group names as well as a type conversion function. These three parameters are specified as three separate arguments in **tidyr**, which results in some separation (e.g., group names defined on L0 and L5 but corresponding sub-patterns defined on L1 and L2) and repetition (e.g., gender appears on L0 and L6) in the code. The pattern also must be repeated: first in the cols argument (L4) to specify the set of input reshape columns, second in the names_pattern argument (L7) to specify the conversion from input reshape column names to output capture column values.

Now suppose we want to extract two numeric columns from ages, for example, to use as interval-censored outputs in a survival regression. Using **nc** we can use the previously defined sub-pattern (including the previously defined group names and type conversion function) as the first part of a larger pattern:

```
> who.typed <- nc::capture_melt_single(who, nc.who.sub.pattern, ages = list(
+   ymin = "0|[0-9]{2}", as.numeric,
+   ymax = "[0-9]{0,2}", function(x)ifelse(x == "", Inf, as.numeric(x))))
> who.typed[1:2]

       country iso2 iso3 year diagnosis gender ages ymin ymax value
1: Afghanistan   AF  AFG 1997        sp      m  014    0   14     0
2: Afghanistan   AF  AFG 1998        sp      m  014    0   14    30

> who.typed[, .(rows = .N), by = .(ages, ymin, ymax)]

   ages ymin ymax  rows
1:  014    0   14 10882
2: 1524   15   24 10868
3: 2534   25   34 10850
```

```
4: 3544   35   44 10875
5: 4554   45   54 10876
6: 5564   55   64 10851
7:   65   65  Inf 10844
```

Note in the code above that each group name, regex pattern string, and the corresponding type conversion function appears on the same line — this syntax keeps these three related pieces of information close together, which makes complex patterns easier to read and build from smaller pieces. Also, note how an anonymous function is used to convert the values captured in the `ymax` group to numeric (and it maps the empty string to `Inf`). Such custom type conversion functions are supported by **tidyr** since version 1.1.0 (early 2020), so we can do:

```
> tidyr.who.range.pattern <- paste0(tidyr.who.sub.pattern, "((0|[0-9]{2})([0-9]{0,2}))")
> tidyr::pivot_longer(
+   who, cols = matches(tidyr.who.range.pattern),
+   names_to = c(tidyr.who.sub.names, "ages", "ymin", "ymax"),
+   names_transform = list(
+     gender = function(x)factor(x, levels = c("m", "f")),
+     ymin = as.numeric,
+     ymax = function(x)ifelse(x == "", Inf, as.numeric(x))),
+   names_pattern = tidyr.who.range.pattern)[1:7,]

# A tibble: 7 x 10
  country     iso2  iso3   year diagnosis gender ages   ymin  ymax value
  <chr>       <chr> <chr> <int> <chr>     <fct>  <chr> <dbl> <dbl> <int>
1 Afghanistan AF    AFG    1980 sp        m      014       0    14    NA
2 Afghanistan AF    AFG    1980 sp        m      1524     15    24    NA
3 Afghanistan AF    AFG    1980 sp        m      2534     25    34    NA
4 Afghanistan AF    AFG    1980 sp        m      3544     35    44    NA
5 Afghanistan AF    AFG    1980 sp        m      4554     45    54    NA
6 Afghanistan AF    AFG    1980 sp        m      5564     55    64    NA
7 Afghanistan AF    AFG    1980 sp        m      65       65   Inf    NA
```

The code above uses the `names_transform` argument to define type conversion functions, which requires some repetition (e.g., `ymax` and `ymin` each appear twice).

To conclude this comparison, we have seen that **nc** syntax makes it easy to read and write complex patterns because it keeps group-specific names and type conversion functions near the corresponding sub-patterns. We have also shown that repetition is often necessary with **tidyr** (e.g., pattern, group names), whereas such repetition can be avoided by using **nc**.

Comparison with other packages which support multiple reshape output columns

In this section, we demonstrate the advantages of using **nc** over several alternatives which support multiple reshape output columns. A major advantage is that **nc** directly supports regular expressions for defining the input reshape columns and output capture columns. Another advantage is that **nc** always returns a correct output data set with multiple reshape columns, even when the input columns are not sorted in a regular order. For example, consider the following simple data set in which the columns are not in regular order:

```
> (TC <- data.table::data.table(
+   treatment.age = 13,
+   control.gender = "M",
+   treatment.gender = "F",
+   control.age = 25))

   treatment.age control.gender treatment.gender control.age
1:            13              M               F          25
```

It is clear from the table above that the treatment group consists of a teenage female, whereas the control group consists of a male aged 25 (not the best experimental design, but easy to remember for the demonstration in this section). Assume we need an output data table with two reshape columns (`age` and `gender`) as well as a capture column (`group`). The **nc** syntax we would use is:

```
> nc::capture_melt_multiple(TC, group = ".*", "[.]", column = ".*")
```

```
       group age gender
1:   control  25      M
2: treatment  13      F
```

The correct result is computed above because **nc** reshapes based on the input column names (the order of the input columns is not relevant). A naïve user may attempt to perform this reshape using `data.table::patterns`:

```
> data.table::melt(TC, measure.vars = patterns(age = "age", gender = "gender"))

   variable age gender
1:        1  13      M
2:        2  25      F
```

First, note that the syntax above requires repetition of `age` and `gender` (in names and in pattern strings). Also, it is clear that the result is incorrect! Actually, the `patterns` function is working as documented; it "returns the matching indices" of the provided regex. However, since the input columns are not sorted in regular order, `melt` returns an incorrect result (this is an incorrect use of these functions, not a bug). To get a correct result, we can provide a list of index vectors:

```
> data.table::melt(TC, measure.vars = list(age = c(1,4), gender = c(3,2)))

   variable age gender
1:        1  13      F
2:        2  25      M
```

This is what **nc** does internally; it also converts the `variable` output column to a more interpretable/useful capture column (e.g., `group` above).

The `stats::reshape` function suffers from the same issue as the `patterns` usage above. Another issue with this function is that it assumes the output reshape column names are the first part of the input column names (e.g., Figure 1, W→M1). When input column names have a different structure (e.g., Figure 1, W→M2), they must be renamed, putting the desired output reshape column names first:

```
> TC.renamed <- structure(TC, names = sub("(.*)[.](.*)", "\\2.\\1", names(TC)))
> stats::reshape(TC.renamed, 1:4, direction = "long", timevar = "group")

       group age gender id
1: treatment  13      M  1
2:   control  25      F  1
```

However, the result above still contains incorrect results in the `gender` column. The correct result can be obtained by sorting the input column names:

```
> TC.sorted <- data.frame(TC.renamed)[, sort(names(TC.renamed))]
> stats::reshape(TC.sorted, 1:4, direction = "long", timevar = "group")

                group age gender id
1.control     control  25      M  1
1.treatment treatment  13      F  1
```

After renaming and sorting the input columns, the correct result is obtained using `stats::reshape`. Another way to obtain a correct result is with the **cdata** package:

```
> cdata::rowrecs_to_blocks(TC, controlTable = data.frame(
+   group = c("treatment", "control"),
+   age = c("treatment.age", "control.age"),
+   gender = c("treatment.gender", "control.gender"),
+   stringsAsFactors = FALSE))

      group age gender
1 treatment  13      F
2   control  25      M
```

The **cdata** package is very powerful and can handle many more types of data reshaping operations than **nc**. However, it requires a very explicit definition of the desired conversion in terms of a control table, which results in rather verbose code. In contrast, the terse regular expression syntax of **nc** is a more implicit approach, which assumes the input columns to reshape have regular names.

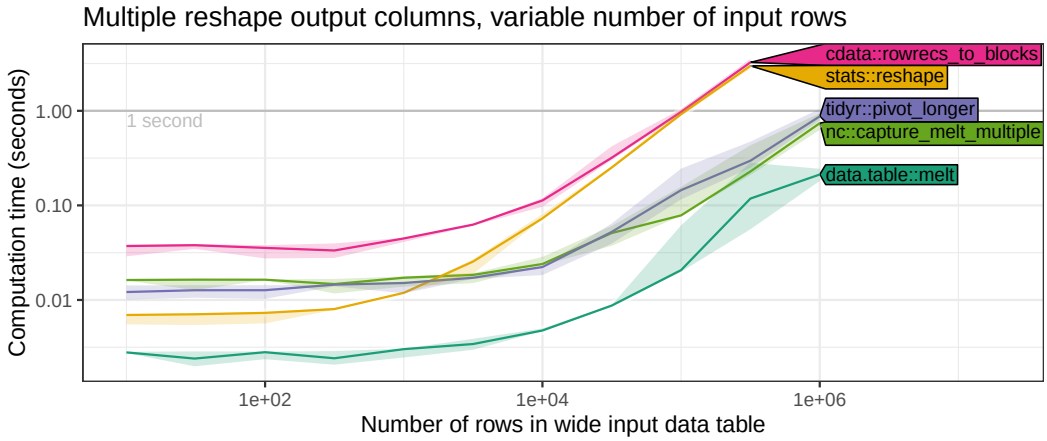


Figure 2: Timings for computing a tall output table with multiple (2) reshape columns from a wide input table with 8 reshape columns and a variable number of rows (x-axis).

To conclude this section, we have discussed some advantages of **nc** relative to other R packages. Input columns with regular names do not need to be renamed/sorted for **nc** functions, whereas renaming/sorting may be necessary using `stats::reshape`. Verbose/explicit control table code is always necessary with **cdata**, whereas a terse/implicit regular expression syntax is used with **nc** to simplify the definition of reshape operations.

Comparing computation times of functions for wide-to-tall data reshaping

In previous sections, we have shown that the **nc** package provides a convenient syntax for defining wide-to-tall reshape operations. In this section, we investigate whether this convenience comes at the cost of increased computation time. We aim to demonstrate that the computation time required for the proposed **nc** package is comparable with other packages for data reshaping. In particular, since **nc** is implemented using **data.table**, we expect that **nc** should be slightly slower than **data.table** (by only the amount of time required for regex matching). In our result figures, we show the median and quartiles over 10 timings using the **microbenchmark** package on an Intel Core i7-8700 3.20GHz processor. Note that these timings include both the regex matching (which should be relatively fast) and the data reshaping operation (which should be relatively slow). We varied the number of rows/columns in each experiment by copying/duplicating the rows/columns in each source data set.

First, we performed timings on variants of the iris data with a variable number of rows and twice the original number of reshape columns (8). The input reshape column names were of the form `day1.Sepal.Length`, `day2.Sepal.Length`, `day1.Sepal.Width`, etc. Since the desired output has two reshape columns (`Sepal` and `Petal`), we considered packages which support multiple output columns (**cdata**, **stats**, **tidyr**, **nc**, **data.table**). As expected, we observed that all algorithms have similar asymptotic time complexity (Figure 2). We observed that **nc** is slightly slower than **data.table** (by constant factors), slightly faster than the other packages (**cdata**, **stats**), and about the same speed as **tidyr**.

Second, we performed similar timings on variants of the iris data with a variable number of columns and the original number of rows (150). As in the previous experiment, we expected that all functions would have similar slopes, indicating linear asymptotic time complexity. Surprisingly, we observed on the log-log plot (Figure 3) that **cdata** has a larger asymptotic slope than the other packages, which suggests its time complexity may be super-linear in the number of columns to reshape. The other packages differed by constant factors, with **data.table** being fastest, followed by **tidyr**, **nc**, **cdata**, and finally the slowest **stats**. All packages except **stats** performed the operation in less than 1 second for 1,000 or fewer columns. This comparison confirms the expectation that **nc** speed is comparable to other packages.

Third, we performed timings on versions of the WHO data with a variable number of duplicated rows and the original number of columns (56). We ran reshaping functions from several additional packages (**utils**, **reshape2**, **tidyfast**) that can compute the desired output table with a single reshape output column. We computed the amount of time it takes to create zero or four capture output columns (with additional post-processing steps for `tidyfast::dt_pivot_longer`, `reshape2::melt`, `tidyr::gather`, `cdata::unpivot_to_blocks`). We expected that functions which require additional post-processing steps should be slower by constant factors. As we expected, all functions appear to have similar asymptotic time complexity and differ only in terms of constant factors. For zero capture

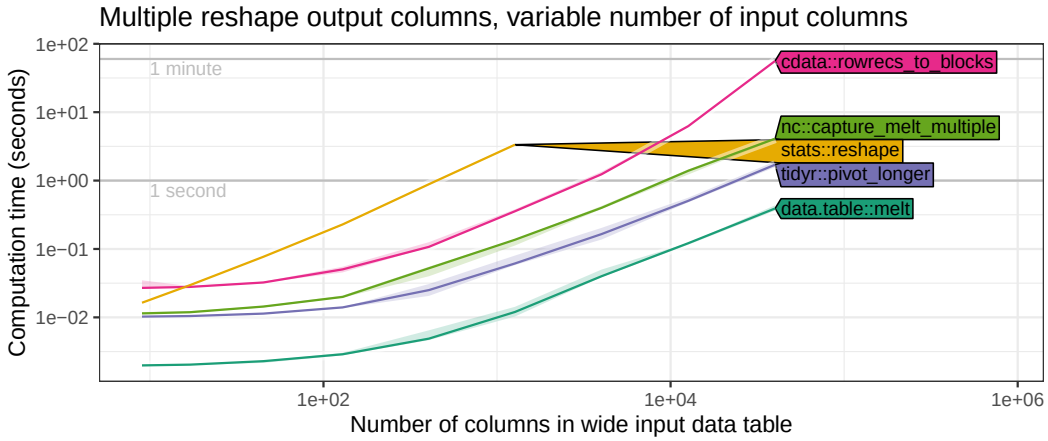


Figure 3: Timings for computing a tall output table with multiple (2) reshape columns from a wide input table with 150 rows and a variable number of columns to reshape (x-axis).

output columns, the slowest functions were `stats::reshape` and `cdata::unpivot_to_blocks`, which were the only ones to take more than one second for 10,000 input rows. The fastest functions were `data.table::melt` and `tidyfast::dt_pivot_longer` (about 10ms for 10,000 input rows). As expected, for four capture output columns, the functions which require post-processing were slower, and the fastest functions were `data.table::melt` and `nc::capture_melt_single`.

Finally, we performed similar timings on variants of the WHO data with a variable number of columns and a fixed number of rows (11). The desired output again has a single reshape output column, and we again tried computing either zero or four capture output columns. We observed timings (Figure 5) with similar asymptotic trends as in the previous comparisons. In particular, timings for most packages appear to be linear in the number of input reshape columns, and timings for **cdata** appear to be super-linear for a large number of columns. These data indicate that **nc** speed is similar to comparable R packages.

Discussion and conclusions

In this paper, we described the **nc** package and its new functions for regular expressions and data reshaping. The **nc** package allows a user to define a regular expression in R code, along with capture group names and corresponding type conversion functions. We showed how this syntax makes it easy to define complex regular expressions in terms of simpler sub-patterns, while providing a uniform interface to three regex engines (ICU, PCRE, RE2). We showed several examples of how **nc** can be used for wide-to-tall data reshaping. We provided a detailed comparison with other data reshaping functions in terms of syntax, functionality, and computation time.

In all of our speed comparisons, we observed that the speed of **nc** is similar to other R functions for wide-to-tall data reshaping. We expected that all R functions would have linear asymptotic timings, and differ only in constant factors. We were surprised to observe in our empirical timings that the **cdata** package appears to have asymptotic time complexity that is super-linear in the number of columns to reshape. This result suggests that the speed of **cdata** could be improved by adopting one of the linear time reshaping algorithms used in the other packages.

The `tidyr::pivot_longer` function provides a feature set which is most similar to **nc** data reshaping functions. We showed that both packages could perform the same data reshaping operations, but **nc** provides a syntax that reduces repetition in user code. Another advantage is that **nc** R code allows sub-pattern lists which contain group names, regex patterns, and type conversion functions, whereas in **tidyr** these three related pieces of information must be defined in seperate arguments. Therefore **nc** syntax may be preferable in order to ease the definition of complex patterns and to avoid repetition in user code.

In **nc**, there are two different functions for wide-to-tall data reshaping: `nc::capture_melt_single` computes a single output reshape column, and `nc::capture_melt_multiple` computes multiple output reshape columns. In contrast, other functions that support multiple output reshape columns also support a single output reshape column (Table 1). It is natural to ask whether these two **nc** functions could be combined into a single function that could handle both kinds of output. Of course, it is possible, but we prefer to keep the two functions separate in order to provide more specific/informative documentation, examples, and error messages.

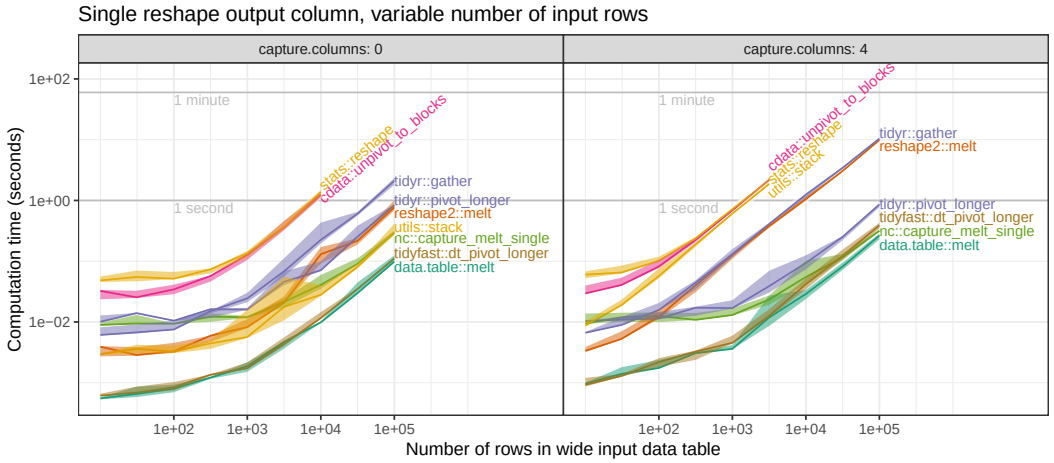


Figure 4: Timings for computing a tall output table with a single reshape column from a wide input table with 56 reshape columns and a variable number of rows (x-axis). The **Left** panel shows time to compute output data table with no capture columns; The **Right** panel shows time to compute output data table with four capture columns (typically slower as post-processing steps may be necessary).

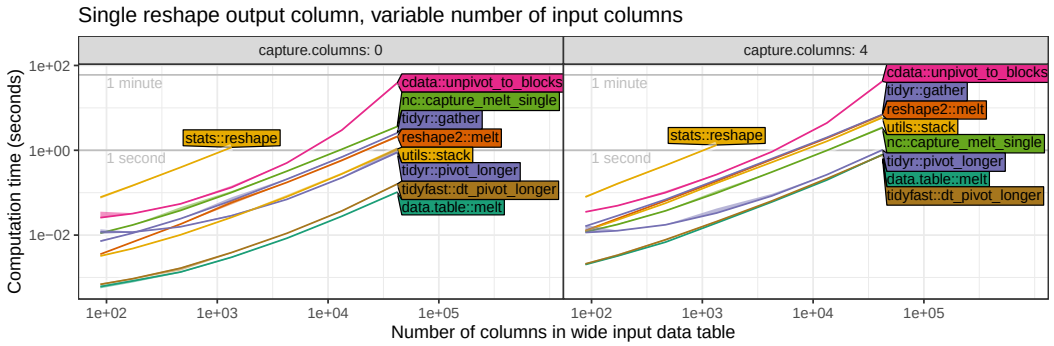


Figure 5: Timings for computing a tall output table with a single reshape column from a wide input table with 11 rows and a variable number of columns to reshape (x-axis). The **Left** panel shows time to compute output data table with no capture columns; The **Right** panel shows time to compute output data table with four capture columns (typically slower as post-processing steps may be necessary).

We have shown how the **nc** package provides a powerful and efficient new syntax for wide-to-tall data reshaping using regular expressions. The inverse operation, tall-to-wide data reshaping, is not supported. For tall-to-wide reshaping operations, we recommend using the efficient implementation in `data.table::dcast`.

Future work

For future work, we will be interested to explore other operations and R packages/functions which could be simplified using regular expressions. For example, the `tidyr::pivot_longer` function requires some repetition of the pattern (in `names_pattern` and `cols` arguments); it could be simplified by changing the behavior when `names_pattern` is specified, and `cols` is not (currently an error, could instead set `cols` to the set of columns which match `names_pattern`).

Another example where there is room for improvement is `data.table::melt`, which we have shown requires some post-processing steps to output capture columns. As a result of this research, we have proposed changes to `data.table::melt`[1] that allow efficient specification and output of capture columns. Since **nc** uses **data.table** internally, we plan to eventually use these changes for speedups of **nc** functions.

Reproducible research statement. The source code for this article can be freely downloaded from https://github.com/tdhock/nc-article

[1] https://github.com/Rdatatable/data.table/pull/4731

Bibliography

T. Barrett. *tidyfast: Fast Tidying of Data*, 2020. URL https://CRAN.R-project.org/package=tidyfast. R package version 0.2.1. [p72]

G. Csárdi. *rematch2: Tidy Output from Regular Expression Matching*, 2017. URL https://CRAN.R-project.org/package=rematch2. R package version 2.0.1. [p70]

M. Dowle and A. Srinivasan. *data.table: Extension of 'data.frame'*, 2019. http://r-datatable.com. [p72]

J. E. F. Friedl. *Mastering Regular Expressions*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2 edition, 2002. [p69]

M. Gagolewski. *R package stringi: Character string processing facilities*, 2018. URL http://www.gagolewski.com/software/stringi/. [p70]

T. D. Hocking. Comparing namedcapture with other r packages for regular expressions. *R Journal*, 2019a. [p69, 70, 72]

T. D. Hocking. *namedCapture: Named Capture Regular Expressions*, 2019b. R package version 2019.01.14. [p70]

T.-Y. Huang and B. Zhao. tidyfst: Tidy verbs for fast data manipulation. *Journal of Open Source Software*, 5(52):2388, 2020. doi: 10.21105/joss.02388. URL https://doi.org/10.21105/joss.02388. [p72]

J. Mount and N. Zumel. *cdata: Fluid Data Transformations*, 2019. URL https://CRAN.R-project.org/package=cdata. R package version 1.1.2. [p72]

K. Ushey, J. Hester, and R. Krzyzanowski. *rex: Friendly Regular Expressions*, 2017. URL https://CRAN.R-project.org/package=rex. R package version 1.1.2. [p70]

Q. Wenfeng. *re2r: RE2 Regular Expression*, 2017. URL https://CRAN.R-project.org/package=re2r. R package version 0.2.0. [p70]

H. Wickham. Reshaping data with the reshape package. *Journal of Statistical Software*, 21(12):1–20, 2007. URL http://www.jstatsoft.org/v21/i12/. [p72]

H. Wickham. *stringr: Simple, Consistent Wrappers for Common String Operations*, 2018. URL https://CRAN.R-project.org/package=stringr. R package version 1.3.1. [p70]

H. Wickham and L. Henry. *tidyr: Easily Tidy Data with 'spread()' and 'gather()' Functions*, 2018. URL https://CRAN.R-project.org/package=tidyr. R package version 0.8.2. [p70]

Toby Dylan Hocking
School of Informatics, Computing, and Cyber Systems
Northern Arizona University
Flagstaff, Arizona
USA
toby.hocking@nau.edu
ORCID 0000-0002-3146-0865

Linear Regression with Stationary Errors: the R Package slm

by Emmanuel Caron, Jérôme Dedecker and Bertrand Michel

Abstract This paper introduces the R package **slm**, which stands for Stationary Linear Models. The package contains a set of statistical procedures for linear regression in the general context where the error process is strictly stationary with a short memory. We work in the setting of Hannan (1973), who proved the asymptotic normality of the (normalized) least squares estimators (LSE) under very mild conditions on the error process. We propose different ways to estimate the asymptotic covariance matrix of the LSE and then to correct the type I error rates of the usual tests on the parameters (as well as confidence intervals). The procedures are evaluated through different sets of simulations.

Introduction

We consider the usual linear regression model

$$Y = X\beta + \varepsilon\,,$$

where Y is the n-dimensional vector of observations, X is a (possibly random) $n \times p$ design matrix, β is a p-dimensional vector of parameters, and $\varepsilon = (\varepsilon_i)_{1\leq i\leq n}$ is the error process (with zero mean and independent of X). The standard assumptions are that the ε_i's are independent and identically distributed (i.i.d.) with zero mean and finite variance.

In this paper, we propose to modify the standard statistical procedures (tests, confidence intervals, ...) of the linear model in the more general context where the ε_i's are obtained from a strictly stationary process $(\varepsilon_i)_{i\in\mathbb{N}}$ with a short memory. To be more precise, let $\hat{\beta}$ denote the usual least squares estimator of β. Our approach is based on two papers: the paper by Hannan (1973) who proved the asymptotic normality of the least squares estimator $D(n)(\hat{\beta}-\beta)$ ($D(n)$ being the usual normalization) under very mild conditions on the design and on the error process; and a recent paper by Caron (2019) who showed that, under Hannan's conditions, the asymptotic covariance matrix of $D(n)(\hat{\beta}-\beta)$ can be consistently estimated.

Let us emphasize that Hannan's conditions on the error process are very mild and are satisfied for most of the short-memory processes (see the discussion in Section 4.4 of Caron and Dede (2018)). Putting together the two above results, we can develop a general methodology for tests and confidence regions on the parameter β, which should be valid for most of the short-memory processes. This is, of course, directly useful for time-series regression, but also in the more general context where the residuals of the linear model seem to be strongly correlated. More precisely, when checking the residuals of the linear model, if the autocorrelation function of the residuals shows significant correlations, and if the residuals can be suitably modeled by an ARMA process, then our methodology is likely to apply. We shall give an example of such a situation on the "Shanghai pollution" dataset at the end of the paper.

Hence, the tools presented in the present paper can be seen from two different points of view:

- as appropriate tools for time series regression with a short memory error process
- as a way to robustify the usual statistical procedures when the residuals are correlated.

Let us now describe the organization of the paper. In the next section, we recall the mathematical background, the consistent estimator of the asymptotic covariance matrix introduced in Caron (2019), and the modified Z-statistics and χ-square statistics for testing the hypothesis on the parameter β. Next, we present the **slm** package and the different ways to estimate the asymptotic covariance matrix: by fitting an autoregressive process on the residuals (default procedure), by means of the kernel estimator described in Caron (2019) (theoretically valid) with a bootstrap method to choose the bandwidth (Wu and Pourahmadi (2009)), by using alternative choices of the bandwidth for the rectangular kernel (Efromovich (1998)) and the quadratic spectral kernel (Andrews (1991)), and by means of an adaptive estimator of the spectral density via Histograms (Comte (2001)). In a section about numerical experiments, we estimate the level of a χ-square test for a linear model with random design, with different kinds of error processes, and for different estimation procedures. In the last section, we apply the package to the "Shanghai pollution" dataset, and we compare the summary output of `slm` with the usual summary output of `lm`. An extended version of this paper is available as an arXiv preprint (see Caron et al. (2019)).

Linear regression with stationary errors

Asymptotic results for the kernel estimator

We start this section by giving a short presentation of linear regression with stationary errors, more details can be found for instance in Caron (2019). Let $\hat{\beta}$ be the usual least squares estimator for the unknown vector β. The aim is to provide hypothesis tests and confidence regions for β in the non i.i.d. context.

Let γ be the autocovariance function of the error process ε: for any integers k and m, let $\gamma(k) = \mathrm{Cov}(\varepsilon_m, \varepsilon_{m+k})$. We also introduce the covariance matrix:

$$\Gamma_n := [\gamma(j-l)]_{1\leq j,l\leq n}\,.$$

Hannan (1973) has shown a Central Limit Theorem for $\hat{\beta}$ when the error process is strictly stationary, under very mild conditions on the design and the error process. Let us notice that the design can be random or deterministic. We introduce the normalization matrix $D(n)$ which is a diagonal matrix with diagonal term $d_j(n) = \left\|X_{.,j}\right\|_2$ for j in $\{1,\ldots,p\}$, where $X_{.,j}$ is the jth column of X. Roughly speaking Hannan's result says in particular that, given the design X, the vector $D(n)(\hat{\beta}-\beta)$ converges in distribution to a centered Gaussian distribution with covariance matrix C. As usual, in practice, the covariance matrix C is unknown, and it has to be estimated. Hannan also showed the convergence of second order moment:[1]

$$\mathbb{E}\left(D(n)(\hat{\beta}-\beta)(\hat{\beta}-\beta)^t D(n)^t \middle| X\right) \xrightarrow[n\to\infty]{} C, \quad a.s.$$

where

$$\mathbb{E}\left(D(n)(\hat{\beta}-\beta)(\hat{\beta}-\beta)^t D(n)^t \middle| X\right) = D(n)(X^tX)^{-1}X^t\Gamma_n X(X^tX)^{-1}D(n).$$

In this paper, we propose a general plug-in approach: for some given estimator $\widehat{\Gamma}_n$ of Γ_n, we introduce the plug-in estimator:

$$\widehat{C} = \widehat{C}(\widehat{\Gamma}_n) := D(n)(X^tX)^{-1}X^t\widehat{\Gamma}_n X(X^tX)^{-1}D(n),$$

and we use $\widehat{C}$ to standardize the usual statistics considered for the study of linear regression.

Let us illustrate this plug-in approach with a kernel estimator which has been proposed in Caron (2019). For some K and a bandwidth h, the kernel estimator $\widetilde{\Gamma}_{n,h}$ is defined by

$$\widetilde{\Gamma}_{n,h} = \left[K\left(\frac{j-l}{h}\right)\tilde{\gamma}_{j-l}\right]_{1\leq j,l\leq n}, \tag{1}$$

where the residual-based empirical covariance coefficients are defined for $0 \leq |k| \leq n-1$ by

$$\tilde{\gamma}_k = \frac{1}{n}\sum_{j=1}^{n-|k|}\hat{\varepsilon}_j\hat{\varepsilon}_{j+|k|}. \tag{2}$$

For a well-chosen kernel K and under mild assumptions on the design and the error process, it has been proved in Caron (2019) that

$$\widetilde{C}_n^{-1/2}D(n)(\hat{\beta}-\beta) \xrightarrow[n\to\infty]{\mathcal{L}} \mathcal{N}_p(0_p, I_p), \tag{3}$$

for the plug-in estimator $\widetilde{C}_n := \widehat{C}(\widetilde{\Gamma}_{n,h_n})$, for some suitable sequence of bandwidths (h_n).

More generally, in this paper, we say that an estimator $\widehat{\Gamma}_n$ of Γ_n is *consistent for estimating the covariance matrix* C if $\widehat{C}(\widehat{\Gamma}_n)$ is positive definite and if it converges in probability to C. Note that such a property requires assumptions on the design, see Caron (2019). If $\widehat{C}(\widehat{\Gamma}_n)$ is consistent for estimating the covariance matrix C, then $\widehat{C}(\widehat{\Gamma}_n)^{-1/2}D(n)(\hat{\beta}-\beta)$ converges in distribution to a standard Gaussian vector.

To conclude this section, let us make some additional remarks. The interest of Caron's recent paper is that the consistency of the estimator $\widehat{C}(\widehat{\Gamma}_n)$ is proved under Hannan's condition on the error process, which is known to be optimal with respect to the convergence in distribution (see for instance Dedecker (2015)), and which allows dealing with most short memory processes. However, the natural estimator of the covariance matrix of $\hat{\beta}$ based on $\widehat{\Gamma}_n$ has been studied by many other

[1] The transpose of a matrix X is denoted by X^t.

authors in various contexts. For instance, let us mention the important line of research initiated by Newey and West (1987, 1994) and the related papers by Andrews (1991), Andrews and Monahan (1992), among others. In the paper by Andrews (1991), the consistency of the estimator based on $\widehat{\Gamma}_n$ is proved under general conditions on the fourth-order cumulants of the error process, and a data-driven choice of the bandwidth is proposed. Note that these authors also considered the case of heteroskedastic processes. Most of these procedures, known as HAC (Heteroskedasticity and Autocorrelation Consistent) procedures, are implemented in the package **sandwich** by Zeileis, Lumley, Berger and Graham, and presented in great detail in the paper by Zeileis (2004). We shall use an argument of the **sandwich** package, based on the data-driven procedure described by Andrews (1991).

Tests and confidence regions

We now present tests and confidence regions for arbitrary estimators $\widehat{\Gamma}_n$. The complete justifications are available for kernel estimators, see Caron (2019).

Z-Statistics. We introduce the following univariate statistics:

$$Z_j = \frac{d_j(n)\hat{\beta}_j}{\sqrt{\widehat{C}_{(j,j)}}}, \tag{4}$$

where $\widehat{C} = \widehat{C}(\widehat{\Gamma}_n)$. If $\widehat{\Gamma}_n$ is consistent for estimating the covariance matrix C and if $\beta_j = 0$, the distribution of Z_j converges to a standard normal distribution when n tends to infinity. We directly derive an asymptotic hypothesis test for testing $\beta_j = 0$ against $\beta_j \neq 0$ as well as an asymptotic confidence interval for β_j.

Chi-square statistics. Let A be an $n \times k$ matrix with $\text{rank}(A) = k$. Under Hannan (1973)'s conditions, $D(n)(A\hat{\beta} - A\beta)$ converges in distribution to a centered Gaussian distribution with covariance matrix ACA^t. If $\widehat{\Gamma}_n$ is consistent for estimating the covariance matrix C, then $A\widehat{C}(\widehat{\Gamma}_n)$ converges in probability to AC. The matrix $A\widehat{C}(\widehat{\Gamma}_n)A^t$ being symmetric positive definite, this yields

$$W := (A\widehat{C}(\widehat{\Gamma}_n))^{-1/2} D(n)A(\hat{\beta} - \beta) \xrightarrow[n\to\infty]{\mathcal{L}} \mathcal{N}_k(0_k, I_k).$$

This last result provides asymptotical confidence regions for the vector $A\beta$. It also provides an asymptotic test for testing the hypothesis $H_0 : A\beta = 0$ against $H_1 : A\beta \neq 0$. Indeed, under the H_0-hypothesis, the distribution of $\|W\|_2^2$ converges to a $\chi^2(k)$-distribution.

The test can be used to simplify a linear model by testing that several linear combinations between the parameters β_j are zero, as we usually do for Anova and regression models. In particular, with $A = I_p$, the test corresponds to the test of overall significance.

Introduction to linear regression with the slm package

Using the **slm** package is very intuitive because the arguments and the outputs of `slm` are similar to those of the standard functions `lm`, `glm`, etc. The output of the main function `slm` is an object of class "slm", a specific class that has been defined for linear regression with stationary processes. The "slm" class has methods `plot`, `summary`, `confint`, and `predict`, see the extended version Caron et al. (2019) for more details. Moreover, the class "slm" inherits from the "lm" class and thus provides the output of the classical `lm` function.

The statistical tools available in `slm` strongly depend on the choice of the covariance plug-in estimator $\widehat{C}(\widehat{\Gamma}_n)$ we use for estimating C. All the estimators $\widehat{\Gamma}_n$ proposed in `slm` are residual-based estimators, but they rely on different approaches. In this section, we present the main functionality of `slm` together with the different covariance plug-in estimators.

For illustrating the package, we simulate synthetic data according to the linear model:

$$Y_i = \beta_1 + \beta_2(\log(i) + \sin(i) + Z_i) + \beta_3 i + \varepsilon_i,$$

where Z is a Gaussian autoregressive process of order 1 and ε is the Nonmixing process described further in the paper. We use the functions `generative_model` and `generative_process` respectively to simulate observations according to this regression design and with this specific stationary process.

```
R> library(slm)
R> set.seed(42)
R> n = 500
R> eps = generative_process(n,"Nonmixing")
R> design = generative_model(n,"mod2")
R> design_sim = cbind(rep(1,n), as.matrix(design))
R> beta_vec = c(2,0.001,0.5)
R> Y = design_sim %*% beta_vec + eps
```

Linear regression via AR fitting on the residuals

A large class of stationary processes with continuous spectral density can be well approximated by AR processes, see for instance Corollary 4.4.2 in Brockwell and Davis (1991). The covariance structure of an AR process having a closed form, it is thus easy to derive an approximation $\widetilde{\Gamma}_{AR(p)}$ of Γ_n by fitting an AR process on the residual process. The AR-based method for estimating C is the default version of `slm`. This method proceeds in four main steps:

1. Fit an autoregressive process on the residual process $\hat{\varepsilon}$;
2. Compute the theoretical covariances of the fitted AR process ;
3. Plug the covariances in the Toeplitz matrix $\widetilde{\Gamma}_{AR(p)}$;
4. Compute $\widehat{C} = \widehat{C}(\widetilde{\Gamma}_{AR(p)})$.

The `slm` function fits a linear regression of the vector Y on the design X and then fits an AR process on the residual process using the `ar` function from the **stats** package. The output of the `slm` function is an object of class "slm". The order p of the AR process is set in the argument `model_selec`:

```
R> regslm = slm(Y ~ X1 + X2, data = design, method_cov_st = "fitAR",
+          model_selec = 3)
```

The estimated covariance is recorded as a vector in the attribute `cov_st` of `regslm`, which is an object of class "slm". The estimated covariance matrix can be computed by taking the Toeplitz matrix of `cov_st`, using the `toeplitz` function.

AR order selection. The order p of the AR process can be chosen at hand by setting `model_selec = p`, or automatically with the AIC criterion by setting `model_selec = -1`.

```
R> regslm = slm(Y ~ X1 + X2, data = design, method_cov_st = "fitAR",
+          model_selec = -1)
```

The order of the fitted AR process is recorded in the `model_selec` attribute of `regslm`:

```
R> regslm@model_selec

[1] 2
```

Here, the AIC criterion suggests to fit an AR(2) process on the residuals.

Linear regression via kernel estimation of the error covariance

The second method for estimating the covariance matrix C is the kernel estimation method (1) studied in Caron (2019). In short, this method estimates C via a smooth approximation of the covariance matrix Γ_n of the residuals. This estimation of Γ_n corresponds to the so-called tapered covariance matrix estimator in the literature, see for instance Xiao and Wu (2012), or also to the "lag-window estimator" defined in Brockwell and Davis (1991), page 330. It applies in particular for non-negative symmetric kernels with compact support, with an integrable Fourier transform and such that $K(0) = 1$. Table 1 gives the list of the available kernels in the package **slm**.

It is also possible for the user to define his own kernel and use it in the argument `kernel_fonc` of the `slm` function. Below we use the triangle kernel, which assures that the covariance matrix is positive definite. The support of the kernel K in Equation (1) being compact, only the terms $\tilde{\gamma}_{j-l}$ for small enough lag $j-l$ are kept and weighted by the kernel in the expression of $\widetilde{\Gamma}_{n,h}$. Rather than setting the bandwidth h, we select the number of $\gamma(k)$'s that should be kept (the lag) with the argument `model_selec` in the `slm` function. Then the bandwidth h is calibrated accordingly, that is equal to `model_selec` $+1$.

kernel_fonc =	kernel definition
rectangular	$K(x) = \mathbb{1}_{\{\lvert x\rvert \leq 1\}}$
triangle	$K(x) = (1-\lvert x\rvert)\mathbb{1}_{\{\lvert x\rvert \leq 1\}}$
trapeze	$K(x) = \mathbb{1}_{\{\lvert x\rvert \leq \delta\}} + \frac{1}{1-\delta}(1-\lvert x\rvert)\mathbb{1}_{\{\delta \leq \lvert x\rvert \leq 1\}}$

Table 1: Available kernel functions in **slm**.

```
R> regslm = slm(Y ~ X1 + X2, data = design, method_cov_st = "kernel",
+          model_selec = 5, kernel_fonc = triangle, plot = TRUE)
```

The plot output by the `slm` function is given in Figure 1.

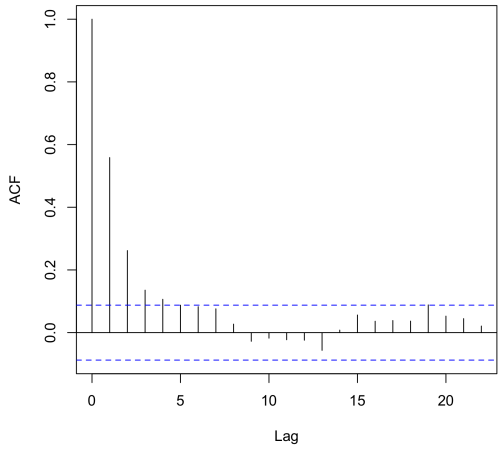


Figure 1: ACF of the residual process.

Order selection via bootstrap. The order parameter can be chosen at hand as before or automatically by setting `model_selec = -1`. The automatic order selection is based on the bootstrap procedure proposed by Wu and Pourahmadi (2009) for banded covariance matrix estimation. The `block_size` argument sets the size of bootstrap blocks, and the `block_n` argument sets the number of blocks. The final order is chosen by taking the order which has the minimal risk. Figure 2 gives the plots of the estimated risk for the estimation of Γ_n (left) and the final estimated ACF (right).

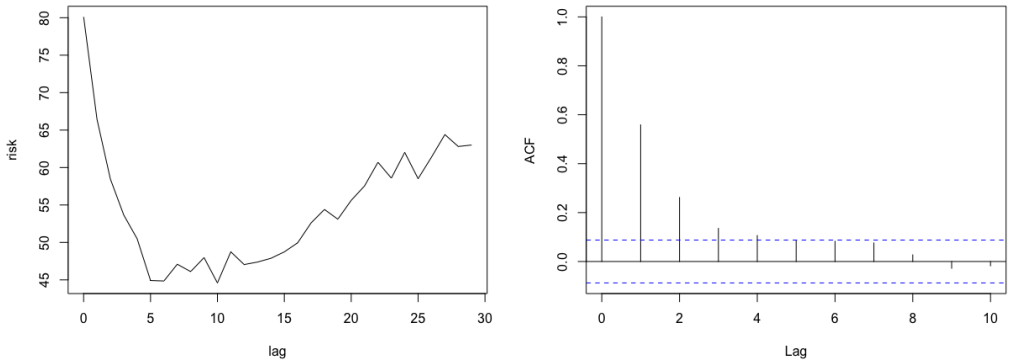
```
R> regslm = slm(Y ~ X1 + X2, data = design, method_cov_st ="kernel",
+          model_selec = -1, kernel_fonc = triangle, model_max = 30,
+          block_size = 100, block_n = 100, plot = TRUE)
```

The selected order is recorded in the `model_selec` attribute of the `slm` object output by the `slm` function:

```
R> regslm@model_selec

[1] 10
```

Order selection by Efromovich's method (rectangular kernel). An alternative method for choosing the bandwidth in the case of the rectangular kernel has been proposed in Efromovich (1998). For a large class of stationary processes with exponentially decaying autocovariance function (mainly the ARMA processes), Efromovich proved that the rectangular kernel is asymptotically minimax, and he proposed the following estimator:

$$\hat{f}_{J_{nr}}(\lambda) = \frac{1}{2\pi} \sum_{k=-J_{nr}}^{k=J_{nr}} \hat{\gamma}_k e^{ik\lambda},$$



(a) Estimated risk error via bootstrap. **(b)** Estimated ACF for the selected order.

Figure 2: Plots output by `slm` for the kernel method with bootstrap selection of the order.

with the lag

$$J_{nr} = \frac{\log(n)}{2r}\left[1 + (\log(n))^{-1/2}\right],$$

where r is a regularity index of the autocovariance index. In practice, this parameter is unknown and is estimated thanks to the algorithm proposed in the section 4 of Efromovich (1998). As for the other methods, we use the residual based empirical covariances $\tilde{\gamma}_k$ to compute $\hat{f}_{J_{nr}}(\lambda)$.

```
R> regslm = slm(Y ~ X1 + X2, data = design, method_cov_st = "efromovich",
+          model_selec = -1)
```

Order Selection by Andrews's method. Another method for choosing the bandwidth has been proposed by Andrews (1991) and implemented in the package **sandwich** by Zeileis, Lumley, Berger and Graham (see the paper by Zeileis (2004)). For the **slm** package, the automatic choice of the bandwidth proposed by Andrews can be obtained as follows:

```
R> regslm = slm(Y ~ X1 + X2, data = design, method_cov_st = "hac")
```

The procedure is based on the function `kernHAC` in the **sandwich** package. This function computes directly the covariance matrix estimator of $\hat{\beta}$, which will be recorded in the slot `Cov_ST` of the `slm` function. Here, we take the quadratic spectral kernel:

$$K(x) = \frac{25}{12\pi^2 x^2}\left(\frac{\sin(6\pi x/5)}{6\pi x/5} - \cos(6\pi x/5)\right),$$

as suggested by Andrews (see Section 2 in Andrews (1991), or Section 3.2 in Zeileis (2004)), but other kernels could be used, such as Bartlett, Parzen, Tukey-Hamming, among others (see Zeileis (2004)).

Positive definite projection. Depending on the method used, the matrix $\widehat{C}(\widehat{\Gamma}_n)$ may not always be positive definite. It is the case of the kernel method with rectangular or trapeze kernel. To overcome this problem, we make the projection of $\widehat{C}(\widehat{\Gamma}_n)$ into the cone of positive definite matrices by applying a hard thresholding on the spectrum of this matrix: we replace all eigenvalues lower or equal to zero with the smallest positive eigenvalue of $\widehat{C}(\widehat{\Gamma}_n)$. Note that this projection is useless for the triangle or quadratic spectral kernels because their Fourier transform is non-negative (leading to a positive definite matrix $\widehat{C}(\widehat{\Gamma}_n)$). Of course, it is also useless for the `fitAR` and `spectralproj` methods.

Linear regression via projection spectral estimation

The projection method relies on the ideas of Comte (2001), where an adaptive nonparametric method has been proposed for estimating the spectral density of a stationary Gaussian process. We use the residual process as a proxy for the error process, and we compute the projection coefficients with the residual-based empirical covariance coefficients $\tilde{\gamma}_k$, see Equation (2). For some $d \in \mathbb{N}^*$,

the estimator of the spectral density of the error process that we use is defined by computing the projection estimators for the residual process on the basis of histogram functions:

$$\phi_j^{(d)} = \sqrt{\frac{d}{\pi}} \mathbb{1}_{[\pi j/d, \pi(j+1)/d[}, \qquad j = 0, 1, \ldots, d-1.$$

The estimator is defined by

$$\hat{f}_d(\lambda) = \sum_{j=0}^{d-1} \hat{a}_j^{(d)} \phi_j^{(d)},$$

where the projection coefficients are

$$\hat{a}_j^{(d)} = \sqrt{\frac{d}{\pi}} \left(\frac{\tilde{\gamma}_0}{2d} + \frac{1}{\pi} \sum_{r=1}^{n-1} \frac{\tilde{\gamma}_r}{r} \left[\sin\left(\frac{\pi(j+1)r}{d} \right) - \sin\left(\frac{\pi j r}{d} \right) \right] \right).$$

The Fourier coefficients of the spectral density are equal to the covariance coefficients. Thus, for $k = 1, \ldots, n-1$ it yields

$$\begin{aligned} \gamma_k &= c_k \\ &= \frac{2}{k} \sqrt{\frac{d}{\pi}} \sum_{j=0}^{d-1} \hat{a}_j^{(d)} \left[\sin\left(\frac{k\pi(j+1)}{d} \right) - \sin\left(\frac{k\pi j}{d} \right) \right], \end{aligned}$$

and for $k = 0$:

$$\gamma_0 = c_0 = 2\sqrt{\frac{\pi}{d}} \sum_{j=0}^{d-1} \hat{a}_j^{(d)}.$$

This method can be proceeded in the `slm` function by setting `method_cov_st = "spectralproj"`:

```
R> regslm = slm(Y ~ X1 + X2, data = design, method_cov_st = "spectralproj",
+          model_selec = 10, plot = TRUE)
```

The graph of the estimated spectral density can be plotted by setting `plot = TRUE` in the `slm` function, see Figure 3.

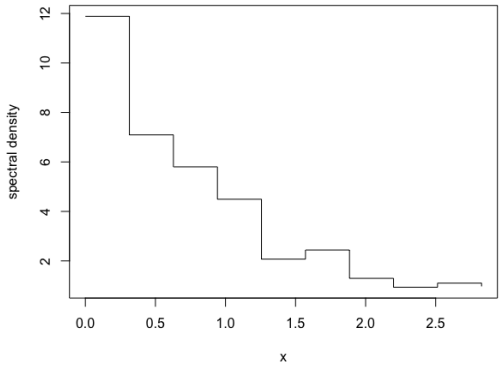


Figure 3: Spectral density estimator by projection on the histogram basis.

Model selection. The Gaussian model selection method proposed in Comte (2001) follows the ideas of Birgé and Massart, see for instance Massart (2007). It consists of minimizing the l_2 penalized criterion, see Section 5 in Comte (2001):

$$\text{crit}(d) := -\sum_{j=0}^{d-1} \left[\hat{a}_j^{(d)} \right]^2 + c\frac{d}{n},$$

where c is a multiplicative constant that in practice can be calibrated using the slope heuristic method, see Birgé and Massart (2007), Baudry et al. (2012) and the R package **capushe**.

```
R> regslm = slm(Y ~ X1 + X2, data = design, method_cov_st = "spectralproj",
+          model_selec = -1, model_max = 50, plot = TRUE)
```

The selected dimension is recorded in the `model_selec` attribute of the `slm` object output by the `slm` function:

```
R> regslm@model_selec
```

```
[1] 8
```

The slope heuristic algorithm here selects a Histogram on a regular partition of size 8 over the interval $[0, \pi]$ to estimate the spectral density.

Linear regression via masked covariance estimation

This method is a full-manual method for estimating the covariance matrix C by only selecting covariance terms from the residual covariances $\tilde{\gamma}_k$ defined by (2). Let I be a set of positive integers, then we consider

$$\hat{\gamma}_I(k) := \tilde{\gamma}_k \mathbb{1}_{k \in I \cup \{0\}}, \qquad 0 \leq |k| \leq n-1,$$

and then we define the estimated covariance marix $\widehat{\Gamma}_I$ by taking the Toeplitz matrix of the vector $\hat{\gamma}_I$. This estimator is a particular example of a masked sample covariance estimator, as introduced by Chen et al. (2012), see also Levina and Vershynin (2012). Finally, we derive from $\widehat{\Gamma}_I$ an estimator $\widehat{C}(\widehat{\Gamma}_I)$ for C.

The next instruction selects the coefficients 0, 1, 2 and 4 from the residual covariance terms:

```
R> regslm = slm(Y ~ X1 + X2, data = design, method_cov_st = "select",
+          model_selec = c(1,2,4))
```

The positive lags of the selected covariances are recorded in the `model_selec` argument. Let us notice that the variance γ_0 is automatically selected.

As for the kernel method, the resulting covariance matrix may not be positive definite. If it is the case, the positive definite projection method described before is used.

Linear regression via manual plugged covariance matrix

This last method is a direct plug-in method. The user proposes his own vector estimator $\hat{\gamma}$ of γ, and then the Toeplitz matrix $\widehat{\Gamma}_n$ of the vector $\hat{\gamma}$ is used for estimating C with $\widehat{C}(\widehat{\Gamma}_n)$.

```
R> v = rep(0,n)
R> v[1:10] = acf(eps, type = "covariance", lag.max = 9)$acf
R> regslm = slm(Y ~ X1 + X2, data = design, cov_st = v)
```

The user can also propose his own covariance matrix $\widehat{\Gamma}_n$ for estimating C.

```
R> v = rep(0,n)
R> v[1:10] = acf(eps, type = "covariance", lag.max = 9)$acf
R> V = toeplitz(v)
R> regslm = slm(Y ~ X1 + X2, data = design, Cov_ST = V)
```

Let us notice that the user must verify that the resulting covariance matrix is positive definite. The positive definite projection algorithm is not used with this method.

Numerical experiments and method comparisons

This section summarizes an extensive study which has been carried out to compare the performances of the different approaches presented before in the context of a linear model with short range dependent stationary errors.

Description of the generative models

We first present the five generative models for the errors that we consider in the paper. We choose different kinds of processes to reflect the diversity of short-memory processes.

- **AR1 process.** The AR1 process is a Gaussian AR(1) process defined by

$$\varepsilon_i - 0.7\varepsilon_{i-1} = W_i,$$

where W_i is a standard gaussian distribution $\mathcal{N}(0,1)$.

- **AR12 process.** The AR12 process is a seasonal AR(12) process defined by

$$\varepsilon_i - 0.5\varepsilon_{i-1} - 0.2\varepsilon_{i-12} = W_i,$$

where W_i is a standard Gaussian distribution $\mathcal{N}(0,1)$. When studying monthly datasets, one usually observes a seasonality of order 12. For example, when looking at climate data, the data are often collected per month, and the same phenomenon tends to repeat every year. Even if the design integrates the deterministic part of the seasonality, a correlation of order 12 usually remains present in the residual process.

- **MA12 process.** The MA12 is also a seasonal process defined by

$$\varepsilon_i = W_i + 0.5W_{i-2} + 0.3W_{i-3} + 0.2W_{i-12},$$

where the (W_i)'s are i.i.d. random variables following Student's distribution with 10 degrees of freedom.

- **Nonmixing process.** The three processes described above are basic ARMA processes, whose innovations have absolutely continuous distributions; in particular, they are strongly mixing in the sense of Rosenblatt (1956), with a geometric decay of the mixing coefficients (in fact, the MA12 process is even 12-dependent, which means that the mixing coefficient $\alpha(k) = 0$ if $k > 12$). Let us now describe a more complicated process: let $(Z_1, \ldots, Z_n)$ satisfying the AR(1) equation

$$Z_{i+1} = \frac{1}{2}(Z_i + \eta_{i+1}),$$

where Z_1 is uniformly distributed over $[0,1]$ and the η_i's are i.i.d. random variables with distribution $\mathcal{B}(1/2)$, independent of Z_1. The process $(Z_i)_{i\geq 1}$ is a strictly stationary Markov chain, but it is not α-mixing in the sense of Rosenblatt (see Bradley (1986)). Let now Q_{0,σ^2} be the inverse of the cumulative distribution function of a centered Gaussian distribution with variance σ^2 (for the simulations below, we choose $\sigma^2 = 25$). The Nonmixing process is then defined by

$$\varepsilon_i = Q_{0,\sigma^2}(Z_i).$$

The sequence $(\varepsilon_i)_{i\geq 1}$ is also a stationary Markov chain (as an invertible function of a stationary Markov chain). By construction, ε_i is $\mathcal{N}(0,\sigma^2)$-distributed, but the sequence $(\varepsilon_i)_{i\geq 1}$ is not a Gaussian process (otherwise, it would be mixing in the sense of Rosenblatt). Although it is not obvious, one can prove that the process $(\varepsilon_i)_{i\geq 1}$ satisfies Hannan's condition (see Caron (2019), Section 4.2).

- **Sysdyn process.** The four processes described above have the property of "geometric decay of correlations", which means that the $\gamma(k)$'s tend to 0 at an exponential rate. However, as already pointed out in the introduction, Hannan's condition is valid for most of the short memory processes, even for processes with slow decay of correlations (provided that the $\gamma(k)$'s are summable). Hence, our last example will be a non-mixing process (in the sense of Rosenblatt), with an arithmetic decay of the correlations.
For $\gamma \in]0,1[$, the intermittent map $\theta_\gamma : [0,1] \mapsto [0,1]$ introduced in Liverani et al. (1999) is defined by

$$\theta_\gamma(x) = \begin{cases} x(1 + 2^\gamma x^\gamma) & \text{if} \quad x \in [0, 1/2[\\ 2x - 1 & \text{if} \quad x \in [1/2, 1]. \end{cases}$$

It follows from Liverani et al. (1999) that there exists a unique θ_γ-invariant probability measure ν_γ. The Sysdyn process is then defined by

$$\varepsilon_i = \theta_\gamma^i.$$

From Liverani et al. (1999), we know that on the probability space $([0,1], \nu_\gamma)$, the autocorrelations $\gamma(k)$ of the stationary process $(\varepsilon_i)_{i\geq 1}$ are exactly of order $k^{-(1-\gamma)/\gamma}$. Hence, $(\varepsilon_i)_{i\geq 1}$ is a short memory process provided $\gamma \in]0, 1/2[$. Moreover, it has been proved in Section 4.4 of Caron and Dede (2018) that $(\varepsilon_i)_{i\geq 1}$ satisfies Hannan's condition in the whole short-memory range, that is for $\gamma \in]0, 1/2[$. For the simulations below, we took $\gamma = 1/4$, which give autocorrelations $\gamma(k)$ of order k^{-3}.

The linear regression models simulated in the experiments all have the following form:

$$Y_i = \beta_1 + \beta_2(\log(i) + \sin(i) + Z_i) + \beta_3 i + \varepsilon_i, \qquad \text{for all } i \text{ in } \{1, \dots, n\}, \tag{5}$$

where Z is a Gaussian autoregressive process of order 1 and ε is one of the stationary processes defined above. For the simulations, β_1 is always equal to 3. All the error processes presented above can be simulated with the **slm** package with the `generative_process` function. The design can be simulated with the `generative_model` function.

Automatic calibration of the tests

It is, of course, of first importance to provide hypothesis tests with correct significance levels or at least with correct asymptotical significance levels, which is possible if the estimator $\widehat{\Gamma}_n$ of the covariance matrix Γ_n is consistent for estimating C. For instance, the results of Caron (2019) show that it is possible to construct statistical tests with correct asymptotical significance levels. However, in practice, such asymptotical results are not sufficient since they do not indicate how to tune the bandwidth on a given dataset. This situation makes the practice of linear regression with dependent errors really more difficult than linear regression with i.i.d. errors. This problem happens for several methods given before ; order choice for the `fitAR` method, bandwidth choice for the `kernel` method, dimension selection for the `spectralproj` method.

It is a tricky issue to design a data-driven procedure for choosing test parameters in order to have a correct Type I Error. Note that unlike with supervised problems and density estimation, it is not possible to calibrate hypothesis tests in practice using cross-validation approaches. We thus propose to calibrate the tests using well-founded statistical procedures for risk minimization ; AIC criterion for the `fitAR` method, bootstrap procedures for the `kernel` method, and slope heuristics for the `spectralproj` method. These procedures are implemented in the `slm` function with the `model_selec = -1` argument, as detailed in the previous section.

Let us first illustrate the calibration problem with the AR12 process. For $T = 1000$ simulations, we generate an error process of size n under the null hypothesis: $H_0 : \beta_2 = \beta_3 = 0$. Then we use the `fitAR` method of the `slm` function with orders between 1 and 50, and we perform the model significance test. The procedure is repeated 1000 times, and we estimate the true level of the test by taking the average of the estimated levels on the 1000 simulations for each order. The results are given in Figure 4 for $n = 1000$. A boxplot is also displayed to visualize the distribution of the order selected by the automatic criterion (AIC).

Non-Seasonal errors

We first study the case of non-Seasonal error processes. We simulate an n-error process according to the AR1, the Nonmixing, or the Sysdyn processes. We simulate realizations of the linear regression model (5) under the null hypothesis: $H_0 : \beta_2 = \beta_3 = 0$. We use the automatic selection procedures for each method (`model_selec = -1`). The simulations are repeated 1000 times in order to estimate the true level of the model significance for each test procedure. We simulate either small samples ($n = 200$) or larger samples ($n = 1000, 2000, 5000$). The results of these experiments are summarized in Table 2.

For n large enough ($n \geq 1000$), all methods work well, and the estimated level is around 0.05. However, for small samples ($n = 200$), we observe that the `fitAR` and the `hac` methods show better performances than the others. The `kernel` method is slightly less effective. With this method, we must choose the size of the bootstrap blocks as well as the number of blocks, and the test results are really sensitive to these parameters. In these simulations, we have chosen 100 blocks with a size of $n/2$. The results are expected to improve with a larger number of blocks.

Let us notice that for all methods and for all sample sizes, the estimated level is much better than if no correction is made (usual Fisher tests).

Seasonal errors

We now study the case of linear regression with seasonal errors. The experiment is exactly the same as before, except that we simulate AR12 or MA12 processes. The results of these experiments are summarized in Table 3.

We directly see that the case of seasonal processes is more complicated than for the non-seasonal processes especially for the AR12 process. For a small samples size, the estimated level is between 0.17 and 0.24, which is clearly too large. It is, however, much better than the estimated level of the

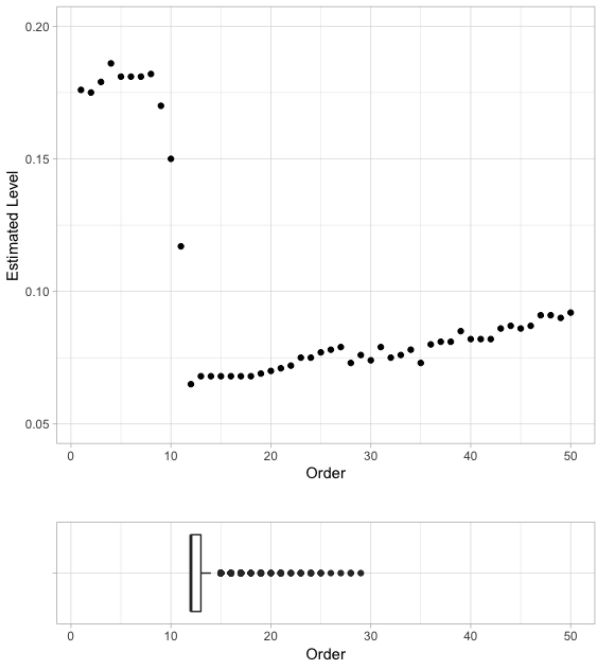


Figure 4: Estimated level of the test according to the order of the fitted AR process on the residuals (top) and boxplot of the order selected by AIC, over 1000 simulations. The data has been simulated according to Model (5) with $\beta_1 = 3$ and $\beta_2 = \beta_3 = 0$, with $n = 1000$.

usual Fisher test, which is around 0.45. The `fitAR` method is the best method here for the AR12 process because for $n \geq 1000$, the estimated level is between 0.06 and 0.07. For `efromovich` and `kernel` methods, a level less than 0.10 is reached but for large samples only. The `spectralproj` and `hac` methods do not seem to work well for the AR12 process, although they remain much better than the usual Fisher tests (around 19% of rejection instead of 45%).

The case of the MA12 process seems easier to deal with. For n large enough ($n \geq 1000$), the estimated level is between 0.04 and 0.07 whatever the method, except for `hac` (around 0.15 for $n = 5000$). It is less effective for a small sample size ($n = 200$) with an estimated level around 0.115 for `fitAR`, `spectralproj` and `efromovich` methods.

I.I.D. errors

To be complete, we consider the case where the ϵ_i's are i.i.d., to see how the five automatic methods perform in that case. We simulate n i.i.d. centered random variables according to the formula:

$$\epsilon_i = W_i^2 - \frac{5}{4},$$

where W follows a student distribution with 10 degrees of freedom. Note that the distribution of the ϵ_i's is not symmetric and has no exponential moments. Except for the `kernel` method, the estimated levels are close to 5% for n large enough ($n \geq 300$). It is slightly worse for small samples, but it remains quite good for the methods `fitAR`, `efromovich`, and `hac`.

As a general conclusion of this section about numerical experiments and method comparison, we see that the `fitAR` method performs quite well in a wide variety of situations and should therefore be used as soon as the user suspects that the error process can be modeled by a stationary short-memory process.

Application to the PM2.5 pollution Shanghai Dataset

This dataset comes from a study about fine particle pollution in five Chinese cities. The data are available on the following website https://archive.ics.uci.edu/ml/datasets/PM2.5+Data+of+Five+Chinese+Cities#. Here we are interested with the city of Shanghai. We study the regression

n	Method / Process	Fisher test	fitAR	spectralproj
200	AR1 process	0.465	**0.097**	0.14
	NonMixing	0.298	0.082	0.103
	Sysdyn process	0.385	**0.105**	0.118
1000	AR1 process	0.418	0.043	**0.049**
	NonMixing	0.298	0.046	**0.05**
	Sysdyn process	0.393	**0.073**	0.077
2000	AR1 process	0.454	0.071	0.078
	NonMixing	0.313	**0.051**	0.053
	Sysdyn process	0.355	**0.063**	0.064
5000	AR1 process	0.439	0.044	**0.047**
	NonMixing	0.315	0.053	0.056
	Sysdyn process	0.381	0.058	0.061

n	Method / Process	efromovich	kernel	hac
200	AR1 process	0.135	0.149	0.108
	NonMixing	0.096	0.125	**0.064**
	Sysdyn process	0.124	0.162	0.12
1000	AR1 process	**0.049**	0.086	**0.049**
	NonMixing	0.053	0.076	0.038
	Sysdyn process	0.079	0.074	0.078
2000	AR1 process	0.075	**0.067**	0.071
	NonMixing	0.057	0.067	0.047
	Sysdyn process	0.066	0.069	0.073
5000	AR1 process	**0.047**	**0.047**	0.044
	NonMixing	0.059	0.068	**0.05**
	Sysdyn process	**0.057**	0.064	0.071

Table 2: Estimated levels for the non-seasonal processes.

of PM2.5 pollution in Xuhui District by other measurements of pollution in neighboring districts and also by meteorological variables. The dataset contains hourly observations between January 2010 and December 2015. More precisely, it contains 52584 records of 17 variables: date, time of measurement, pollution and meteorological variables. More information on these data is available in the paper of Liang et al. (2016).

We remove the lines that contain `NA` observations, and we then extract the first 5000 observations. For simplicity, we will only consider pollution variables and weather variables. We start the study with the following 10 variables:

- `PM_Xuhui`: PM2.5 concentration in the Xuhui district (ug/m^3)
- `PM_Jingan`: PM2.5 concentration in the Jing'an district (ug/m^3)
- `PM_US.Post`: PM2.5 concentration in the U.S diplomatic post (ug/m^3)
- `DEWP`: Dew Point (Celsius Degree)
- `TEMP`: Temperature (Celsius Degree)
- `HUMI`: Humidity (%)
- `PRES`: Pressure (hPa)
- `Iws`: Cumulated wind speed (m/s)
- `precipitation`: hourly precipitation (mm)
- `Iprec`: Cumulated precipitation (mm)

```
R> shan = read.csv("ShanghaiPM20100101_20151231.csv", header = TRUE,
+          sep = ",")
R> shan = na.omit(shan)
R> shan_complete = shan[1:5000,c(7,8,9,10,11,12,13,15,16,17)]
R> shan_complete[1:5,]
```

n	Method / Process	Fisher test	fitAR	spectralproj
200	AR12 process	0.436	0.178	0.203
	MA12 process	0.228	**0.113**	**0.113**
1000	AR12 process	0.468	**0.068**	0.183
	MA12 process	0.209	0.064	0.066
2000	AR12 process	0.507	**0.071**	0.196
	MA12 process	0.237	0.064	0.064
5000	AR12 process	0.47	**0.062**	0.183
	MA12 process	0.242	0.044	**0.048**

n	Method / Process	efromovich	kernel	hac
200	AR12 process	0.223	0.234	**0.169**
	MA12 process	0.116	0.15	0.222
1000	AR12 process	0.181	0.124	0.179
	MA12 process	0.069	**0.063**	0.18
2000	AR12 process	0.153	0.104	0.192
	MA12 process	**0.058**	0.068	0.173
5000	AR12 process	0.1	0.091	0.171
	MA12 process	0.043	0.057	0.147

Table 3: Estimated levels for the seasonal processes.

n	Method / Process	Fisher test	fitAR	spectralproj
150	i.i.d. process	0.053	0.068	0.078
300	i.i.d. process	0.052	0.051	0.06
500	i.i.d. process	0.047	0.049	0.053

n	Method / Process	efromovich	kernel	hac
150	i.i.d. process	0.061	0.124	0.063
300	i.i.d. process	0.05	0.095	0.052
500	i.i.d. process	0.049	0.082	0.056

Table 4: Estimated levels for the i.i.d. process

```
      PM_Jingan PM_US.Post PM_Xuhui DEWP  HUMI PRES TEMP Iws
26305        66         70       71   -5 69.00 1023    0  60
26306        67         76       72   -5 69.00 1023    0  62
26308        73         78       74   -4 74.41 1023    0  65
26309        75         77       77   -4 80.04 1023   -1  68
26310        73         78       80   -4 80.04 1023   -1  70
      precipitation Iprec
26305             0     0
26306             0     0
26308             0     0
26309             0     0
26310             0     0
```

The aim is to study the concentration of particles in Xuhui District according to the other variables. We first fit a linear regression with the `lm` function.

```
R> reglm = lm(shan_complete$PM_Xuhui ~ . ,data = shan_complete)
R> summary.lm(reglm)

Call:
lm(formula = shan_complete$PM_Xuhui ~ ., data = shan_complete)
```

```
Residuals:
     Min       1Q   Median       3Q      Max
-132.139   -4.256   -0.195    4.279  176.450

Coefficients:
               Estimate Std. Error t value Pr(>|t|)
(Intercept)   -54.859483  40.975948  -1.339 0.180690
PM_Jingan       0.596490   0.014024  42.533  < 2e-16 ***
PM_US.Post      0.375636   0.015492  24.246  < 2e-16 ***
DEWP           -1.038941   0.170144  -6.106 1.10e-09 ***
HUMI            0.291713   0.045799   6.369 2.07e-10 ***
PRES            0.025287   0.038915   0.650 0.515852
TEMP            1.305543   0.168754   7.736 1.23e-14 ***
Iws            -0.007650   0.002027  -3.774 0.000163 ***
precipitation   0.462885   0.132139   3.503 0.000464 ***
Iprec          -0.125456   0.039025  -3.215 0.001314 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 10.68 on 4990 degrees of freedom
Multiple R-squared:  0.9409,        Adjusted R-squared:  0.9408
F-statistic:  8828 on 9 and 4990 DF,  p-value: < 2.2e-16
```

The variable `PRES` has no significant effect on the `PM_Xuhui` variable. We then perform a backward selection procedure, which leads to select 9 significant variables:

```
R> shan_lm = shan[1:5000,c(7,8,9,10,11,13,15,16,17)]
R> reglm = lm(shan_lm$PM_Xuhui ~ . ,data = shan_lm)
R> summary.lm(reglm)

Call:
lm(formula = shan_lm$PM_Xuhui ~ ., data = shan_lm)

Residuals:
     Min       1Q   Median       3Q      Max
-132.122   -4.265   -0.168    4.283  176.560

Coefficients:
               Estimate Std. Error t value Pr(>|t|)
(Intercept)   -28.365506   4.077590  -6.956 3.94e-12 ***
PM_Jingan       0.595564   0.013951  42.690  < 2e-16 ***
PM_US.Post      0.376486   0.015436  24.390  < 2e-16 ***
DEWP           -1.029188   0.169471  -6.073 1.35e-09 ***
HUMI            0.285759   0.044870   6.369 2.08e-10 ***
TEMP            1.275880   0.162453   7.854 4.90e-15 ***
Iws            -0.007734   0.002023  -3.824 0.000133 ***
precipitation   0.462137   0.132127   3.498 0.000473 ***
Iprec          -0.127162   0.038934  -3.266 0.001098 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 10.68 on 4991 degrees of freedom
Multiple R-squared:  0.9409,        Adjusted R-squared:  0.9408
F-statistic:  9933 on 8 and 4991 DF,  p-value: < 2.2e-16
```

The autocorrelation of the residual process shows that the errors are clearly not i.i.d., see Figure 5. We thus suspect the `lm` procedure to be unreliable in this context.

The autocorrelation function decreases pretty fast, and the partial autocorrelation function suggests that fitting an AR process on the residuals should be an appropriate method in this case. The automatic `fitAR` method of `slm` selects an AR process of order 28. The residuals of this AR fitting look like white noise, as shown in Figure 6. Consequently, we propose to perform a linear regression with `slm` function, using the `fitAR` method on the complete model.

```
R> regslm = slm(shan_complete$PM_Xuhui ~ . ,data = shan_complete,
+          method_cov_st = "fitAR", model_selec = -1)
R> summary(regslm)
```

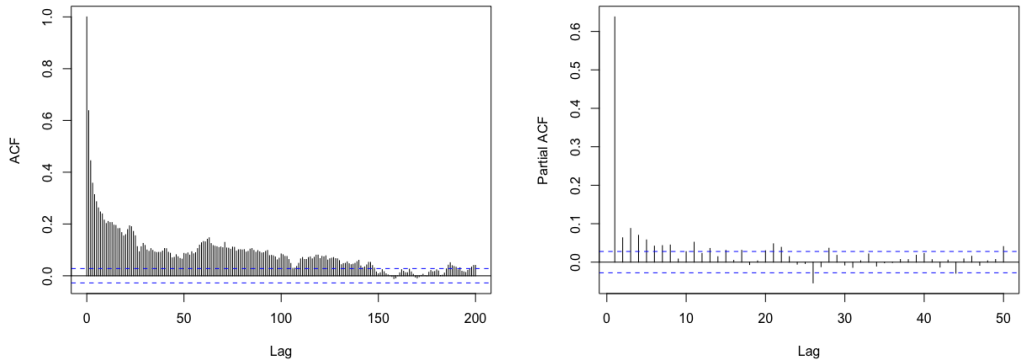


Figure 5: Autocorrelation function (left) and partial autocorrelation function (right) of the residuals.

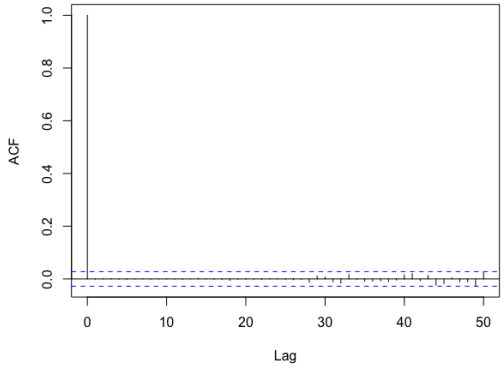


Figure 6: Autocorrelation function of the residuals for the AR fitting.

```
Call:
"slm(formula = myformula, data = data, x = x, y = y)"

Residuals:
     Min       1Q   Median       3Q      Max
-132.139   -4.256   -0.195    4.279  176.450

Coefficients:
                Estimate Std. Error z value Pr(>|z|)
(Intercept)   -54.859483 143.268399  -0.383 0.701783
PM_Jingan       0.596490   0.028467  20.953  < 2e-16 ***
PM_US.Post      0.375636   0.030869  12.169  < 2e-16 ***
DEWP           -1.038941   0.335909  -3.093 0.001982 **
HUMI            0.291713   0.093122   3.133 0.001733 **
PRES            0.025287   0.137533   0.184 0.854123
TEMP            1.305543   0.340999   3.829 0.000129 ***
Iws            -0.007650   0.005698  -1.343 0.179399
precipitation   0.462885   0.125641   3.684 0.000229 ***
Iprec          -0.125456   0.064652  -1.940 0.052323 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 10.68
Multiple R-squared:  0.9409
chi2-statistic:  8383 on 9 DF,  p-value: < 2.2e-16
```

Note that the variables show globally larger p-values than with the `lm` procedure, and more variables have no significant effect than with `lm`. After performing a backward selection, we obtain the following results:

```
R> shan_slm = shan[1:5000,c(7,8,9,10,11,13)]
R> regslm = slm(shan_slm$PM_Xuhui ~ . , data = shan_slm,
+          method_cov_st = "fitAR", model_selec = -1)
R> summary(regslm)

Call:
"slm(formula = myformula, data = data, x = x, y = y)"

Residuals:
     Min       1Q   Median       3Q      Max
-132.263   -4.341   -0.192    4.315  176.501

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept) -29.44924    8.38036  -3.514 0.000441 ***
PM_Jingan     0.60063    0.02911  20.636  < 2e-16 ***
PM_US.Post    0.37552    0.03172  11.840  < 2e-16 ***
DEWP         -1.05252    0.34131  -3.084 0.002044 **
HUMI          0.28890    0.09191   3.143 0.001671 **
TEMP          1.30069    0.32435   4.010 6.07e-05 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 10.71
Multiple R-squared:  0.9406
chi2-statistic:  8247 on 5 DF,  p-value: < 2.2e-16
```

The backward selection with `slm` only keeps 5 variables.

Acknowledgements

The authors are grateful to Anne Philippe (Nantes University) and Aymeric Stamm (CNRS - Nantes University) for valuable discussions.

Bibliography

D. Andrews. Heteroskedasticity and autocorrelation consistent covariant matrix estimation. Econometrica, 59(3):817–858, 1991. [p83, 85, 88]

D. W. Andrews and J. C. Monahan. An improved heteroskedasticity and autocorrelation consistent covariance matrix estimator. Econometrica: Journal of the Econometric Society, pages 953–966, 1992. [p85]

J.-P. Baudry, C. Maugis, and B. Michel. Slope heuristics: overview and implementation. Statistics and Computing, 22(2):455–470, 2012. [p89]

L. Birgé and P. Massart. Minimal penalties for gaussian model selection. Probability theory and related fields, 138(1-2):33–73, 2007. [p89]

R. C. Bradley. Basic properties of strong mixing conditions. In Dependence in probability and statistics (Oberwolfach, 1985), volume 11 of Progr. Probab. Statist., pages 165–192. Birkhäuser Boston, Boston, MA, 1986. [p91]

P. J. Brockwell and R. A. Davis. Time Series: Theory and Methods. Springer Science & Business Media, 1991. [p86]

E. Caron. Asymptotic distribution of least square estimators for linear models with dependent errors. Statistics, 53(4):885–902, 2019. [p83, 84, 85, 86, 91, 92]

E. Caron and S. Dede. Asymptotic distribution of least squares estimators for linear models with dependent errors: Regular designs. Mathematical Methods of Statistics, 27(4):268–293, 2018. [p83, 91]

E. Caron, J. Dedecker, and B. Michel. Linear regression with stationary errors: the R package slm. arXiv preprint arXiv:1906.06583, 2019. [p83, 85]

R. Y. Chen, A. Gittens, and J. A. Tropp. The masked sample covariance estimator: an analysis using matrix concentration inequalities. Information and Inference: A Journal of the IMA, 1(1): 2–20, 2012. [p90]

F. Comte. Adaptive estimation of the spectrum of a stationary gaussian sequence. Bernoulli, 7(2): 267–298, 2001. [p83, 88, 89]

J. Dedecker. On the optimality of McLeish's conditions for the central limit theorem. Comptes Rendus Mathematique, 353(6):557–561, 2015. [p84]

S. Efromovich. Data-driven efficient estimation of the spectral density. Journal of the American Statistical Association, 93(442):762–769, 1998. [p83, 87, 88]

E. J. Hannan. Central limit theorems for time series regression. Zeitschrift für Wahrscheinlichkeitstheorie und verwandte Gebiete, 26(2):157–170, 1973. [p83, 84, 85]

E. Levina and R. Vershynin. Partial estimation of covariance matrices. Probability theory and related fields, 153(3-4):405–419, 2012. [p90]

X. Liang, S. Li, S. Zhang, H. Huang, and S. X. Chen. Pm2.5 data reliability, consistency, and air quality assessment in five chinese cities. Journal of Geophysical Research: Atmospheres, 121(17): 10–220, 2016. [p94]

C. Liverani, B. Saussol, and S. Vaienti. A probabilistic approach to intermittency. Ergodic theory and dynamical systems, 19(3):671–685, 1999. [p91]

P. Massart. Concentration inequalities and model selection, volume 1896 of Lecture Notes in Mathematics. Springer-Verlag Berlin Heidelberg, 2007. [p89]

W. K. Newey and K. D. West. A simple, positive-definite, heteroskedasticity and autocorrelation consistent covariance matrix. Econometrica, 55:703–708, 1987. [p85]

W. K. Newey and K. D. West. Automatic lag selection in covariance matrix estimation. The Review of Economic Studies, 61(4):631–653, 1994. [p85]

M. Rosenblatt. A central limit theorem and a strong mixing condition. Proceedings of the National Academy of Sciences, 42(1):43–47, 1956. [p91]

W. B. Wu and M. Pourahmadi. Banding sample autocovariance matrices of stationary processes. Statistica Sinica, pages 1755–1768, 2009. [p83, 87]

H. Xiao and W. B. Wu. Covariance matrix estimation for stationary time series. The Annals of Statistics, 40(1):466–493, 2012. [p86]

A. Zeileis. Econometric computing with hc and hac covariance matrix estimators. Journal of Statistical Software, 11(12), 2004. [p85, 88]

Emmanuel Caron
Laboratoire de Mathématiques d'Avignon EA2151
Avignon Université
74 Rue Louis Pasteur, 84029 Avignon France
emmanuel.caron-parte@univ-avignon.fr
URL: http://ecaron.perso.math.cnrs.fr/

Jérôme Dedecker
Laboratoire MAP5 UMR 8145
Université Paris Descartes
45 Rue des Saints-Pères, 75006 Paris France
jerome.dedecker@parisdescartes.fr
URL: http://w3.mi.parisdescartes.fr/~jdedecke/

Bertrand Michel
Laboratoire de Mathématiques Jean Leray UMR 6629
Ecole Centrale Nantes
1 Rue de la Noë, 44300 Nantes France
bertrand.michel@ec-nantes.fr
URL: http://bertrand.michel.perso.math.cnrs.fr

exPrior: An R Package for the Formulation of Ex-Situ Priors

by Falk Heße, Karina Cucchi, Nura Kawa, and Yoram Rubin

Abstract The **exPrior** package implements a procedure for formulating informative priors of geostatistical properties for a target field site, called *ex-situ priors* and introduced in Cucchi et al. (2019). The procedure uses a Bayesian hierarchical model to assimilate multiple types of data coming from multiple sites considered as similar to the target site. This prior summarizes the information contained in the data in the form of a probability density function that can be used to better inform further geostatistical investigations at the site. The formulation of the prior uses ex-situ data, where the data set can either be gathered by the user or come in the form of a structured database. The package is designed to be flexible in that regard. For illustration purposes and for easiness of use, the package is ready to be used with the worldwide hydrogeological parameter database (WWHYPDA) Comunian and Renard (2009).

Introduction

Characterizing the subsurface of our planet is an important task in fields such as geology, hydrogeology, and soil sciences. Yet compared to many other fields, the characterization of the subsurface is always burdened by large uncertainties. These uncertainties are caused by the general lack of data and the large spatial variability of many subsurface properties. The need to represent this uncertainty has led to the development of the field of geostatistics, wherein parameter values are treated as random variables defined by their probability distribution function (PDF). Today, the field of geostatistics has reached a mature state with many textbooks on the topic (Pyrcz and Deutsch, 2002; Rubin, 2003; Kitanidis, 2008) and a solid number of software tools being available for practitioners (for an overview, see, e.g. Rubin et al. (2018)). For the R language (R Core Team, 2014), a solid ecosystem for geostatistical analysis has evolved in the last years (Slater et al., 2019). Packages like **geoR** (Ribeiro and Diggle, 2001), **gstat** (Pebesma, 2004), **georob** (Papritz et al., 2014), and **RGeostats** (MINES ParisTech / ARMINES, 2019) provide a large collection of tools for geostatistical analysis. Moreover, geostatistical databases can be conveniently accessed with packages like **aqp** (Beaudette et al., 2013) and textbooks on geostatistics are starting to provide all their examples in R code (Diggle and Ribeiro, 2007; Banerjee et al., 2014).

Bayesian statistics provides the most appropriate framework to characterize uncertainty in general (Heße et al., 2019a). Bayesian methods are able to combine and assimilate data from disparate sources and jointly represent the different forms of uncertainty. As a result, Bayesian methods are nowadays increasingly employed in geostatistics and software implementations come as either standalone versions (Vrugt et al., 2009; Rubin et al., 2010) or R packages like **spBayes** (Finley et al., 2015), **R-INLA** (Lindgren and Rue, 2015), **spTimer** (Bakar and Sahu, 2015), **BayesNSGP** (Risser and Turek, 2020), and **anchoredDistr** (Savoy et al., 2017).

Yet, there is no package to date, which would provide such tools with the necessary foundation, i.e. prior distributions for the modeled quantities. Since the prior is the first step of any Bayesian analysis, its overall importance can hardly be overstated. Moreover, the ability of prior distributions to represent available background information in a given field makes them an important source of information that should not be neglected. Integrating them into a Bayesian workflow should be straightforward since most packages for Bayesian inference allow users to specify their prior distributions. In addition, the use of informative prior distributions in this field is easy to motivate. First, the parameters of geostatistical models are typically not simple convenience parameters but are part of physically-based partial differential equations. As a result, they correspond to real-world, physical measurements, making it possible to calibrate their prior distributions against empirical frequencies. Second, geostatistical models are usually site specific, making it conceptually easy to discriminate between the case-specific data, which should be used to compute the likelihood, and background data, which could be used to compute the prior distribution. In geostatistics, they are often called in-situ and ex-situ data, respectively. Calibrating the prior against ex-situ data only, therefore guarantees a clear separation between likelihood and prior.

To provide practitioners therefore with a tool for prior derivation, this paper introduces the R package **exPrior** (Heße et al., 2019b). It implements the derivation of ex-situ priors, i.e., statistical distributions of subsurface properties at a given site from ex-situ data collected at similar sites, following the Bayesian hierarchical model developed by Cucchi et al. (2019). The implementation is based on the **nimble** package (de Valpine et al., 2017) itself based on the BUGS language (Lunn et al., 2009). The objective of the **exPrior** package is to provide a ready-to-use software tool for

assimilating ex-situ data into ex-situ priors. This will encourage the use of informative distributions for practitioners in geosciences that might not be experts in Bayesian hierarchical models and may find it therefore difficult to work with them otherwise. The focus of this package is on the Gaussian process (GP) modelling paradigm. Although criticism exists, it is by far the most widely used paradigm, and a wide range range of software tools exist for the modeling of GPs (Pebesma, 2004). Functions in **exPrior** provide wrappers around **nimble** functions implementing the Bayesian data assimilation framework. Non-expert practitioners can therefore apply this method without needing to implement the model itself. Moreover, the package is tightly integrated with two other R packages that help to expand its functionality. First, the **geostatDB** package (Heße et al., 2019c) provides access to a large data set from the worldwide hydrogeological parameter database (WWHYPDA) Comunian and Renard (2009). Second, the **siteSimilarity** package (Kawa et al., 2020) allows for clustering of similar sites and therefore facilitates a further reduction of uncertainty.

Due to the background of the authors, the examples and data are drawn from stochastic hydrogeology, i.e., the field of geostatistics concerned with the statistical characterization of groundwater systems. Yet, the presented package is not confined to this field, and simply using other data or making slight revisions to the hierarchical model will quickly make the workflow amendable to other fields of geostatistics as well.

To familiarize the reader with the package, we start in the following by explaining the workflow for formulating informative prior distributions, where the prior distribution at an unexplored site is based on data collected from other sites. After that, we explain the package by detailing its structure and functionalities. Finally, we present several examples of prior derivation based on synthetic data and on an established database for hydrogeological parameters (Comunian and Renard, 2009).

Ex-Situ Priors

Let us assume that we want to model a specific geostatistical variable x at a target site S_0. Examples would be hydraulic conductivity, porosity, or permeability. To account for the unavoidable uncertainty, this variable should be modeled as a random variable X (Pyrcz and Deutsch, 2002; Rubin, 2003; Kitanidis, 2008). The simplest way to characterize this variable statistically is through its distribution $p(x)$. Yet, this would leave out any spatial correlations, so most geostatistical analyses try to account for them by using spatial random field models, typically a GP (Rasmussen and Williams, 2006; Gelfand and Schliep, 2016). Since such models are fully defined by their parameter vector θ, the aim of Bayesian inference is to use available, in-situ data y_{in} and derive the posterior distribution over these parameters $p(\theta|y_{in})$. This posterior represents a compromise between likelihood $p(y_{in}|\theta)$ and the prior distribution $p(\theta)$, with the likelihood representing the impact of the in-situ data. This, however, leaves open the specification of the prior distribution.

By definition, the prior distribution characterizes the knowledge about target parameters before observing in-situ data y_{in}. Therefore, y_{in} cannot be used for the definition of the prior (Berger, 2006). On the other hand, using no data and making the prior distribution as vague as possible seems far too prudent since this would ignore the wealth of background knowledge which exists for virtually any geostatistical variable. Such background knowledge can come from data collected at other sites $S_i, i \in 1 \dots I$ (see schematic in Figure 1). To distinguish them from the site-specific, in-situ data y_{in}, we use the term ex-situ data y_{ex}. Our prior pdf for the parameters at a new site S_0 could therefore be based on these ex-situ data $p(\theta|y_{ex})$. To determine this $p(\theta|y_{ex})$, we propose the use of a dedicated statistical model (more on this below). By virtue of this model, the transfer of information from known donor sites S_i to a new site S_0 is a case of Bayesian prediction

$$p(\theta|y_{ex}) = \int_{\vartheta} p(\theta|\vartheta) p(\vartheta|y_{ex}) d\vartheta. \tag{1}$$

According to Eq. 1, the prior distribution $p(\theta|y_{ex})$ for a new site S_0 is the posterior predictive distribution of all S_i w.r.t. S_0 (see schematic in Figure 1). Mathematically, this means $p(\theta|y_{ex})$ is derived by weighing each single predictive distribution $p(\theta|\vartheta)$ with its corresponding posterior distribution $p(\vartheta|y_{ex})$ and marginalizing over the parameters ϑ. Please note the difference between the model for X at site S_0, say a GP defined by θ, and the model used for the transfer of data between sites defined by ϑ. Since geostatistical data are hierarchical in nature, this model should be hierarchical too (Kruschke, 2010; Gelfand, 2012; Gelman et al., 2013).

Formulation of the hierarchical model

In geostatistics, a common way to conceptualize a hierarchical ordering of the data is by using two levels (see schematic in Figure 1). The first level represents the population of the (spatially distributed)

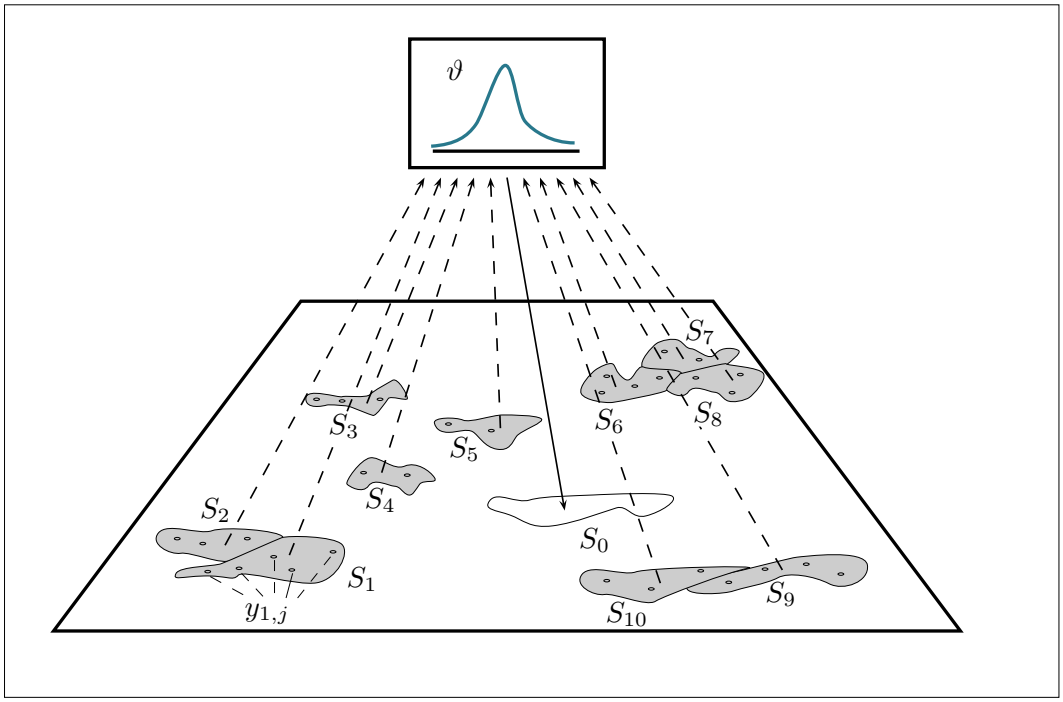


Figure 1: Schematic of the transfer of data from a number of donor sites S_i to a new site S_0. Measurement locations are denoted by circles. The statistical model used for the transfer is denoted by its parameter vector ϑ.

local data $y_{i,j}$ available on every given site S_i whereas the second level represents the global population of all available sites. In such a two-level hierarchical model, this relationship is represented such that the set of parameters ϑ is split up into two subsets $\vartheta = (\phi, \eta)$ for level 1 and 2 respectively. The hierarchical relationship between these two levels is represented by factorizing their joint probability through the definition of conditional probability $p(\phi, \eta) = p(\phi|\eta)p(\eta)$. The general formulation of the two-level model would then look like the following:

$$y_{i,j} \sim p(y|\phi_i), \tag{2a}$$

$$\phi_i \sim p(\phi|\eta), \tag{2b}$$

$$\eta \sim p(\eta). \tag{2c}$$

This means that each datum $y_{i,j}$ is drawn from a distribution with parameters ϕ_i, which are specific to site S_i only. This distribution, therefore, represents the local variability of the data found within a given site or intra-site variability (Eq. 2a). These local parameters ϕ_i are, in turn, drawn from a distribution specified by the global parameters called *hyperparameters* η. These hyperparameters, therefore, represent the global variability between sites or inter-site variability (Eq. 2b).

This general formulation allows to flexibly choose parametric models used for all distributions. Since $p(y|\phi_i)$ represents the data, this distribution should fit the empirically observed frequencies of y. Depending on the geostatistical parameter of interest, a user can use, e.g., the normal, log-normal, multivariate normal, or truncated normal distributions to model parameter behavior. Choosing the distributions of the hierarchical model itself, i.e., $p(\phi|\eta)$ and $p(\eta)$ is less straightforward and should reflect of mixture of the domain knowledge and statistical expertise. To exemplify this procedure, let us look at measurements of hydraulic conductivity. These data are often modeled with a log-normal distribution (Hoeksema and Kitanidis, 1985), while $p(\phi|\eta)$ can be modeled as a normal distribution (Gelman et al., 2013). The parametric form of $p(\eta)$ should be specified as vague priors initially, with the posteriors being determined by the data y_{ex}. Transforming our data into their log-normal form, as often done, the resulting hierarchical model would then be

$$y_{i,j} \sim \mathcal{N}(\mu_i, \sigma^2), \tag{3a}$$

$$\mu_i \sim \mathcal{N}(\alpha, \tau), \tag{3b}$$

$$(\sigma^2, \alpha, \tau) \sim p(\sigma^2)p(\alpha)p(\tau). \tag{3c}$$

In this example, we assumed that the variance σ^2 at each site is the same, making the local and global parameters identical. This assumption is, of course, a simplification but allows to reduce the number of parameters to be inferred.

Generation of the ex-situ prior distribution

Once the target variable is specified, and the ex-situ data y_{ex} collected, three steps are necessary to actually calculate the prior distribution. First, the user has to decide on the parametric model for the distributions and therefore fully specify the hierarchical model according to Eq. 2. Next, the posterior distributions of the parameters $p(\phi, \eta|y_{ex})$ are inferred. In **exPrior**, this is done using the Markov chain Monte Carlo (MCMC) implementation of **NIMBLE**. Finally, the ex-situ prior can be determined as the posterior predictive distribution (see Equation 1).

To familiarize ourselves with this procedure, let us look at these three steps in more detail.

Specifying a hierarchical model The specification of the hierarchical model in **exPrior** is done in `BUGS` code wrapped within the **NIMBLE** function `nimbleCode()`. The results are `R` objects from the `BUGS` models. In **NIMBLE**, every model is represented as a Directed Acyclic Graph (DAG), where each declaration in the model is a node which can be either deterministic or stochastic. Nodes are represented as vertices of a DAG, with edges connecting nodes implying dependence relationships.

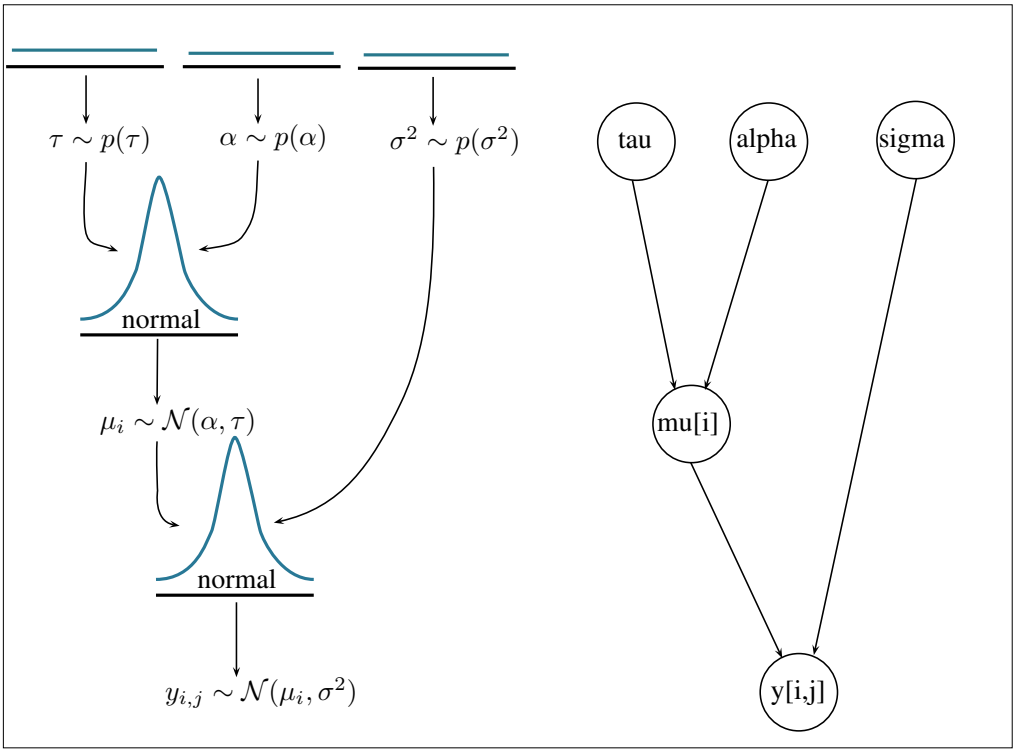


Figure 2: Schematic of the example model showing the statistical model on the left as defined in see Eq. 3 and the corresponding DAG on the right. The arrows show the hierarchical relationships between the variables. Both the ex-situ data $y_{i,j}$, and the site-specific mean μ_i are drawn from normal distributions. The hyperparameters $\eta = (\tau, \alpha, \sigma^2)$ are given initially vague hyperpriors to be updated later by the ex-situ data.

To exemplify this, let us consider the aforementioned model for the log-hydraulic conductivity. As mentioned, the data at each site are modeled as being drawn from a normal distribution with every site having the same variance σ^2 but site-specific means. This mean value is again drawn from a normal distribution. Accordingly, α is the global mean of the local means, and τ^2 represents the

global variability of the local means. Since no data are used at this point, the hyperpriors of the hyperparameters $\eta = (\alpha, \tau, \sigma)$ should be non-informative.

The model is therefore declared with five variables: `alpha`, `tau`, `sigma`, `mu`, and `y`. Once compiled, the model contains multiple nodes: one node for each hyperparameter `alpha`, `tau`, and `sigma`, `I` nodes for the site-specific means `mu[i]`, and $\sum_i J_i$ nodes for the site measurements `y[i,j]`. In this model, the dependent nodes are the ex-situ data $y_{i,j}$, the μ_i values are estimated from these deterministic $y_{i,j}$. Similarly, the α and τ are estimated from the μ_i. In cases where the data are provided to `genExPrior()` in the form of moments (more on this later), `mu[i]` is deterministic, as the site-specific means are already provided in numeric form and not considered to be realizations from any distribution. The following pseudocode illustrates how the model is declared:

```
## declaration of a hierarchical model in nimbleCode

# 1. declare prior distributions for hyperparameters
hyperparameters ~ disn(...)

# 2. declare distributions for site-specific means
for i in 1:I {
  mu[i] ~ disn(hyperparam1, hyperparam2, ...) # distribution of mean mu at site i

# 3. declare distributions for measurements
  for j in 1:J {
  y[i,j] ~ disn(mu[i], hyperparam3, ...)
  }
}
```

This pseudocode model denotes the mathematical formulation of the hierarchical model in BUGS code, where the parameter for site-specific means μ_i is `mu[i]`, and site-specific variances σ_i^2 is `sigma2[i]`. First, hyperparameters `alpha` and `tau` are assigned non-informative hyperpriors. Next, the code loops through each site and assigns the mean `mu[i]` a distribution whose parameters are hyperparameters. Finally, each of the `J` measurements `y[i,j]` at each site `i` is assigned a distribution with parameters `mu[i]` and hyperparameters.

The flexibility of this formulation allows the data y_{ex} to be assimilated in a full MCMC hierarchical model, as explained in the next section.

Estimating posterior values of parameters in the hierarchical model Parameters in the hierarchical model are estimated from the data provided by the user, using MCMC. Once the model and the variables are declared, the hierarchical model is compiled using `nimble::compileNimble()`. The MCMC object is configured, built, and compiled using `nimble::configureMCMC()`, `nimble::buildMCMC()`, and `nimble::compileNimble()`, respectively and run using the run method of the compiled object. This method calculates an MCMC chain, the result of the estimation step. To improve numerical efficiency, NIMBLE includes a library of algorithms and a compiler that generates C++ for declared models and functions. Once a model is declared, `nimbleCode` is generated as C++ code, compiled, and reloaded into R.

The values in the compiled model are declared with data that are supplied by the user when running the function. For example, if a user inputs a data frame of measurements, each of the `y[i,j]` is defined with its corresponding data point. If a user provides moments, then each of the `mu[i]` is defined with its provided value. Once a model is compiled, `genExPrior()` envokes the MCMC sampler, which estimates the posterior distributions of the parameters.

Predicting the prior Now that the ex-situ data are assimilated, our hierarchical model is fully conditioned on all available data. Normally, this would conclude a Bayesian inference. Yet as explained above, the posterior distribution has to be used to compute the predictive posterior distribution of the target variable at a new site (see Eq 1). In **exPrior**, this is simply done by drawing realizations of the target variable from distributions specified by the hierarchical model and parameterized by values sampled from the MCMC chains from the previous step. The final predictive posterior distribution is then estimated using kernel density estimation.

Associated packages

To support the functionality of **exPrior**, we provide two additional R packages on the GitHub account. First, the **geostatDB** package provides real-world data on subsurface measurements, which can

therefore be used directly for the derivation of informed prior distributions. Second, the **siteSimilarity** package allows users to determine a cluster of similar sites to focus on the most relevant data only and therefore reducing the overall uncertainty. Since both these packages provide benefits independently of **exPrior**, they will remain independent for the foreseeable future.

Real-world data: The geostatDB package

In this package, we provide real-world, geostatistical data from the WorldWide HYdrogeological Parameters DAtabase *WWHYPDA* (Comunian and Renard, 2009). This database has been designed to store values of the most important properties of earth materials and has been developed with the purpose of offering a complement of information in hydrogeological studies where there is a lack of data. To the best of the authors' knowledge, the WWHYPDA is the largest open-source database of hydrogeological parameters. Currently it contains a total of 20,523 subsurface measurements of 6 geostatistical parameters spanning 128 sites. A complete description and schematic of the database in its original form can be found in Comunian and Renard (2009).

Due to its size, as well as to facilitate further additions to the database, we use a dedicated R package **geostatDB** to host the data from WWHYPDA. This package is not yet on CRAN, but the latest release version can be found on `https://github.com/GeoStat-Bayesian/geostatDB` (Heße et al., 2019c). To exemplify the usage and impact of real-world data, we furnished **exPrior** with an example data set on porosity values from sandy aquifers. Its usage is described in Section 2.4.2 below.

geostatDB itself includes the WWHYPDA as an SQLite database, converted from the online MySQL database. The reasons for using SQLite are efficiency and accessibility. First, both SQL and SQLite databases can be easily read into `R`, while maintaining their original structure. SQLite specifically can be included in R packages without the need for a server, making it accessible to the user. A downside of this decision is that a user who updates the SQLite database in **geostatDB** does so without making changes to the original database, hosted on `https://www.wwwhypda.org`. Thus, those who wish to contribute to the WWHYPDA are currently encouraged to do so by submitting data online.

It is important to note that data quality steps need to be implemented before applying the statistical algorithm to this database. Figure 3 contains two visualizations that describe the data present in the WWHYPDA, created in `R`. This visualization was done using the function `getData()` from the **geostatDB** package. The code can be found in the associated vignette of the package at `https://github.com/GeoStat-Bayesian/geostatDB/blob/master/vignettes/explore_data.Rmd`.

The notion of site similarity: The siteSimilarity package

In order to reduce the uncertainty in the prior distribution as much as possible, it is beneficial to focus only on data coming from sites which are similar to the one under investigation. It is therefore crucial to have a sound notion of site similarity. The **siteSimilarity** package uses hierarchical agglomerative clustering to categorize sites into clusters based on observable characteristics, such as environment type or rock type. Using the schematic in Figure 1, only those sites similar to site S_0 would be used as donor sites. Currently, the clustering achieves only a modest reduction in uncertainty when using a leave-one-out validation. This is caused by the overall limited number of sites, which, after clustering, get even more reduced. Yet the algorithm already provides the user with a tangible benefit, which is projected to increase as more and larger data sets with more sites become available.

Examples

Having now formally explained the workflow and associated packages of **exPrior**, we will illustrate said workflow with a series of examples. Starting with an easy inference problem, we will then explain how to use the included data, how to account for autocorrelation, and finally how to use soft data, in particular bounds, for the inference.

Please note that all examples given in the following only refer to the distribution of the expected quantity itself, e.g., porosity. This does, however, not mean that the parameters of a GP cannot be inferred since μ and σ are nodes in the hierarchical model and can be derived from it. On the other hand, higher-order statistics, like correlation length or anisotropy, are usually considered homogeneous across a given site and are consequentially not hierarchical. They can, therefore, be inferred using the classical estimation procedure.

The following four examples correspond to four vignettes, which can be found on the GitHub account of the **exPrior** package at `https://github.com/GeoStat-Bayesian/exPrior/blob/master/`

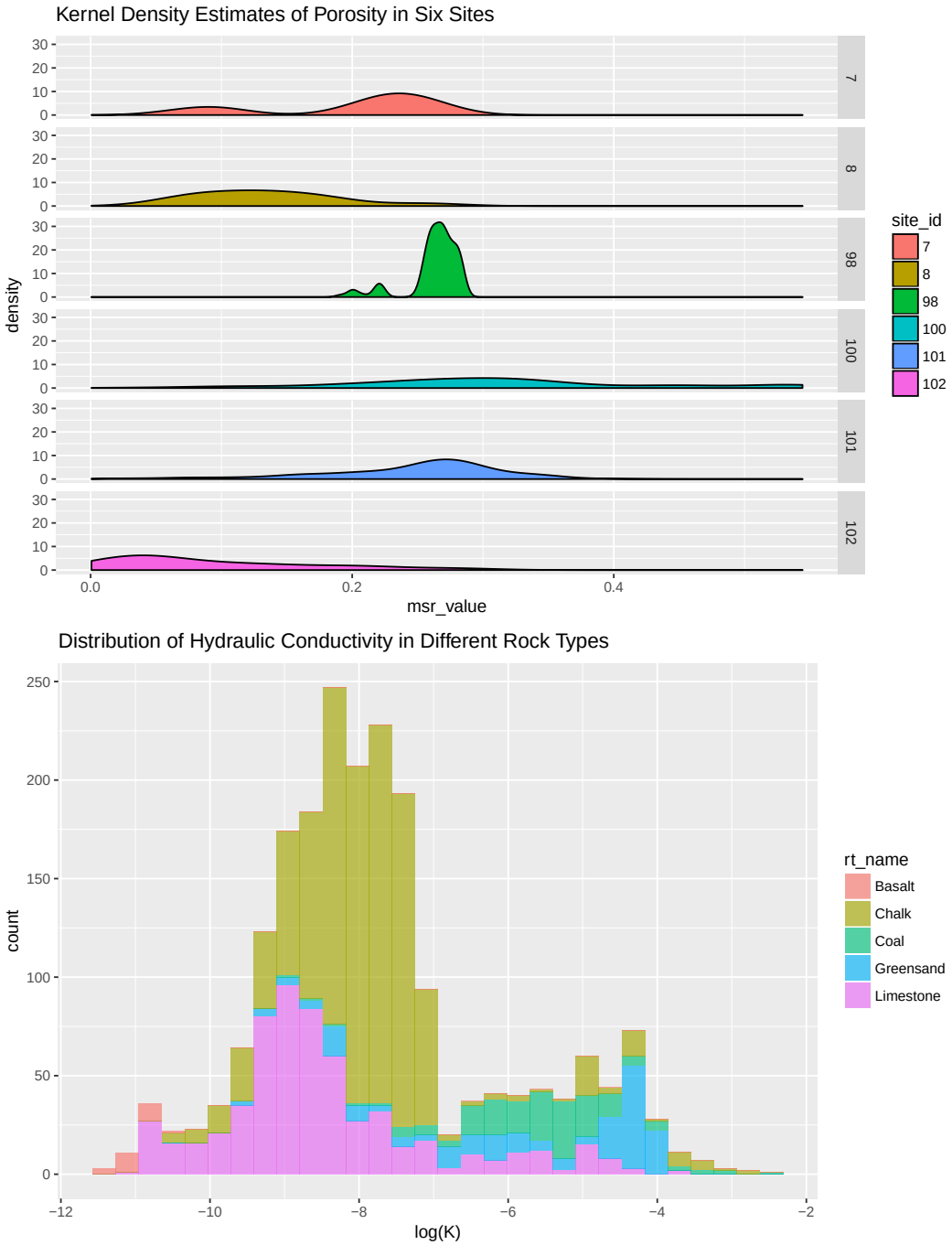


Figure 3: The two visualizations are made using the data from the WWHYPDA, obtained using the function `getData()` from the **geostatDB** package. The first figure shows the distribution of porosity values at several sites, derived using kernel density estimation. The second figure shows a set of histograms describing the distribution of hydraulic conductivity values for different material types.

vignettes.

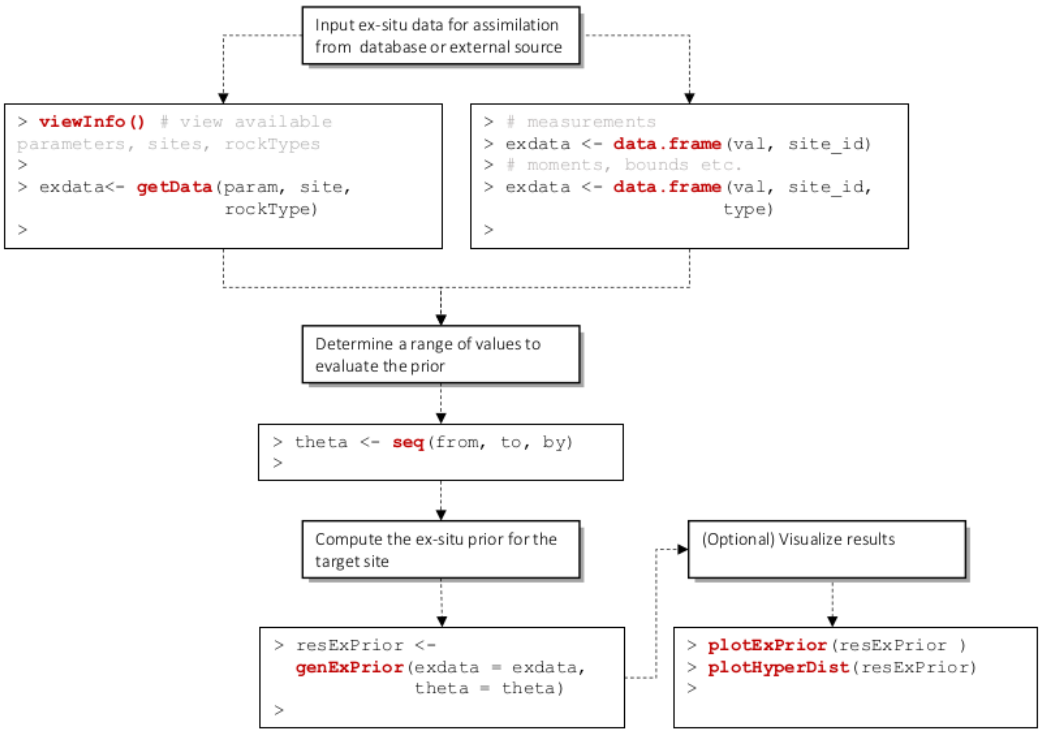


Figure 4: A flowchart showing the workflow of the user of **exPrior**. A user first expresses ex-situ data as an R dataframe. The user then determines the range of values at which to compute a prior (usually the minimum and maximum values of a parameter). Finally, the user uses `genExPrior()` to compute the ex-situ prior for a target site. The user has the options of using built-in plotting functions to visualize results.

The general workflow, which is followed in all of these examples, is visualized in Figure 4. First, a user has to represent the ex-situ data in the form of an `R dataframe` object. These data can be manually entered, like explained in Section 2.4.1, taken from the included data, like explained in Section 2.4.2, or being supplied though some additional database. Next, the user would enter as an `R vector` object a range of values at which to estimate the prior distribution. This range is typically the minimum and maximum values of the parameter of interest (for example, porosity takes values between 0 and 1). Finally, the user would input the ex-situ data and specified range into the function `genExPrior()`, which outputs a prior and the distributions of hyperparameters of the model. The user has the options of using built-in plotting functions to visualize results.

Example 1: Using exPrior with synthetic data

To familiarize the reader with this general workflow, let us start with a simple example using only a few synthetic data (on a log 10 scale) from three arbitrarily labeled sites S_1, S_2, and S_3. The source code for the corresponding vignette can be found at https://github.com/GeoStat-Bayesian/exPrior/blob/master/vignettes/using_genExPrior.Rmd. The goal is to derive the ex-situ prior for target site S_0 with the following code:

```
> exdata <- data.frame(val = c(c(-2,-3,-4), c(-2,-1), c(-6,-7,-2,-3)),
+                      site_id = c(rep('S1',3), rep('S2',2), rep('S3',4)))
> ex_prior <- genExPrior(exdata = exdata, theta = seq(from=-10, to=10, by=0.1))
```

By following the above workflow, we started with generating an `R dataframe exdata` for the ex-situ data. Then, we entered the range over which to estimate the variable θ as an `R vector theta`. The actual computation of the ex-situ prior is finally performed by the function `genExPrior()`. To investigate the output of this function, **exPrior** provides a number of plotting functions.

```
> plotHyperDist(ex_prior)
> plotExPrior(ex_prior)
```

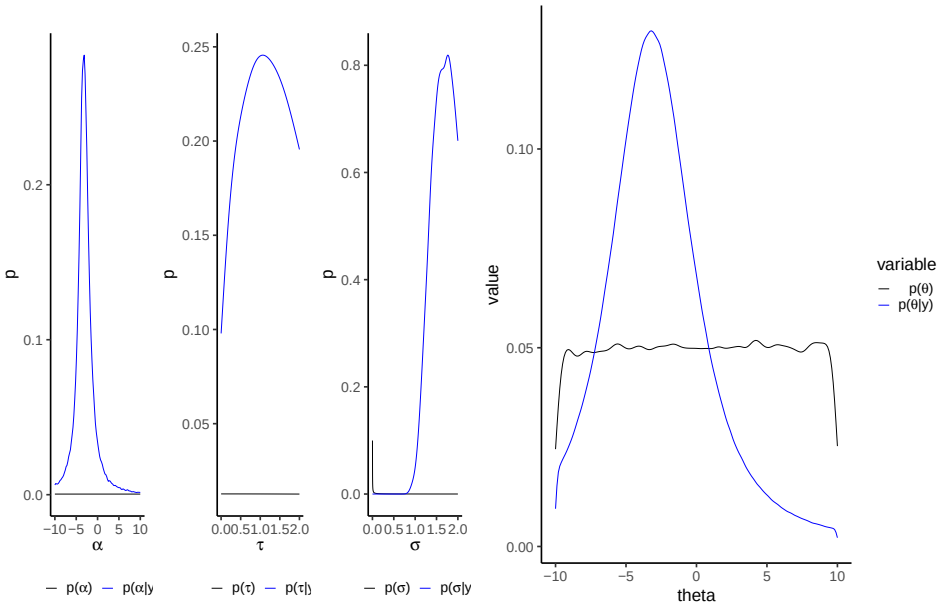


Figure 5: The left panel shows the distributions of the hyperparameters `alpha`, `tau`, and `sigma`. The right panel shows ex-situ prior computed using the data assimilation framework (blue curve) against the uninformative prior (black curve).

In this little example, the first command `plotHyperDist` plots the posterior distributions of the hyperparameters α, τ, and σ (see Figure 5 left panel). This captures the impact of the data on the Bayesian hierarchical model. The second command `plotExPrior` shows the ex-situ data from the three sites jointly with the predicted prior distribution for the new site S_0 (see Figure 5 right panel). As can be seen, the essentially flat, uninformative prior got updated into a much sharper, informative prior representing a much-reduced uncertainty.

Example 2: Using exPrior with real-world data

As introduced above, **exPrior** provides real-world, geostatistical data from the WWHYPDA. Let us exemplify its use and impact on the inference by first importing the data on porosity. As above, the associated vignette can be found at https://github.com/GeoStat-Bayesian/exPrior/blob/master/vignettes/real_world_data.Rmd.

```
> load(file="data/df_porosity.rda")
```

These real-world data are now loaded into the workspace and can be used to compute the ex-situ prior using the 'exPrior' function as describe in above example

```
> resExPrior = genExPrior(exdata = df_porosity, theta = seq(from=0, to=1, by=0.01))
```

Here, the range of the `theta` vector reflects the common-sense intuition that porosity values can only exist between 0 and 1. This change should also be reflected in the used model

$$y_{i,j} \sim \Phi(\mu_i, \sigma^2, 0, 1), \tag{4a}$$

$$\mu_i \sim \mathcal{N}(\alpha, \tau), \tag{4b}$$

$$(\sigma^2, \alpha, \tau) \sim p(\sigma^2)p(\alpha)p(\tau). \tag{4c}$$

Equation (4a) makes this change clear, such that the data are drawn from a truncated normal distribution. Since the boundaries are fixed, the hierarchical model itself still has the same number of parameters, and the other parts of the model remain the same.

After the completion of `exPrior`, we can visualize again the posteriors of the model as well as the prior of θ using `plotExPrior`.

```
> plotHyperDist(resExPrior)
> plotExPrior(resExPrior)
```

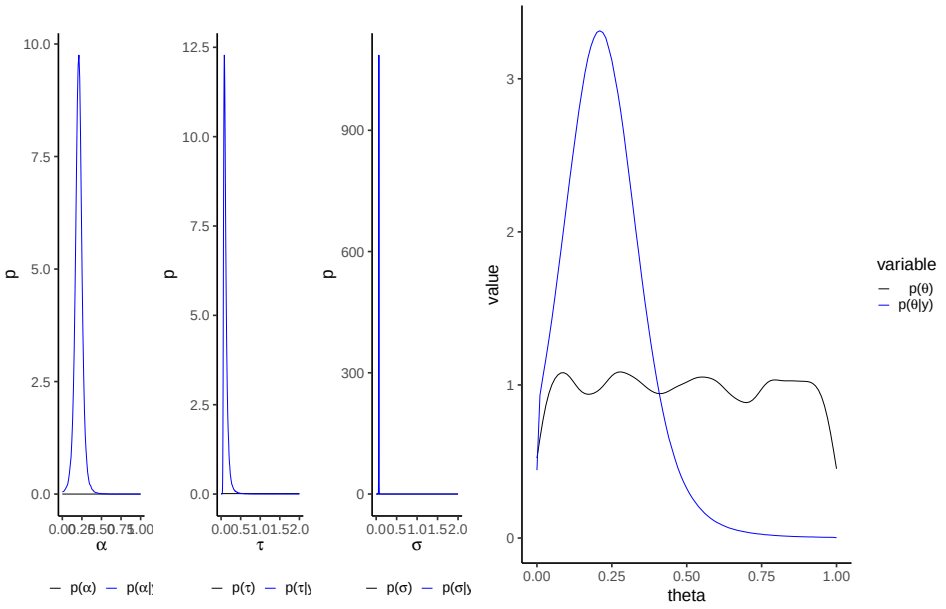


Figure 6: Informative (blue) and non-informative (black) priors computed with `genExPrior()` using real-world, ex-situ data of porosity in sand stone.

Compared to Figure 5, the results in Figure 6 show some relevant differences. In particular, the hyperpriors seen in the left panel of Figure 6 are much more peaked, resulting in near-certainty about their value. This is due to the large amount of evidence provided by the data. Since the parameter distributions on the higher levels in a hierarchical model represent the uncertainty about the parameters on the lower ones, it can be said that the results of this inference capture the uncertainty of porosity in sandstone with high certainty. This means that the prior distribution in the right panel of Figure 6 is very close to the statistical uncertainty for the hypothetical population of all porosity values in sandstone aquifers in general. As can be seen in this figure, this distribution is strongly peaked between 0.2 and 0.3. Using this prior therefore provides a practitioner with a sound foundation for the geostatistical inference of the in-situ porosity.

Example 3: Accounting for spatial autocorrelation in ex-situ data

In most cases, ex-situ data used in the analysis are spatially correlated since measurements are usually collected in a clustered way (Rubin, 2003; Pyrcz and Deutsch, 2002). The data assimilation model outlined in 2.2.1 can, in principle, account for patterns of spatial variability by using multivariate distributions as site-specific distributions. As above, the associated vignette can be found at `https://github.com/GeoStat-Bayesian/exPrior/blob/master/vignettes/spatial_correlation.Rmd`. To account for this spatial correlation, let us use a revised version of the hierarchical model from Equation (3)

$$y_{i,j} \sim \mathcal{N}(\mu_i, \Sigma), \tag{5a}$$

$$\mu_i \sim \mathcal{N}(\alpha, \tau), \tag{5b}$$

$$\Sigma = \sigma^2 \exp\left(-\frac{h}{\lambda}\right), \tag{5c}$$

$$(\sigma^2, \lambda, \alpha, \tau) \sim p(\sigma^2)p(\lambda)p(\alpha)p(\tau). \tag{5d}$$

The relevant adjustment can be seen in Equation (5a), where the data are no longer modeled to be drawn from a univariate normal distribution but a multivariate distribution instead. The main difference is the replacement of the variance σ^2 by the covariance Σ. In our example, this covariance is modeled as an isotropic exponentially decaying function, with a characteristic length scale λ. This function means that measurements being taken at large distances h are essentially independent, and no relevant difference to the simple univariate model from Equation (3) would exist. However, measurements taken at distances h similar or smaller to λ exhibit substantial correlation and must be

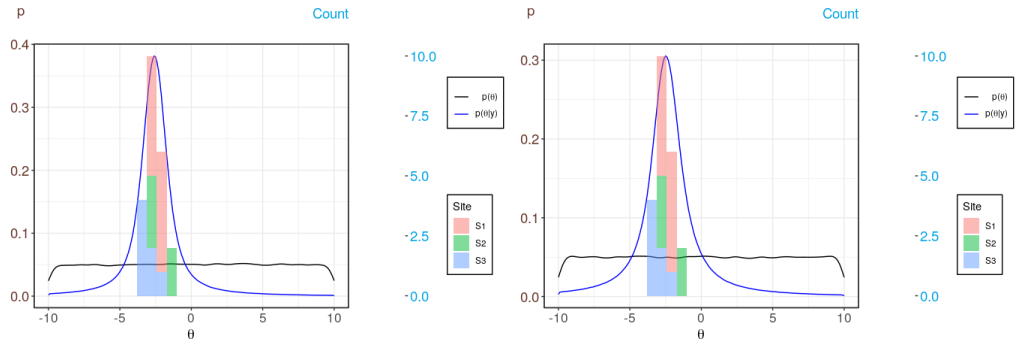


Figure 7: Informative (blue) and non-informative (black) priors computed with `genExPrior()`. The left panel shows the prior without accounting for spatial correlation, whereas the right panel shows the prior accounting for spatial correlation (right panel). The colored bars represent the ex-situ data from the three different sites.

assimilated accordingly. Failing to do so would result in an underestimation of the actual uncertainty, a phenomenon which is known in the literature as pseudoreplication (Hurlbert, 1984; Legendre, 1993). Due to the additional parameter, the model has now 4 hyperparameters, which need to be inferred.

To exemplify the workflow with this revised hierarchical model, let us use synthetic data coming again from three different sites only. To generate data with spatial correlation, we used the **gstat** package. These synthetic data were then transformed to have different mean values for each site.

```
> set.seed(1)
> xy <- data.frame("x" = sample(seq(0.00,1.00,0.01),22),
+                  "y" = sample(seq(0.00,1.00,0.01),22))
> model = vgm(psill=1, range=1, model='Exp')
> g.dummy <- gstat(formula=z~1, locations=~x+y, dummy=TRUE, beta=1, model=model, nmax=20)
> exdata_spatial <- predict(g.dummy, newdata=xy, nsim=1)
```

To adapt this data frame from **gstat** to the format needed for **exPrior**, we have to change one of the column names and add the site's id.

```
> colnames(exdata_spatial)[3] <- "val"
> exdata_spatial\$site_id = c(rep("S1", 10), rep("S2", 5), rep("S3", 7))
> exdata_spatial[ 1:10, 'val'] <- exdata_spatial[ 1:10, 'val'] - 3
> exdata_spatial[11:15, 'val'] <- exdata_spatial[11:15, 'val'] - 2.5
> exdata_spatial[16:22, 'val'] <- exdata_spatial[16:22, 'val'] - 3.5
```

With these data, we can now generate the ex-situ prior distribution. To tell **exPrior** to account for the spatial correlation in the data, we have to toggle the `spatialCoordinates` flag in the `genExPrior()` function to `TRUE`.

```
> resExPrior = genExPrior(exdata = exdata, theta = seq(from=-10, to=10, by=0.1),
+                         spatialCoordinates = TRUE)
> plotExPrior(resExPrior, plotExData = TRUE)
```

To compare the effects of accounting for spatial correlation, we provide plots of the ex-situ prior with `spatialCoordinates` being both set to `FALSE` and `TRUE` (see the left and right panel in Figure 7, respectively). As can be seen, both priors look overall similar in shape. The main difference is that the latter shows a somewhat increased uncertainty, i.e., a wider variance, which can be seen by the increased mode of the distribution. The fact that the more realistic model produces more uncertain results may seem counterintuitive at first. However, the aim of statistical inference is not to reduce the uncertainty as much as possible but to correctly capture the uncertainty in the used data and the model. As mentioned above, this problem of not accounting for possible correlations between measurements is called pseudoreplication and can have serious consequences by leading to overconfident statistical analyses.

Example 4: Assimilating Multiple Data Types

As mentioned above, **exPrior** is written in a flexible manner, such that it can assimilate data that come in the form of measurements, bounds, or moments (see schematic in Figure 2). To exemplify this

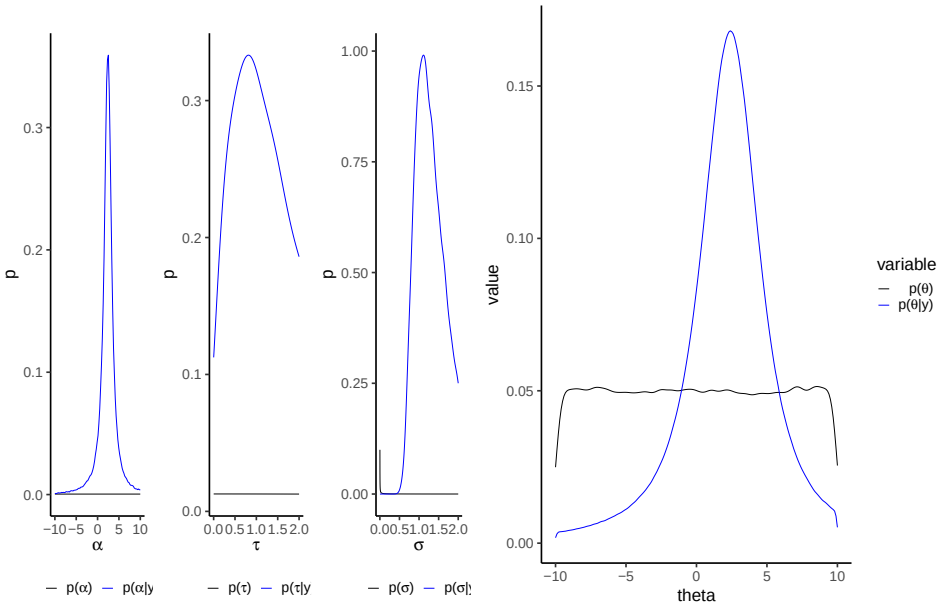


Figure 8: The left panel shows the distributions of the hyperparameters `alpha`, `tau`, and `sigma`. The right panel shows ex-situ data from three synthetic sites S_1, S_2, and S_3. The blue curve is the ex-situ prior computed using the data assimilation framework, while the black curve is the uninformative prior.

flexibility, let us use in this example synthetic data from three sites labeled S1, S2, and S3. From Site S1, we have data in the form of bounds, where the minimum value of a hydrogeological property of S1 is 2, and its maximum value is 4. Site S2 has data in the form of moments, where the first moment, or site mean, is 2, while the second moment, or site variance, is 0.1. Finally, site S3 has three measurements. Again, the associated vignette can be found at https://github.com/GeoStat-Bayesian/exPrior/blob/master/vignettes/multi_type_data.Rmd. The code below shows how to format the data in R such that it can be read into `genExPrior()`.

```
> exdata_S1 <- data.frame(val=c(2,4), site_id=rep('S1',2),
+                         type=c('bound.min','bound.max'))
> exdata_S2 <- data.frame(val=c(2,0.1), site_id=rep('S2',2),
+                         type=c('moment.1','moment.2'))
> exdata_S3 <- data.frame(val=c(2,3,4), site_id=rep('S3',3),
+                         type=c('meas','meas','meas'))
> exdata <- rbind(exdata_S1, exdata_S2, exdata_S3)
```

As in previous examples, the data frame `exdata_multitype` as well as the vector `theta` can be input directly into `genExPrior()` as such

```
> resExPrior <- genExPrior(exdata = exdata, theta = seq(from=-10, to=10, by=0.1))
```

Finally, we can visualize the results `resExPrior` again using the `plotHyperDist` and `plotExPrior` functions.

```
> plotHyperDist(resExPrior)
> plotExPrior(resExPrior)
```

The resulting hyperparameters and ex-situ prior distributions look very similar to the simple example from Section 2.4.1 (compare Figure 5 to Figure 8). This comparison shows that data in the form of bounds and moments can have a similar impact on the inference and how they can be assimilated by `exPrior`.

Summary

In this paper, we have introduced the R package **exPrior**, which contains methods for assimilating ex-situ data to generate prior probabilities for geostatistical parameters. We explain the formulation

of a prior distribution as a Bayesian Hierarchical Model (Section 2.2.1) and its implementation using **NIMBLE**, an R package created for efficient hierarchical modeling. We illustrate the model through a number of examples where **exPrior** can be used, including univariate and multivariate models (Section 2.4.3), as well as the assimilation of multiple data types (2.4.4). The package also contains data from the WWHYPDA, an open-source, hydrogeological database that provides valuable information for hydrogeological modeling. The goal of this package is to provide methods to facilitate geostatistical modeling, as well as to encourage the open-source and open-data movements between scientists.

Acknowledgements

This work has been partly funded by the German Research Foundation (DFG) under grant HE 7028/2-1, "What we talk about when we talk about uncertainty." Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the DFG.

Bibliography

K. S. Bakar and S. K. Sahu. spTimer: Spatio-Temporal Bayesian Modeling Using R. *Journal of Statistical Software*, 63(15):32, 2015. ISSN 1548-7660. doi: 10.18637/jss.v063.i15. [p101]

S. Banerjee, B. P. Carlin, and A. E. Gelfand. *Hierarchical Modeling and Analysis for Spatial Data*. Chapman & Hall/CRC Monographs on Statistics and Applied Probability. Chapman and Hall/CRC, 2nd edition, Sep 2014. [p101]

D. Beaudette, P. Roudier, and A. O'Geen. Algorithms for quantitative pedology: A toolkit for soil scientists. *Computers & Geosciences*, 52:258 – 268, 2013. ISSN 0098-3004. doi: https://doi.org/10.1016/j.cageo.2012.10.020. [p101]

J. Berger. The Case for Objective Bayesian Analysis. *Bayesian Analysis*, 1(3):385–402, 2006. ISSN 1931-6690. doi: 10.1214/06-BA115. [p102]

A. Comunian and P. Renard. Introducing wwhypda: a world-wide collaborative hydrogeological parameters database. *Hydrogeology Journal*, 17(2):481–489, 2009. ISSN 1435-0157. doi: 10.1007/s10040-008-0387-x. [p101, 102, 106]

K. Cucchi, F. Heße, N. Kawa, C. Wang, and Y. Rubin. Ex-situ priors: A Bayesian hierarchical framework for defining informative prior distributions in hydrogeology. *Advances in Water Resources*, 126:65 – 78, 2019. ISSN 0309-1708. doi: 10.1016/j.advwatres.2019.02.003. [p101]

P. de Valpine, D. Turek, C. J. Paciorek, C. Anderson-Bergman, D. T. Lang, and R. Bodik. Programming With Models: Writing Statistical Algorithms for General Model Structures With NIMBLE. *Journal of Computational and Graphical Statistics*, 26(2):403–413, 2017. doi: 10.1080/10618600.2016.1172487. [p101]

P. J. Diggle and P. J. Ribeiro. *Model-based Geostatistics*. Springer, 1rst edition, 2007. [p101]

A. O. Finley, S. Banerjee, and A. E. Gelfand. spBayes for Large Univariate and Multivariate Point-Referenced Spatio-Temporal Data Models. *Journal of Statistical Software*, 63(13):1–28, 2015. ISSN 1548-7660. doi: 10.18637/jss.v063.i13. [p101]

A. E. Gelfand. Hierarchical modeling for spatial data problems. *Spatial Statistics*, 1(Supplement C):30 – 39, 2012. ISSN 2211-6753. doi: 10.1016/j.spasta.2012.02.005. [p102]

A. E. Gelfand and E. M. Schliep. Spatial statistics and Gaussian processes: A beautiful marriage. *Spatial Statistics*, 18(A, SI):86–104, Nov 2016. ISSN 2211-6753. doi: 10.1016/j.spasta.2016.03.006. [p102]

A. Gelman, J. B. Carlin, H. S. Stern, D. B. Dunson, A. Vehtari, and D. B. Rubin. *Bayesian Data Analysis*. Chapman & Hall/CRC Texts in Statistical Science (Book 106). Chapman and Hall/CRC, 3 edition, 2013. [p102, 103]

F. Heße, A. Comunian, and S. Attinger. What we talk about when we talk about uncertainty. Toward a unified, data-driven framework for uncertainty characterization in hydrogeology. *Frontiers in Earth Science*, 2019a. doi: 10.3389/feart.2019.00118. [p101]

F. Heße, K. Cucchi, and N. Kawa. Geostat-bayesian/exprior: First release, Nov. 2019b. URL https://doi.org/10.5281/zenodo.3544517. [p101]

F. Heße, K. Cucchi, and N. Kawa. Geostat-bayesian/geostatdb: First release, Oct. 2019c. URL https://doi.org/10.5281/zenodo.3473022. [p102, 106]

R. J. Hoeksema and P. K. Kitanidis. Analysis of the spatial structure of properties of selected aquifers. *Water Resources Research*, 21(4):563–572, 1985. ISSN 1944-7973. doi: 10.1029/WR021i004p00563. [p103]

S. H. Hurlbert. Pseudoreplication and the design of ecological field experiments. *Ecological Monographs*, 54(2):187–211, 1984. ISSN 00129615. URL http://www.jstor.org/stable/1942661. [p111]

N. Kawa, K. Cucchi, and F. Heße. Geostat-bayesian/sitesimilarity: First release, Sept. 2020. URL https://doi.org/10.5281/zenodo.4059706. [p102]

P. Kitanidis. *Introduction to Geostatistics: Applications in Hydrogeology*. Cambridge University Press, 2008. [p101, 102]

J. K. Kruschke. *Doing Bayesian Data Analysis: A Tutorial with R and BUGS*. Academic Press, 2010. [p102]

P. Legendre. Spatial Autocorrelation: Trouble or New Paradigm? *Ecology*, 74(6):1659–1673, Sep 1993. ISSN 0012-9658. doi: {10.2307/1939924}. [p111]

F. Lindgren and H. Rue. Bayesian Spatial Modelling with R-INLA. *Journal of Statistical Software*, 63(19): 1–25, Jan 2015. ISSN 1548-7660. doi: 10.18637/jss.v063.i19. [p101]

D. Lunn, D. Spiegelhalter, A. Thomas, and N. Best. The bugs project: Evolution, critique and future directions. *Statistics in Medicine*, 28(25):3049–3067, 2009. ISSN 1097-0258. doi: 10.1002/sim.3680. [p101]

MINES ParisTech / ARMINES. RGeostats: The Geostatistical R Package. Free download from: http://cg.ensmp.fr/rgeostats, 2019. [p101]

A. Papritz, C. Schwierz, and M. Papritz. georob. https://CRAN.R-project.org/package=georob, 2014. [p101]

E. J. Pebesma. Multivariable geostatistics in S: the gstat package. *Computers & Geosciences*, 30(7):683 – 691, 2004. ISSN 0098-3004. doi: 10.1016/j.cageo.2004.03.012. [p101, 102]

M. J. Pyrcz and C. V. Deutsch. *Geostatistical Reservoir Modeling*. Oxford University Press, 2002. [p101, 102, 110]

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2014. URL http://www.R-project.org/. [p101]

C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006. [p102]

P. J. Ribeiro and P. J. Diggle. geoR: a package for geostatistical analysis. *R-News*, 1:15–18, 2001. [p101]

M. D. Risser and D. Turek. Bayesian inference for high-dimensional nonstationary gaussian processes, 2020. [p101]

Y. Rubin. *Applied Stochastic Hydrogeology*. Oxford University Press, USA, 2003. [p101, 102, 110]

Y. Rubin, X. Chen, H. Murakami, and M. Hahn. A Bayesian approach for inverse modeling, data assimilation, and conditional simulation of spatial random fields. *Water Resources Research*, 46, Oct 6 2010. ISSN 0043-1397. doi: {10.1029/2009WR008799}. [p101]

Y. Rubin, C.-F. Chang, J. Chen, K. Cucchi, B. Harken, F. Heße, and H. Savoy. Stochastic hydrogeology's biggest hurdles analyzed and its big blind spot. *Hydrology and Earth System Sciences Discussions*, 2018:1–36, 2018. doi: 10.5194/hess-2018-290. [p101]

H. Savoy, F. Heße, and Y. Rubin. anchoredDistr: a Package for the Bayesian Inversion of Geostatistical Parameters with Multi-type and Multi-scale Data. *The R Journal*, 9(2):6–17, 2017. doi: 10.32614/RJ-2017-034. [p101]

L. J. Slater, G. Thirel, S. Harrigan, O. Delaigue, A. Hurley, A. Khouakhi, I. Prosdocimi, C. Vitolo, and K. Smith. Using R in hydrology: a review of recent developments and future directions. *Hydrology and Earth System Sciences*, 23(7):2939–2963, 2019. doi: 10.5194/hess-23-2939-2019. URL https://www.hydrol-earth-syst-sci.net/23/2939/2019/. [p101]

J. Vrugt, C. ter Braak, H. Gupta, and B. Robinson. Equifinality of formal (dream) and informal (glue) bayesian approaches in hydrologic modeling? *Stochastic Environmental Research and Risk Assessment*, 23(7):1011–1026, 2009. ISSN 1436-3240. doi: 10.1007/s00477-008-0274-y. [p101]

Falk Heße
Institute of Earth and Environmental Sciences
University Potsdam
Potsdam, Germany
ORCiD: 0000-0002-2547-8102
falk.hesse@ufz.de

Karina Cucchi
Department of Civil and Environmental Engineering
University of California, Berkeley
Berkeley, CA, USA
karina.cucchi@berkeley.edu

Nura Kawa
Department of Statistics
University of California, Berkeley
Berkeley, CA, USA
currently at:
Leuven Statistics Research Centre,
KU Leuven,
Leuven, Belgium
nkawa@berkeley.edu

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

penPHcure: Variable Selection in Proportional Hazards Cure Model with Time-Varying Covariates

by Alessandro Beretta and Cédric Heuchenne

Abstract We describe the **penPHcure** R package, which implements the semiparametric proportional-hazards (PH) cure model of Sy and Taylor (2000) extended to time-varying covariates and the variable selection technique based on its SCAD-penalized likelihood proposed by Beretta and Heuchenne (2019a). In survival analysis, cure models are a useful tool when a fraction of the population is likely to be immune from the event of interest. They can separate the effects of certain factors on the probability of being susceptible and on the time until the occurrence of the event. Moreover, the **penPHcure** package allows the user to simulate data from a PH cure model, where the event-times are generated on a continuous scale from a piecewise exponential distribution conditional on time-varying covariates, with a method similar to Hendry (2014). We present the results of a simulation study to assess the finite sample performance of the methodology and illustrate the functionalities of the **penPHcure** package using criminal recidivism data.

Introduction

In contrast to other statistical methods, survival analysis models are designed to model the time to an event of interest (e.g., death or occurrence of a disease in medical studies). A typical feature of time-to-event data is the presence of right censoring, an incomplete information problem that arises when a subject is lost to follow-up or does not experience the event before the end of the study. In these cases, it is unknown whether the subject will eventually experience the event and when it will occur, given that it can occur. The most common assumption of standard survival analysis models is that the whole population will sooner or later experience the event of interest. However, in practice, this may not be the case because a fraction of the population may be immune (i.e., not susceptible) to this event. *Cure models*, also known as *split population duration models* or *limited-failure population models*, were developed to handle this kind of situation. They allow us to investigate the effects of some covariates (e.g., type of treatment, stage of the tumor, sex, or age) on the probability to be susceptible to the event of interest (i.e., incidence), and on the survival time conditional on being susceptible (i.e., latency).

Originally, cure models were introduced in the medical literature by Boag (1949) and Berkson and Gage (1952), but they have been used in several other disciplines during the years. In reliability engineering, Meeker (1987) investigates the failure of solid-state electronic components (e.g., integrated circuits). In social science, Schmidt and Witte (1989) investigate the timing of return to prison for a sample of prison releases, and they use it to make predictions of whether or not individuals return to prison. In finance, Cole and Gunther (1995) analyze the determinants of commercial bank failures in the United States; in credit scoring, Tong et al. (2012) predict defaults on a portfolio of UK personal loans. In political science, Svolik (2008) studies the likelihood that a democracy consolidates and the timing of authoritarian reversals in democracies that are not consolidated. In marketing, Polo et al. (2011) investigate the drivers of customer retention in a liberalizing market, using data for a sample of 650 consumers in the Spanish mobile phone industry. In the literature, several variants of cure models have been proposed (see Amico and Van Keilegom (2018) for a comprehensive survey), which belong to two main families: mixture cure models and promotion time cure models.

In this article, we present the **penPHcure** package (Beretta and Heuchenne, 2019b), which implements the semiparametric proportional-hazards (PH) mixture cure model of Sy and Taylor (2000) extended to time-varying covariates, where the incidence and latency distributions are modeled by a logistic regression and a Cox's PH model (Cox, 1972), respectively. The **penPHcure** package contains two main functions: `penphcure`, to estimate the regression coefficients, their confidence intervals using the basic/percentile bootstrap method, and to perform variable selection using the SCAD-penalized likelihood technique proposed by Beretta and Heuchenne (2019a); and `penphcure.simulate` to simulate data from a PH cure model, where the event-times are generated on a continuous scale from a piecewise exponential distribution conditional on time-varying covariates, using a method similar to the one described in Hendry (2014).

At the time of writing this article, we are unaware of other R packages for estimation of semiparametric PH mixture cure models with time-varying covariates and, above all, that enable the user to perform variable selection. In the context of cure models for right-censored data, available

R packages include: the **flexsurvcure** package (Amdahl, 2019) for estimation of parametric mixture and non-mixture cure models with time-invariant covariates using time-to-event distributions from the **flexsurv** package (Jackson, 2016); the **nltm** package (Garibotti et al., 2019) for estimation of the semiparametric PH cure model with time-invariant covariates of Tsodikov et al. (2003), as well as other nonlinear transformation models for analyzing survival data using the method of Tsodikov (2003); and the **smcure** package (Cai et al., 2012) for estimation of the semiparametric PH cure model and the accelerated failure time cure model with time-invariant covariates and the **spduration** package (Beger et al., 2018) that implements a parametric cure model with time-varying covariates using Weibull and Log-Logistic latency distributions. Compared to `spduration`, the **penPHcure** package has some advantages: the latency distribution is modeled by a more flexible semiparametric Cox's PH model; the response variable and the time to the event of interest are continuous; and, above all, it allows the user to simultaneously select variables and estimate their parameters using a variable selection technique based on SCAD penalties.

The remainder of this article is structured as follows:

- *Methodology*. We present the PH cure model with time-varying covariates implemented in the `penPHcure` function when the argument `pen.type` is set equal to `"none"` (default);
  - *Variable selection*. We present the variable selection technique based on SCAD penalties implemented in the `penPHcure` function when `pen.type=="SCAD"`;
  - *Data generation*. We describe the algorithm implemented in the `penPHcure.simulate` function, which generates data from a PH cure model with time-varying covariates.
- *Simulation study*. We analyze the finite sample performance of the PH cure model estimates and its variable selection technique implemented in the `penPHcure` function.
- *An application to Criminal Recidivism data*. We provide an example of practical use of the `penPHcure` function to analyze a real data set.

Methodology

Let Y be a Bernoulli random variable indicating whether an individual is susceptible ($Y = 1$) or immune ($Y = 0$) to the event of interest with probability $p = P(Y = 1)$. Let T be the time to event, defined only when $Y = 1$. Assuming that a fraction of the population is immune to the event of interest, the marginal survival function of T is defined as

$$S(t) = (1 - p) + pS(t|Y = 1),$$

where p is the incidence (i.e., probability of being susceptible) and $S(t|Y = 1)$ is the latency (i.e., survival function conditional on being susceptible).

The incidence is modeled by a logistic regression model:

$$p(\mathbf{x}) = P(Y = 1|\mathbf{x}) = \frac{\exp(\mathbf{x}'\mathbf{b})}{1 + \exp(\mathbf{x}'\mathbf{b})},$$

where $\mathbf{x}$ is a vector of time-fixed covariates (including the intercept) and $\mathbf{b}$ a vector of unknown coefficients. Whereas the latency is modeled by a Cox's PH model:

$$S(t|Y = 1, \bar{\mathbf{z}}(t)) = \exp\left(-\int_0^t h_0(u)e^{\mathbf{z}'(u)\boldsymbol{\beta}}du\right),$$

where $\mathbf{z}(t)$ is a vector of time-varying covariates (we denote by $\bar{\mathbf{z}}(t)$ the full history of the covariates up to time t), $\boldsymbol{\beta}$ is a vector of unknown coefficients, and $h_0(t)$ is an arbitrary baseline conditional hazard function.

Let $\mathbf{O} = \{(t_i, \delta_i, \bar{\mathbf{z}}_i(t_i), \mathbf{x}_i); i = 1, ..., n\}$ denote the observed data, where t_i is the event/censoring time and δ_i is the censoring indicator, which takes value 1 if t_i is uncensored and 0 otherwise. Since we know that $y_i = 1$ when $\delta_i = 1$, but y_i is unobserved when $\delta_i = 0$, we can estimate the unknown parameters $\boldsymbol{\theta} = (\mathbf{b}, \boldsymbol{\beta}, h_0)$ using the Expectation-Maximization (EM) algorithm (Dempster et al., 1977). The complete-data likelihood can be written as

$$L_C(\mathbf{b}, \boldsymbol{\beta}, h_0) = \underbrace{\prod_{i=1}^{n} p(\mathbf{x}_i)^{y_i} [1 - p(\mathbf{x}_i)]^{(1-y_i)}}_{L_1(\mathbf{b})} \times \underbrace{\prod_{i=1}^{n} \left[h_0(t_i)e^{\mathbf{z}_i'(t_i)\boldsymbol{\beta}}\right]^{\delta_i y_i} \left[e^{-\int_0^{t_i} h_0(u)e^{\mathbf{z}_i'(u)\boldsymbol{\beta}}\, du}\right]^{y_i}}_{L_2(\boldsymbol{\beta}, h_0)}, \quad (1)$$

i.e., the product between the incidence component L_1 depending on a set of time-fixed covariates $\mathbf{x}_i$, and the latency component L_2 depending on a set of time-varying covariates $\mathbf{z}_i'(t)$.

Given some starting values $\boldsymbol{\theta}^{(0)}$, the m-th iteration of the EM algorithm consists of two steps:

E step. Compute the expectation of the complete-data likelihood with respect to the conditional distribution of the y_i's given the current parameter estimates $\hat{\boldsymbol{\theta}}^{(m-1)}$ and the observed data $\mathbf{O}$. This expectation is obtained by replacing the y_i's in (1) by their expectation

$$\pi_i^{(m)} = E\left[Y_i \left| \hat{\boldsymbol{\theta}}^{(m-1)}, \mathbf{O}\right.\right] = \delta_i + (1-\delta_i)\frac{p(\mathbf{x}_i)S(t|Y=1,\bar{\mathbf{z}}_i(t))}{1-p(\mathbf{x}_i)+p(\mathbf{x}_i)S(t|Y=1,\bar{\mathbf{z}}_i(t))}.$$

Note that we removed the dependence of the theoretical functions on the estimated parameters to simplify the notation.

M step. Maximize the expected complete-data likelihood with respect to $\mathbf{b}$, $\boldsymbol{\beta}$, and the function h_0. Given $\boldsymbol{\pi}^{(m)} = \{\pi_1^{(m)}, ..., \pi_n^{(m)}\}$, the incidence component L_1 in (1) is maximized using the Newton-Raphson method as in the classical logistic regression model. Whereas the latency component L_2 in (1) is maximized using a profile likelihood approach. The latter involves two steps: (i) the baseline conditional hazard function is estimated nonparametrically by

$$\hat{h}_0(t) = \frac{1}{\left(t_{(j)} - t_{(j-1)}\right)\sum_{i\in R_j}\pi_i^{(m)}e^{\mathbf{z}_i'(t_{(j)})\boldsymbol{\beta}}}, \quad \text{for } t \in (t_{(j-1)}, t_{(j)}], \tag{2}$$

where $t_{(1)} \leq ... \leq t_{(k)}$ are the k ordered event times and R_j is the risk set at $t_{(j)}^-$ (i.e., the set of all individuals who did not experience the event of interest and have not been censored just prior to time $t_{(j)}$); and then (ii) the function h_0 in L_2 is replaced by its estimator given in (2) to obtain the following partial likelihood, which does not depend on the function h_0 anymore,

$$\tilde{L}_2\left(\boldsymbol{\beta}\left|\boldsymbol{\pi}_i^{(m)}\right.\right) = \prod_{j=1}^{k}\frac{e^{\mathbf{z}_i(t_{(j)})\boldsymbol{\beta}}}{\sum_{i\in R_j}\pi_i^{(m)}e^{\mathbf{z}_i'(t_{(j)})\boldsymbol{\beta}}}. \tag{3}$$

Finally, the latency component is estimated by maximizing (3) with respect to $\boldsymbol{\beta}$. In case of tied event-times, (2) and (3) can be rewritten using the Breslow (1974) or Efron (1977) approximation as in the standard Cox's PH model.

The EM algorithm terminates whenever $\left\|\hat{\mathbf{b}}^{(m)} - \hat{\mathbf{b}}^{(m-1)}\right\|_2 < \epsilon$ and $\left\|\hat{\boldsymbol{\beta}}^{(m)} - \hat{\boldsymbol{\beta}}^{(m-1)}\right\|_2 < \epsilon$, where ϵ is a tolerance threshold (by default 10^{-6}).

Variable selection

When the number of available covariates is large, fitting all possible subsets to find the most relevant covariates would be too time consuming. Beretta and Heuchenne (2019a) proposed a regularization method based on the maximization of a penalized version of the complete-data log-likelihood

$$\ell_C^p(\boldsymbol{\theta};\lambda_1,\lambda_2) = \underbrace{\ell_1(\mathbf{b}) - n\sum_{j=2}^{q_1+1}p_{\lambda_1}\left(|b_j|\right)}_{\ell_1^p(\mathbf{b};\lambda_1)} + \underbrace{\ell_2(\boldsymbol{\beta},\mathbf{h}_0) - n\sum_{l=1}^{q_2}p_{\lambda_2}(|\beta_l|)}_{\ell_2^p(\boldsymbol{\beta},\mathbf{h}_0;\lambda_2)},$$

where ℓ denotes a log-likelihood and $p_\lambda(.)$ a SCAD penalty function, which role is to shrink the small coefficients toward zero. We assume that the q_1 and q_2 covariates in the incidence and latency component, respectively, have been standardized, such that the coefficients in $\mathbf{b}$ and $\boldsymbol{\beta}$ are on the same scale. The Smoothly Clipped Absolute Deviation (SCAD) penalty function (Fan and Li, 2001) is given by

$$p_\lambda(|\beta_j|) = \begin{cases} \lambda|\beta_j|, & \text{if } |\beta_j| \leq \lambda \\ \frac{(a^2-1)\lambda^2-(|\beta_j|-a\lambda)^2}{2(a-1)}, & \text{if } \lambda < |\beta_j| \leq a\lambda \\ \frac{(a+1)\lambda^2}{2}, & \text{if } |\beta_j| > a\lambda \end{cases},$$

for some $a > 2$ and $\lambda > 0$. As explained by Fan and Li (2001, 2002) in the context of linear regression, generalized linear models, Cox's PH model, and frailty models, the SCAD estimator has three desirable properties: unbiasedness (do not penalize to much large coefficients), sparsity, and continuity. Moreover, with a proper choice of the tuning parameters (λ, a), it also possesses what is known as the "oracle property", meaning that the SCAD estimator is asymptotically equivalent to the oracle

estimator (i.e., an estimator with only the relevant variables with nonzero coefficients).

For fixed values of the tuning parameters $(\lambda_1, \lambda_2, a_1, a_2)$, estimates of $\boldsymbol{\theta}$ can be obtained using an EM algorithm close to the one described in the previous section. The only difference lies in the M-step. Given that the penalized log-likelihoods $\ell_1^P(\mathbf{b}; \lambda_1)$ and $\tilde{\ell}_2^P(\boldsymbol{\beta}; \lambda_2)$, where $\tilde{\ell}_2^P(\boldsymbol{\beta}; \lambda_2)$ is the logarithm of (3), are non-concave and non-differentiable at the origin, $\mathbf{b}$ and $\boldsymbol{\beta}$ are now estimated using the MM algorithm of Hunter and Li (2005) based on a perturbed Local Quadratic Approximation (LQA) of the penalty function. By default, when the argument `pen.type` of the function `penPHcure` is set equal to `"SCAD"`, the initial values for $\mathbf{b}$ and $\boldsymbol{\beta}$ are vectors with all elements equal to zero. Otherwise, the user can specify other values (e.g., the estimated coefficients of the model with all covariates) using the argument `SV`.

Regarding the choice of the tuning parameters, following the suggestion of Fan and Li (2001), we keep $a_1 = a_2 = 3.7$, and given a set of possible values for (λ_1, λ_2), we select the ones that minimize the following Akaike (AIC) or Bayesian (BIC) Information Criteria:

$$\text{AIC}(\lambda_1, \lambda_2) = -2\ell\left(\hat{\boldsymbol{\theta}}_{\lambda_1,\lambda_2}\right) + 2\nu; \text{ and}$$

$$\text{BIC}(\lambda_1, \lambda_2) = -2\ell\left(\hat{\boldsymbol{\theta}}_{\lambda_1,\lambda_2}\right) + \ln(n)\nu,$$

where $\ell\left(\hat{\boldsymbol{\theta}}_{\lambda_1,\lambda_2}\right)$ is the observed data log-likelihood evaluated at the penalized MLE $\hat{\boldsymbol{\theta}}_{\lambda_1,\lambda_2}$ and ν is the number of nonzero coefficients, identified as the number of coefficients with an absolute value greater than a given threshold (by default 10^{-6}).

Data generation

Let $\mathbf{S} = \{s_1, s_2, ..., s_J\}$ be a partition of the time scale forming $J+1$ intervals $(0, s_1]$, $(s_1, s_2]$, ..., $(s_{J-1}, s_J]$, $(s_J, \infty]$. Define a vector of time-varying covariates piecewise constant in each interval: $\mathbf{z}(t) = \mathbf{z}_j$, for $t \in (s_{j-1}, s_j]$. Consider a transformation g, such that $g(0) = 0$, $g(t)$ is strictly increasing for $t > 0$, and $g^{-1}(t)$ is differentiable. In the implementation of the `penPHcure.simulate` function, we use $g(t) = t^{1/\gamma}$, where the parameter γ can be specified by the user via the argument `gamma`, which by default is equal to 1. According to Hendry (2014), if we generate a random variable V as a piecewise exponential distribution with density function given by

$$f_V(t) = \prod_{l=1}^{j-1} \exp\{-\lambda_l[g^{-1}(s_l) - g^{-1}(s_{l-1})]\}\lambda_j \exp\{-\lambda_j[t - g^{-1}(s_{j-1})]\}, \text{ for } t \in (g^{-1}(s_{j-1}), g^{-1}(s_j)],$$

where $\lambda_j = \exp(\mathbf{z}_j'\boldsymbol{\beta})$ is the constant hazard in the interval $(g^{-1}(s_{j-1}), g^{-1}(s_j)]$, then $g(V)$ follows a Cox's PH model with time-varying covariates with a baseline hazard function given by $h_0(t) = \frac{d}{dt}[g^{-1}(t)]$. This method is part of the algorithm implemented in the `penPHcure.simulate` function to simulate data from a PH cure model with time-varying covariates (see Table 1 for a detailed description).

Require: N, sample size; $\mathbf{S}$, partition of the time scale; $g(t)$, variable transformation; $\mathbf{b}$, incidence coefficients; $\boldsymbol{\beta}$, latency coefficients.

for $i = 1, ..., N$ **do**

1. Generate a vector $\mathbf{x}_i$ from an arbitrary distribution;
2. Generate y_i from a Bernoulli distribution with probability $p(\mathbf{x}_i)$;
3. Generate $\mathbf{z}_i = \{\mathbf{z}_{i,1}, \mathbf{z}_{i,2}, ...\mathbf{z}_{i,J}\}$ from an arbitrary distribution;
4. Generate v_i from a piecewise exponential distribution with density $f_V(t)$;
5. Compute $w_i = g(v_i)$;
6. Generate c_i from an arbitrary distribution;

if $y_i = 0$ **or** $w_i > c_i$ **then**

$t_i = c_i$;
$\delta_i = 0$;

else

$t_i = w_i$;
$\delta_i = 1$;

end if

end for

return $\{(t_i, \delta_i, \mathbf{z}_i, \mathbf{x}_i); i = 1, ..., N\}$

Table 1: Data generation algorithm: PH cure model with time-varying covariates

Simulation study

In this section, we present the results of a simulation study conducted to assess the finite sample performance of the PH cure model estimation and its variable selection technique implemented in the `penPHcure` function. The event-times follow a Cox's PH model with baseline hazard function $h_0(t) = 3t^2$ and 8 time-varying covariates. These covariates are constant within $J = 30$ equally-spaced intervals $(0, s_1]$, $(s_1, s_2]$, ..., $(s_{J-1}, s_J]$, where $s_1 = 0.2$ and $s_J = 6$. They follow a multivariate normal distribution $\mathbf{z}_{i,j} \sim N(\mathbf{0}, \boldsymbol{\Sigma})$, where $\Sigma_{p,q} = 0.5^{|p-q|}$, for $p, q = 1, ..., 8$. The censoring times follow an exponential distribution truncated above 6 and with parameter λ_C. The cure indicators are generated from a logistic regression model with 8 time-fixed covariates that follow a multivariate normal distribution $\mathbf{x}_i \sim N(\mathbf{0}, \boldsymbol{\Sigma})$, where $\Sigma_{p,q} = 0.5^{|p-q|}$, for $p, q = 1, ..., 8$. Finally, the regression coefficients vectors are set equal to $\boldsymbol{\beta}_0 = (-0.7, 0, 1, 0, -0.5, 0.75, 0, 0)'$ and $\mathbf{b}_0 = (b_0, 1.5, 0, -0.75, 0, -1.5, 0, 0.75, 0)'$.

We consider 6 simulation settings, with different levels of censoring and proportions of non-susceptible individuals (expressed as a fraction of the sample size), depending on different values of b_0 and λ_C (see Table 2). For each of these settings, we generated 500 replications using the `penPHcure.simulate` function for different sample sizes $N = \{250, 500, 1000\}$. Then, for all simulated datasets, we use the `penPHcure` function to (i) fit a standard PH cure model with all covariates (FULL), (ii) fit a standard PH cure model with the covariates associated to the non-zero coefficients only (ORACLE), and (iii) to perform variable selection using the regularization method with SCAD penalties and tuning parameters chosen according to the BIC criterion. The possible values of the tuning parameters (λ_1, λ_2) are obtained with the function `exp(seq(-6,-1,length.out = 10))`, whereas (a_1, a_2) are kept equal to 3.7. Furthermore, we use the `coxph` function in the **survival** package (Therneau, 2015) to fit the classical Cox's PH model with the covariates associated to the non-zero coefficients only (COX).

Censoring	Cure	λ_C	b_0
Low (40%)	High (30%)	0.02	1.45
Low (40%)	Medium (20%)	0.3	2.35
Medium (60%)	High (45%)	0.35	0.35
Medium (60%)	Medium (30%)	0.75	1.45
High (80%)	High (60%)	0.95	-0.7
High (80%)	Medium (40%)	1.55	0.7

Table 2: Simulation settings (censoring and cure are expressed as fractions of all individuals).

The performance is measured in terms of Mean Estimation Error (MEE) and average number of correct and incorrect zeros identified by the variable selection technique (SCAD). In particular, the estimation error for the incidence component is computed as $E\left[(\hat{p}(\mathbf{x}) - p_0(\mathbf{x}))^2\right]$, where $\hat{p}(\mathbf{x})$ and $p_0(\mathbf{x})$ are the estimated and true probabilities of being susceptible. Whereas the estimation error for the latency component is computed as $E\left[(\hat{S}(T|Y=1) - S_0(T|Y=1))^2\right]$, where $\hat{S}(T|Y=1)$ and $S_0(T|Y=1)$ are the estimated and true survival functions conditional on being susceptible.

In Figures 1 and 2, we provide the MEEs for the incidence and latency components, respectively, while in Figure 3, we provide the average number of correct and incorrect zeros. From those figures, we can see that the PH cure model estimation and its variable selection technique implemented in the `penPHcure` function perform reasonably well. For an increase of the sample size or a decrease of the level of censoring, the MEE decreases, and the number of correct (resp. incorrect) zeros converges to 4 (resp. 0). The MEEs of the ORACLE model are always the lowest ones, but we notice that the ones of the SCAD method tend towards them as the sample size increases. It is important to note that, for a fixed level of censoring, we observe higher MEEs in the case of a lower fraction of cured individuals. The worst results are obtained in situations of high censoring and low cure rates, but it is enough to increase the sample size to obtain better results. This is evidence of the fact that a cure model should always be applied to data with a sufficient number of non-susceptible individuals. Last but not the least important, we note that the use of the classical Cox's PH model (COX) leads to very high errors. This was expected since the model is wrongly specified as it ignores the existence of cured subjects.

Finally, in Table 3, we also present the coverage probabilities of the estimated 95% confidence intervals for the ORACLE model using the basic and percentile bootstrap methods with 500 resamples. In most cases, the basic bootstrap method outperforms the percentile bootstrap method, especially for the smallest sample sizes, with coverage probabilities closer to the 95% nominal level.

The R code used to obtain the results in Figures 1 to 3 (resp. Table 3) are provided in Section 2 (resp. Section 3) of the supplementary material ('beretta-heuchenne.R'). Moreover, in the file 'beretta-heuchenne-suppl.pdf', we provide a table with all the results contained in Figures 1 to 3.

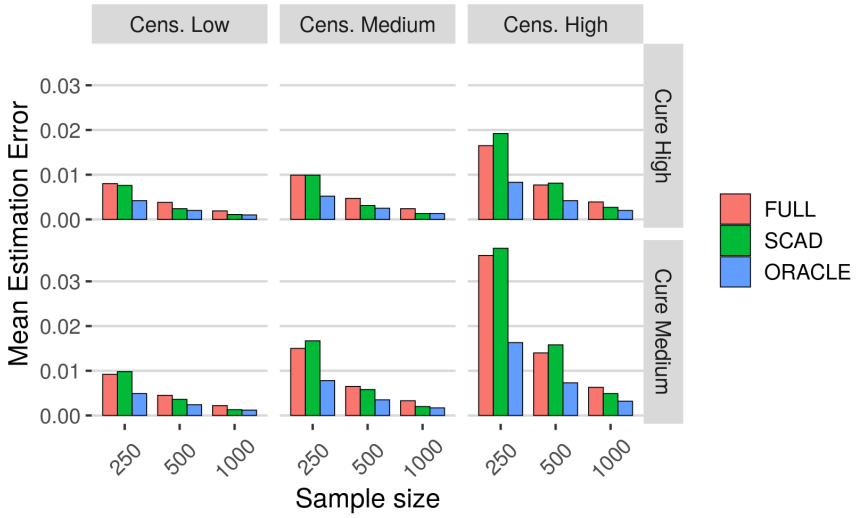


Figure 1: Results of the simulations: mean estimation errors (incidence component).

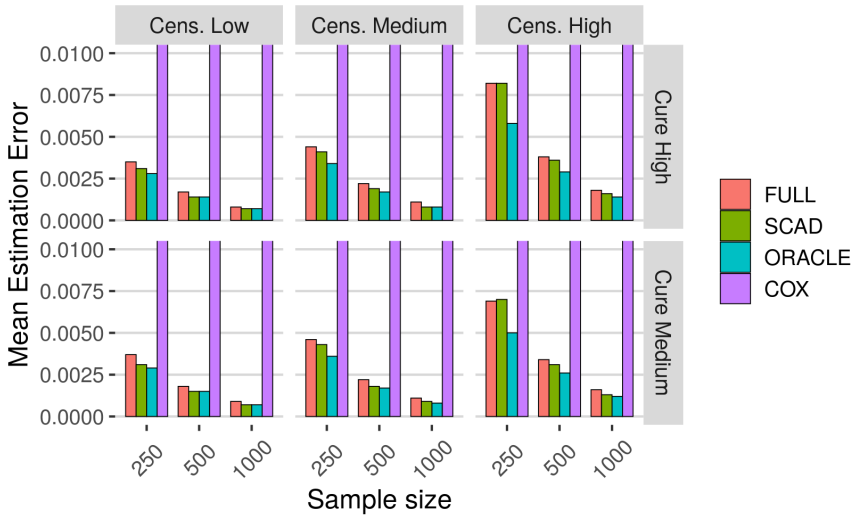


Figure 2: Results of the simulations: mean estimation errors (latency component).

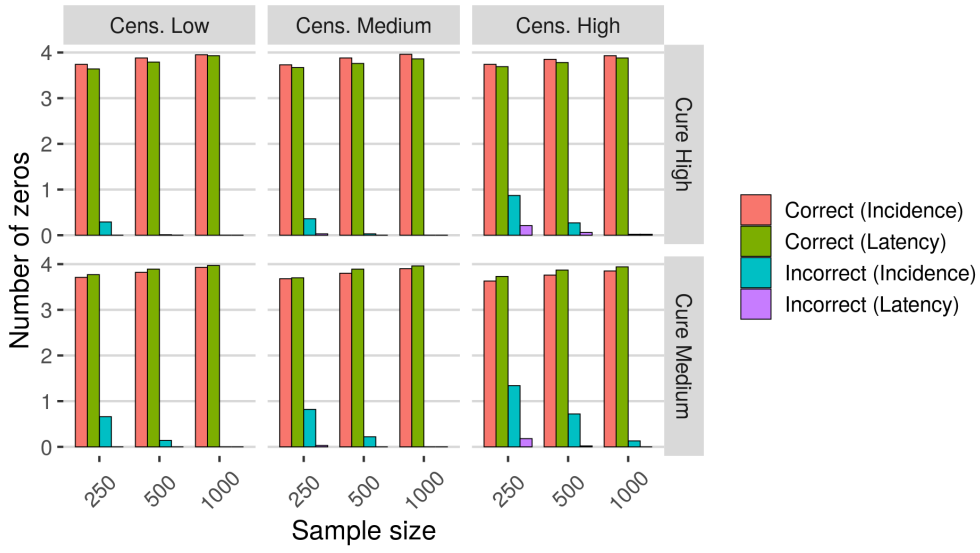


Figure 3: Results of the simulations: average number of correct/incorrect zeros identified by the variable selection technique (SCAD).

cens	cure	N	Method	b_0	b_1	b_2	b_3	b_4	β_1	β_2	β_3	β_4
0.40	0.30	250	basic	0.97	0.97	0.98	0.97	0.97	0.96	0.97	0.96	0.94
			perc	0.9	0.91	0.94	0.91	0.94	0.93	0.94	0.95	0.94
		500	basic	0.97	0.95	0.95	0.94	0.96	0.96	0.96	0.96	0.96
			perc	0.91	0.91	0.94	0.92	0.94	0.94	0.91	0.95	0.95
		1000	basic	0.95	0.96	0.95	0.97	0.97	0.93	0.95	0.95	0.96
			perc	0.94	0.94	0.93	0.95	0.97	0.93	0.95	0.95	0.95
0.40	0.20	250	basic	0.95	0.96	0.99	0.96	0.97	0.96	0.97	0.97	0.96
			perc	0.89	0.9	0.94	0.91	0.93	0.94	0.96	0.96	0.96
		500	basic	0.98	0.97	0.97	0.97	0.98	0.95	0.96	0.96	0.96
			perc	0.93	0.94	0.94	0.94	0.94	0.94	0.94	0.93	0.93
		1000	basic	0.97	0.96	0.96	0.95	0.95	0.94	0.93	0.96	0.95
			perc	0.95	0.94	0.94	0.93	0.94	0.94	0.94	0.96	0.95
0.60	0.45	250	basic	0.98	0.95	0.96	0.96	0.97	0.98	0.98	0.97	0.97
			perc	0.95	0.91	0.93	0.93	0.94	0.93	0.94	0.94	0.94
		500	basic	0.98	0.95	0.94	0.97	0.96	0.94	0.94	0.96	0.95
			perc	0.96	0.93	0.94	0.92	0.93	0.94	0.92	0.95	0.95
		1000	basic	0.95	0.96	0.96	0.95	0.95	0.95	0.94	0.93	0.95
			perc	0.95	0.94	0.94	0.93	0.94	0.94	0.92	0.94	0.93
0.60	0.30	250	basic	0.96	0.94	0.98	0.94	0.99	0.97	0.96	0.96	0.96
			perc	0.89	0.9	0.91	0.89	0.93	0.94	0.94	0.95	0.95
		500	basic	0.96	0.97	0.98	0.95	0.97	0.96	0.96	0.95	0.95
			perc	0.94	0.91	0.94	0.93	0.93	0.95	0.96	0.95	0.95
		1000	basic	0.96	0.96	0.96	0.95	0.96	0.96	0.95	0.95	0.96
			perc	0.95	0.94	0.94	0.92	0.94	0.94	0.94	0.94	0.95
0.80	0.60	250	basic	0.99	0.97	0.98	0.97	0.97	0.97	0.96	0.97	0.97
			perc	0.94	0.91	0.92	0.91	0.92	0.94	0.92	0.96	0.94
		500	basic	0.98	0.95	0.97	0.95	0.97	0.97	0.96	0.98	0.98
			perc	0.94	0.93	0.95	0.93	0.94	0.95	0.94	0.95	0.96
		1000	basic	0.96	0.95	0.96	0.94	0.96	0.96	0.95	0.94	0.96
			perc	0.93	0.95	0.94	0.93	0.95	0.95	0.94	0.93	0.93
0.80	0.40	250	basic	0.98	0.95	0.99	0.98	1	0.97	0.96	0.99	0.99
			perc	0.92	0.88	0.91	0.91	0.93	0.95	0.93	0.97	0.95
		500	basic	0.97	0.95	0.97	0.96	0.98	0.96	0.96	0.97	0.96
			perc	0.94	0.9	0.93	0.91	0.93	0.94	0.94	0.96	0.94
		1000	basic	0.96	0.96	0.97	0.95	0.95	0.98	0.95	0.93	0.96
			perc	0.96	0.95	0.95	0.94	0.93	0.94	0.94	0.93	0.93

Table 3: Results of the simulations: coverage probabilities.

An application to Criminal Recidivism data

In this section, we illustrate the use of the **penPHcure** R package using a Criminal Recidivism dataset, which contains a sample of 432 inmates released from Maryland state prisons and followed for one year after release (Rossi et al., 1980). The aim of this study was to investigate the relationship between the time to first arrest after release and some covariates observed during the follow-up period. In particular, to study the effect of providing financial aid at the moment of release. The original source of the data is the `Rossi` dataset in the **RcmdrPlugin.survival** package (Fox and Carvalho, 2012), which has been converted into a counting process format and included in the **penPHcure** package.

Let us load and illustrate the dataset:

```
> library(penPHcure)
> data("cpRossi", package = "penPHcure")
> head(cpRossi)
  id tstart tstop arrest fin age  race wexp mar paro prio educ emp
1  1      0    20    yes  no  27 black   no  no  yes    3    3  no
2  2      0     9     no  no  18 black   no  no  yes    8    4  no
3  2      9    14     no  no  18 black   no  no  yes    8    4 yes
4  2     14    17    yes  no  18 black   no  no  yes    8    4  no
5  3      0    16     no  no  19 other  yes  no  yes   13    3  no
6  3     16    17     no  no  19 other  yes  no  yes   13    3 yes
> str(cpRossi)
'data.frame': 1405 obs. of  13 variables:
 $ id    : int  1 2 2 2 3 3 3 4 4 4 ...
```

```
 $ tstart: int  0 0 9 14 0 16 17 0 4 21 ...
 $ tstop : int  20 9 14 17 16 17 25 4 21 31 ...
 $ arrest: Factor w/ 2 levels "no","yes": 2 1 1 2 1 1 2 1 1 1 ...
 $ fin   : Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 2 2 2 ...
 $ age   : int  27 18 18 18 19 19 19 23 23 23 ...
 $ race  : Factor w/ 2 levels "black","other": 1 1 1 1 2 2 2 1 1 1 ...
 $ wexp  : Factor w/ 2 levels "no","yes": 1 1 1 1 2 2 2 2 2 2 ...
 $ mar   : Factor w/ 2 levels "yes","no": 2 2 2 2 2 2 2 1 1 1 ...
 $ paro  : Factor w/ 2 levels "no","yes": 2 2 2 2 2 2 2 2 2 2 ...
 $ prio  : int  3 8 8 8 13 13 13 1 1 1 ...
 $ educ  : Factor w/ 3 levels "3","4","5": 1 2 2 2 1 1 1 3 3 3 ...
 $ emp   : Factor w/ 2 levels "no","yes": 1 1 2 1 1 2 1 1 2 1 ...
```

The object `cpRossi` is a `data.frame` in counting process format with 1405 observations for 432 individuals on 13 variables. The `id` variable provides the unique identification number for every individual in the study. The variables `tstart` and `tstop` denote the time interval of the observation (measured in weeks). The variable `arrest` denotes whether the individual has been arrested during the 1-year follow-up period. The remaining explanatory variables are described hereafter.

- `fin`. Financial aid received after release: `yes` or `no`;
- `age`. Age in years at the time of release;
- `race`. Race of the individual: `black` or `other`;
- `wexp`. Full-time work experience before incarceration: `yes` or `no`;
- `mar`. Married at the time of release: `yes` or `no`;
- `paro`. Released on parole: `yes` or `no`;
- `prio`. Number of convictions prior to incarceration;
- `educ`. Level of education: <=9th degree ("3"), 10th or 11th degree ("4"), or >=12th degree ("5");
- `emp`. Working full time during the observed time interval: `yes` or `no`. This is the only variable which is varying over time (e.g., the individual with `id = 2` did not work full time during the first 9 weeks after release, then he did for 5 weeks, and, finally, he has been arrested after 3 weeks without working full time).

Using the `penPHcure` function, by default, we can fit the standard PH cure model. First, we use a formula object with the response on the left of the tilde operator and the explanatory variables to be included in the latency component on the right. The response is a survival object returned by the `Surv(tstart,tstop,arrest)` function. Then, using the argument `cureform`, we specify the explanatory variables to be included in the incidence component. By default, these covariates are set equal to the last observation, but in this case, we set the argument `which.X = "mean"` to compute the time-weighted average over the full history. Finally, setting the argument `inference = TRUE`, we conduct inference about the parameter estimates via bootstrapping (by default, 100 bootstrap resamples). The user can increase/decrease the number of bootstrap resamples with the argument `nboot`.

```
> set.seed(123) # for reproducibility
> fit <- penPHcure(Surv(tstart,tstop,arrest)~fin+age+race+wexp+mar+paro+prio+educ+emp,
+                  cureform = ~fin+age+race+wexp+mar+paro+prio+educ+emp,
+                  data = cpRossi,which.X = "mean",inference = TRUE)

Initializing PH cure model with time-varying covariates...

Number of individuals: 432
Censoring proportion: 0.736
Tied failure times: TRUE
Number of unique failure times: 49
Number of covariates in the survival component: 10
Number of covariates in the cure component: 10

Checking starting values...

Fitting standard PH cure model with time-varying covariates... Please wait...

Performing inference via bootstrapping... Please wait ...
  |======================================================================| 100%
DONE!
```

This call to the penPHcure function returned an object of class "PHcure", and we can print a summary of the results using the summary method. By default, confidence intervals are computed using the basic bootstrap method (the alternative is percentile bootstrap) and a confidence level of 95%. In order to control these features, the user can provide the arguments conf.int and conf.int.level, respectively.

```
> summary(fit)

-----------------------------------------------------
+++   PH cure model with time-varying covariates   +++
-----------------------------------------------------
Sample size: 432
Censoring proportion: 0.7361111
Number of unique event times: 49
Tied failure times: TRUE

log-likelihood:  -643.65

-----------------------------------------------------
+++     Cure (incidence) coefficient estimates     +++
+++          and 95% confidence intervals *        +++
-----------------------------------------------------
             Estimate       2.5%     97.5%
(Intercept)  1.136709 -34.041743  9.769052
finyes      -0.455199  -2.299870 12.188817
age         -0.067715  -0.382429  0.413001
raceother   -0.100950  -2.988104 35.336024
wexpyes      0.251663  -3.257339  2.193371
marno        0.261947 -15.041406 35.102574
paroyes     -0.041289  -3.156395  1.659637
prio         0.068443  -0.285553  0.237089
educ4       -0.570782  -2.614311  2.532353
educ5       -1.163257 -39.473732 34.360612
empyes      -0.860659  -3.299354  1.216299

-----------------------------------------------------
+++    Survival (latency) coefficient estimates    +++
+++          and 95% confidence intervals *        +++
-----------------------------------------------------
           Estimate      2.5%     97.5%
finyes     0.062630 -1.427436  1.446067
age        0.046192 -0.067209  0.176043
raceother -0.759985 -2.770654  0.720247
wexpyes   -0.552549 -1.672866  0.576657
marno      0.123655 -2.327914  1.600195
paroyes    0.040388 -0.816177  1.110058
prio       0.048407 -0.107942  0.195763
educ4      0.588156 -0.545885  1.881803
educ5      0.838098 -2.527512  5.118107
empyes    -1.431782 -1.980471 -0.781978

-----------------------------------------------------
* Confidence intervals computed by the basic
  bootstrap method, with 100 replications.
-----------------------------------------------------
```

As you can see, only one covariate (emp) in the latency component is statistically significant (the 95% confidence interval does not include zero). The negative sign of the estimated coefficient implies that the individuals working full time after release have a lower risk of being rearrested (among the individuals susceptible to be rearrested). The lack of significance of the other covariates might be explained by the small sample size, the high level of censoring (only 114 out of 432 individuals have been rearrested), or by potential confounding factors.

We now perform variable selection with the proposed SCAD-penalized likelihood method to check whether other covariates may be relevant to explain incidence and latency. First, we specify the possible values of the tuning parameters (using the argument pen.tuneGrid) and set the starting

values equal to the coefficient estimates from the unpenalized model (using the argument SV). Then, we still use the penPHcure function, but we now include the argument pen.type = "SCAD".

```
> pen.tuneGrid <- list(CURE = list(lambda = seq(0.01,0.12,by=0.01),
+                                  a = 3.7),
+                      SURV = list(lambda = seq(0.01,0.12,by=0.01),
+                                  a = 3.7))
> SV <- list(b=fit$b,beta=fit$beta)
> tuneSCAD <- penPHcure(Surv(tstart,tstop,arrest)~fin+age+race+wexp+mar+paro+prio+educ+emp,
+                       cureform = ~fin+age+race+wexp+mar+paro+prio+educ+emp,
+                       data = cpRossi,which.X = "mean",pen.type = "SCAD",
+                       pen.tuneGrid = pen.tuneGrid,SV = SV)

Initializing PH cure model with time-varying covariates...

Number of individuals: 432
Censoring proportion: 0.736
Tied failure times: TRUE
Number of unique failure times: 49
Number of covariates in the survival component: 10
Number of covariates in the cure component: 10

Checking starting values...

Tuning SCAD-penalized PH cure model with time-varying covariates... Please wait...

  iter     aCURE     aSURV  lambdaCURE  lambdaSURV           AIC           BIC      df
     1      3.70      3.70        0.01        0.01     1319.1625     1384.2573      16
     2      3.70      3.70        0.01        0.02     1319.1625     1384.2573      16
     3      3.70      3.70        0.01        0.03     1316.0665     1360.8192      11
     4      3.70      3.70        0.01        0.04     1318.0458     1358.7300      10
     5      3.70      3.70        0.01        0.05     1318.0457     1358.7300      10

   ...       ...   (omitted rows)  ...         ...     (omitted rows)      ...     ...

   140      3.70      3.70        0.12        0.08     1325.5349     1333.6718       2
   141      3.70      3.70        0.12        0.09     1325.5349     1333.6718       2
   142      3.70      3.70        0.12        0.10     1325.5349     1333.6718       2
   143      3.70      3.70        0.12        0.11     1325.5349     1333.6718       2
   144      3.70      3.70        0.12        0.12     1325.5349     1333.6718       2

DONE!
```

This time, the call to the penPHcure function returned an object of class "penPHcure". We can print a summary of the results using the summary method, and, by default, the fitted model with the lowest BIC criterion is returned.

```
> summary(tuneSCAD)

-----------------------------------------------------
+++   PH cure model with time-varying covariates   +++
+++             [ Variable selection ]             +++
-----------------------------------------------------
Sample size:  432
Censoring proportion:  0.7361111
Number of unique event times: 49
Tied failure times:  TRUE
Penalty type:  SCAD
Selection criterion:  BIC

-----------------------------------------------------
+++               Tuning parameters                +++
-----------------------------------------------------
 Cure (incidence) --- lambda:  0.09
                          a:  3.7
```

```
 Survival (latency) - lambda:  0.05
                           a:  3.7

 BIC =  1329.481

-----------------------------------------------------
+++                Cure (incidence)               +++
+++     [ Coefficients of selected covariates ]    +++
-----------------------------------------------------
             Estimate
(Intercept)  1.776907
age         -0.076498

-----------------------------------------------------
+++              Survival (latency)               +++
+++     [ Coefficients of selected covariates ]    +++
-----------------------------------------------------
        Estimate
prio    0.101202
empyes -1.537286
```

The Bayesian Information Criterion is minimized for $\lambda_1 = 0.09$ and $\lambda_1 = 0.05$. In this case, the covariate age is selected in the incidence component. The negative sign of the estimated coefficient implies that younger individuals are more susceptible to be rearrested. The covariates `prio` and `emp` are selected in the latency component. The positive sign of the estimated coefficient (`prio`) implies that a higher number of convictions prior to incarceration increases the risk of being rearrested (among the individuals susceptible to be rearrested).

Let us now have a look at the fitted model with the lowest AIC criterion:

```
> summary(tuneSCAD,crit.type = "AIC")

-----------------------------------------------------
+++   PH cure model with time-varying covariates   +++
+++             [ Variable selection ]             +++
-----------------------------------------------------
Sample size:  432
Censoring proportion:  0.7361111
Number of unique event times: 49
Tied failure times:  TRUE
Penalty type:  SCAD
Selection criterion:  AIC

-----------------------------------------------------
+++               Tuning parameters               +++
-----------------------------------------------------
 Cure (incidence) --- lambda:  0.06
                           a:  3.7

 Survival (latency) - lambda:  0.03
                           a:  3.7

 AIC =  1310.79

-----------------------------------------------------
+++                Cure (incidence)               +++
+++     [ Coefficients of selected covariates ]    +++
-----------------------------------------------------
             Estimate
(Intercept)  1.829260
finyes      -0.585638
age         -0.067130
educ5       -0.887636

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

```
+++              Survival (latency)                +++
+++     [ Coefficients of selected covariates ]    +++
-------------------------------------------------------
           Estimate
raceother -0.586626
prio       0.103746
empyes    -1.552737
```

The Akaike Information Criterion is minimized for $\lambda_1 = 0.06$ and $\lambda_1 = 0.03$. As expected the AIC criterion selected a less penalized and more complex model. In the incidence component, also the covariates `fin` and `educ` have been selected. The negative signs imply that individuals who received financial aid or with a high level of education (>=12th degree) are less susceptible to be rearrested. In the latency component, also the covariate `race` has been selected. The negative coefficient implies that individuals of a race other than black have a lower risk of being rearrested (among the individuals susceptible to be rearrested).

Conclusion

In survival analysis studies, it may be the case that a fraction of the population is likely to be not susceptible to the event of interest. In this article, we presented the **penPHcure** R package, which implements the semiparametric proportional-hazards (PH) cure model of Sy and Taylor (2000) extended to time-varying covariates. This model can measure the effects of some covariates on the probability of being susceptible and on the time until the occurrence of the event. The **penPHcure** package is composed of two main functions: `penpHcure`, to estimate the regression coefficients, their confidence intervals using the basic/percentile bootstrap method and to perform variable selection using the SCAD-penalized likelihood technique proposed by Beretta and Heuchenne (2019a); and `penpHcure.simulate` to simulate data from a PH cure model with time-dependent covariates. We first explained the methodology behind these functions and presented the results of a simulation study to assess its finite-sample performance. Then, we illustrated the use of the `penPHcure` function through an example based on the Criminal Recidivism dataset.

Availability

The latest release and a development version of the **penPHcure** package are respectively avilable on CRAN and at https://github.com/a-beretta/penPHcure.

Bibliography

J. Amdahl. *flexsurvcure: Flexible Parametric Cure Models*, 2019. URL https://CRAN.R-project.org/package=flexsurvcure. R package version 1.0.0. [p117]

M. Amico and I. Van Keilegom. Cure models in survival analysis. *Annual Review of Statistics and Its Application*, 5(1):311–342, 2018. URL https://doi.org/10.1146/annurev-statistics-031017-100101. [p116]

A. Beger, D. Chiba, D. W. Hill, Jr., N. W. Metternich, S. Minhas, and M. D. Ward. *spduration: Split-Population Duration (Cure) Regression*, 2018. URL https://CRAN.R-project.org/package=spduration. R package version 0.17.1. [p117]

A. Beretta and C. Heuchenne. Variable selection in proportional hazards cure model with time-varying covariates, application to us bank failures. *Journal of Applied Statistics*, 46(9):1529–1549, 2019a. URL https://doi.org/10.1080/02664763.2018.1554627. [p116, 118, 127]

A. Beretta and C. Heuchenne. *penPHcure: Variable Selection in PH Cure Model with Time-Varying Covariates*, 2019b. URL https://CRAN.R-project.org/package=penPHcure. R package version 1.0.2. [p116]

J. Berkson and R. P. Gage. Survival curve for cancer patients following treatment. *Journal of the American Statistical Association*, 47(259):501–515, 1952. URL https://doi.org/10.2307/2281318. [p116]

J. W. Boag. Maximum likelihood estimates of the proportion of patients cured by cancer therapy. *Journal of the Royal Statistical Society: Series B (Methodological)*, 11(1):15–44, 1949. URL https://doi.org/10.1111/j.2517-6161.1949.tb00020.x. [p116]

N. Breslow. Covariance analysis of censored survival data. *Biometrics*, 30(1):89–99, 1974. URL https://doi.org/10.2307/2529620. [p118]

C. Cai, Y. Zou, Y. Peng, and J. Zhang. *smcure: Fit Semiparametric Mixture Cure Models*, 2012. URL https://CRAN.R-project.org/package=smcure. R package version 2.0. [p117]

R. A. Cole and J. W. Gunther. Separating the likelihood and timing of bank failure. *Journal of Banking & Finance*, 19(6):1073 – 1089, 1995. URL https://doi.org/10.1016/0378-4266(95)98952-M. [p116]

D. R. Cox. Regression models and life-tables. *Journal of the Royal Statistical Society. Series B (Methodological)*, 34(2):187–220, 1972. URL https://doi.org/10.2307/2985181. [p116]

A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977. URL https://doi.org/10.1111/j.2517-6161.1977.tb01600.x. [p117]

B. Efron. The efficiency of cox's likelihood function for censored data. *Journal of the American Statistical Association*, 72(359):557–565, 1977. URL https://doi.org/10.1080/01621459.1977.10480613. [p118]

J. Fan and R. Li. Variable selection via nonconcave penalized likelihood and its oracle properties. *Journal of the American Statistical Association*, 96(456):1348–1360, 2001. URL https://doi.org/10.1198/016214501753382273. [p118, 119]

J. Fan and R. Li. Variable selection for cox's proportional hazards model and frailty model. *Annals of Statistics*, 30(1):74–99, 2002. URL https://doi.org/10.1214/aos/1015362185. [p118]

J. Fox and M. Carvalho. The rcmdrplugin.survival package: Extending the r commander interface to survival analysis. *Journal of Statistical Software, Articles*, 49(7):1–32, 2012. URL https://doi.org/10.18637/jss.v049.i07. [p122]

G. Garibotti, A. Tsodikov, and M. Clements. *nltm: Non-Linear Transformation Models*, 2019. URL https://CRAN.R-project.org/package=nltm. R package version 1.4.2. [p117]

D. Hendry. Data generation for the cox proportional hazards model with time-dependent covariates: a method for medical researchers. *Statistics in medicine*, 33:436–454, 2014. URL https://doi.org/10.1002/sim.5945. [p116, 119]

D. R. Hunter and R. Li. Variable selection using mm algorithms. *The Annals of Statistics*, 33(4):1617–1642, 08 2005. URL https://doi.org/10.1214/009053605000000200. [p119]

C. Jackson. flexsurv: A platform for parametric survival modeling in R. *Journal of Statistical Software*, 70(8):1–33, 2016. URL https://doi.org/10.18637/jss.v070.i08. [p117]

W. Q. Meeker. Limited failure population life tests: Application to integrated circuit reliability. *Technometrics*, 29(1):51–65, 1987. URL https://doi.org/10.1080/00401706.1987.10488183. [p116]

Y. Polo, F. J. Sese, and P. C. Verhoef. The effect of pricing and advertising on customer retention in a liberalizing market. *Journal of Interactive Marketing*, 25(4):201 – 214, 2011. URL https://doi.org/10.1016/j.intmar.2011.02.002. [p116]

P. H. Rossi, R. A. Berk, and K. J. Lenihan. 2 - historical background of the transitional aid research project experiments. In P. H. Rossi, R. A. Berk, and K. J. Lenihan, editors, *Money, Work, and Crime*, pages 21 – 46. Academic Press, 1980. URL https://doi.org/10.1016/B978-0-12-598240-5.50009-0. [p122]

P. Schmidt and A. D. Witte. Predicting criminal recidivism using 'split population' survival time models. *Journal of Econometrics*, 40(1):141 – 159, 1989. URL https://doi.org/10.1016/0304-4076(89)90034-1. [p116]

M. Svolik. Authoritarian reversals and democratic consolidation. *American Political Science Review*, 102 (2):153–168, 2008. URL https://doi.org/10.1017/S0003055408080143. [p116]

J. P. Sy and J. M. G. Taylor. Estimation in a cox proportional hazards cure model. *Biometrics*, 56(1): 227–236, 2000. URL https://doi.org/10.1111/j.0006-341X.2000.00227.x. [p116, 127]

T. M. Therneau. *A Package for Survival Analysis in S*, 2015. URL https://CRAN.R-project.org/package=survival. version 2.38. [p120]

E. N. Tong, C. Mues, and L. C. Thomas. Mixture cure models in credit scoring: If and when borrowers default. *European Journal of Operational Research*, 218(1):132–139, 2012. URL https://doi.org/10.1016/j.ejor.2011.10.007. [p116]

A. Tsodikov. Semiparametric models: a generalized self-consistency approach. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 65(3):759–774, 2003. URL https://doi.org/10.1111/1467-9868.00414. [p117]

A. D. Tsodikov, J. G. Ibrahim, and A. Y. Yakovlev. Estimating cure rates from survival data: An alternative to two-component mixture models. *Journal of the American Statistical Association*, 98(464): 1063–1078, 2003. URL https://doi.org/10.1198/016214503000000001007. [p117]

Alessandro Beretta
Centre for Quantitative Methods and Operations Management (QuantOM)
HEC Liège
Rue Louvrex, 14 - 4000 Liège
Belgium
a.beretta@uliege.be

Cédric Heuchenne
Centre for Quantitative Methods and Operations Management (QuantOM)
HEC Liège
Rue Louvrex, 14 - 4000 Liège
Belgium
c.heuchenne@uliege.be

The bdpar Package: Big Data Pipelining Architecture for R

by Miguel Ferreiro-Díaz, Tomás R. Cotos-Yáñez, José R. Méndez and David Ruano-Ordás

Abstract In the last years, big data has become a useful paradigm for taking advantage of multiple sources to find relevant knowledge in real domains (such as the design of personalized marketing campaigns or helping to palliate the effects of several fatal diseases). Big data programming tools and methods have evolved over time from a MapReduce to a pipeline-based archetype. Concretely the use of pipelining schemes has become the most reliable way of processing and analyzing large amounts of data. To this end, this work introduces **bdpar**, a new highly customizable pipeline-based framework (using the OOP paradigm provided by **R6** package) able to execute multiple preprocessing tasks over heterogeneous data sources. Moreover, to increase the flexibility and performance, **bdpar** provides helpful features such as (i) the definition of a novel object-based pipe operator (`%>|%`), (ii) the ability to easily design and deploy new (and customized) input data parsers, tasks, and pipelines, (iii) only-once execution which avoids the execution of previously processed information (instances), guaranteeing that only new both input data and pipelines are executed, (iv) the capability to perform serial or parallel operations according to the user needs, (v) the inclusion of a debugging mechanism which allows users to check the status of each instance (and find possible errors) throughout the process.

Introduction

Social networks and instant messaging applications have arguably become an essential part of the human experience. In fact, nowadays, more than 60% of the population from industrialized countries use these mechanisms to communicate or share information. This phenomenon emerged due to (i) the declining costs of computers and storage systems by a factor of more than 200 (Engineering, 1984), (ii) an exponential increase in processing speed and computer hardware capabilities (Iansiti and Khansa, 1995), (iii) the emergence of high-throughput and fully-available communication networks (Dorogovtsev and Mendes, 2013), and (iv) certain human needs such as keeping interconnected and having permanent access to the data (Kabeer, 2005).

This scenario has promoted an exponential growth in the amount of data generated and stored in the last decade. Concretely, the latest reports from 2018 showed that around 2.16EB (exabytes) of data are created every day (Domo-Data, 2019; VCloud, 2019), and trends are showing that the growth of available information is four times higher than the world economy (VCloud, 2019). Indeed, 90% of the total world data have been created in the last two years alone (IBM, 2019).

In addition to the availability of unlimited sources and tools to generate, exchange, and handle information, the lack of a standardized way of representing data has led to a massive increase in unstructured information. In fact, approximately 80% of the existing data is unstructured (VCloud, 2019; IBM, 2019). The data obtained from a single source are usually insufficient to carry out a suitable decision making-process. However, the ability to take advantage of the combination of data from multiple (and unstructured) sources requires the execution of preprocessing operations that guarantee a unified data format. The need for facilitating the management and exploitation of vast amounts of heterogeneous data (in terms of data types and formats) within a reasonable elapsed time and cost-effective manner led to the emergence of the Big Data era (Mervis, 2012; Labrinidis and Jagadish, 2012).

Big Data is an abstract concept used to refer to the use of new programming paradigms able to handle large volumes of information and the execution of data mining tasks by taking advantage of parallel programming schemes over large computer clusters (IBM et al., 2011; Brown et al., 2011). In this context, MapReduce (Wu et al., 2014; Miner and Shook, 2012) is the most popular programming model to develop, execute and deploy Big Data analyses on large clusters. However, its batch-processing nature forces uploading data to the system (cluster) every time is analyzed, even when the input data has been previously utilized. This requirement (i) makes this programming paradigm unsuitable when leading with real-time streaming sources and (ii) avoids achieving full use of the computational capabilities and resources since clusters are idle while the data is being loaded. In order to solve these limitations, the utilization of pipelining schemes for big data processing was recently introduced by Di Tommaso (2019). This concept (extrapolated from the electronic domain) is focused on dividing the whole data analysis process into a set of computationally simple tasks (O'Donovan et al., 2015) whereby the required information for each task is handled exclusively, which avoids the (pre)loading of unnecessary information. This advantage has prompted the emergence of multiple enterprises offering cloud pipeline-based data analysis services such as BDB Solutions for Big Data (Solutions,

2019), AWS Amazon Data Pipeline (Amazon, 2019) or Google Cloud Dataflow (Google, 2019). These services allow users to build highly customized pipelines using a simple graphical interface, even if their technical skills are basic. Despite the great advantages of these services, their cloud-oriented nature causes customers to be reluctant to use them due to (i) the full control of the pipeline by the company offering the service, (ii) data privacy and security concerns (since information is executed in a foreign infrastructure), and (iii) the difficulty of assessing and calculating the cost of computational resources required to process data through the defined pipeline.

In order to cope with these problems, several customers decided to change the third-party cloud-based service to a proprietary solution by designing and implementing their own pipelining tools. Meanwhile, multiple open-source offline Big Data Pipeline frameworks emerged from the academic community to make this new paradigm available to everybody (Di Tommaso, 2019). However, as can be observed from the list shown in (Di Tommaso, 2019), despite the great number of available solutions, the majority are developed using Java language (>90%) while only a few belong to the R ecosystems. Among these, two packages should be mentioned, **repo** (Napolitano, 2020) and **drake** (Landau, 2018). The former one is a data-centered pipeline focused on solving bioinformatics data problems (specific-purpose application). Conversely, the latter is a generic pipeline tool that provides similar functionality to the GNU Make utility. As can be seen, despite both applications are focused on the same concept (pipelines), the target, implementation schema, and provided functionalities are quite divergent. In addition, we found some important issues that are not addressed by the actual pipelines tools such as (i) lack of a pure object-oriented (OO) implementation to facilitate the use and reduce the learning curve for people coming from object-oriented environments, (ii) the absence of an application based on the pipelining concept used by the well-known **magrittr** package (Bache and Wickham, 2020) (focused on UNIX pipes), (iii) the use of a black-box implementation which hampers users to easily trace and debug both code and the intermediate results.

This scenario motivated us to design and implement **bdpar**, a framework capable of unifying and preprocessing heterogeneous data through the development and execution of customizable pipelines. To this end, our package allows automatizing the management of a large amount of information by segmenting data into a sequence of simple and indivisible tasks (divide and conquer paradigm). Specifically, **bdpar** allows to (i) use or develop content extractors (such as SMS or email parsers), (ii) use and implement new preprocessing tasks (pipes), (iii) define customization pipelines (set of tasks) to achieve the desired (structured) output, (iv) visualize the intermediate results achieved by each instance after being processed by the tasks comprising the pipeline (white-box implementation), (v) prevent the re-execution of previously computed instances and tasks, and finally (vi) execute the pipeline following both a sequential or parallel paradigm.

This paper provides a full description of the main functionalities and resources of the **bdpar** package. The current version is 3.0.1, and an updated list (with the whole collection of resources) is available in the vignette document and the reference manual. The following section provides a complete description of the package structure and functionalities. Then, the use of the package is described, and finally, an illustrative case study is provided.

Package structure and operation

In order to exploit the main advantages and strengths of the object-oriented paradigm (such as maintainability, modularity, or inheritance), the **bdpar** package was fully developed using `R6` classes (package **R6** (Chang, 2019b)). Particularly, **bdpar** was implemented using R6 classes due to its high performance and ease of use when compared with other alternatives (such as S3 or S4) (Chang (2019a); Wickham (2019)). From an operational point of view, `R6` classes implemented in **bdpar** package are divided into three different categories: (i) data extraction functionalities; (ii) pipe-based operations; and finally (iii) **bdpar** framework configuration utilities.

The first category (data extraction methods) comprises all methods responsible for automatically detect the format and parse contents according to the inner structure of data gathered from input sources. The second category encapsulates the functionality of extracting features from the parsed information. By default, the **bdpar** framework provides a complete data preprocessing flow comprising 18 different tasks. Additionally, **bdpar** allows for the easy creation of new customized data flows by combining multiple tasks (object-based pipes). Finally, the third category of methods allows handling the configuration parameters needed for the proper operation of both the **bdpar** framework and some tasks using third-party functions (such as credentials for **rtweet** (Kearney, 2019) or **tuber** (Sood, 2019)).

In order to improve the readability of the code and facilitate the comprehension of each implemented class, a naming convention was adopted. Methods included in the data extraction category are labeled using `Extractor` as a prefix, followed by the type (or structure) of the input source (e.g., `ExtractorEml` and `ExtractorSms` methods are able to parse text contents from emails and SMS, re-

spectively). Finally, pipe-based functionalities are named using the operation name followed by the suffix `Pipe` (e.g., `ToLowerPipe` and `FindHashtagPipe` are tasks designed to convert text characters to lowercase and detect Twitter hashtags from textual contents, respectively).

As stated before, the **bdpar** framework is focused on designing, implementing, and deploying customized processing flows for the big data domain. In order to handle the information obtained from the input sources, the package uses a specific structure called `Instance`, which is responsible for storing the properties extracted by each pipe comprising the processing flow. To provide an insightful view of our **bdpar** framework, Figure 1 provides a graphical representation of its inner operation, which is divided into two main stages: (i) data loading and (ii) pipeline executing.

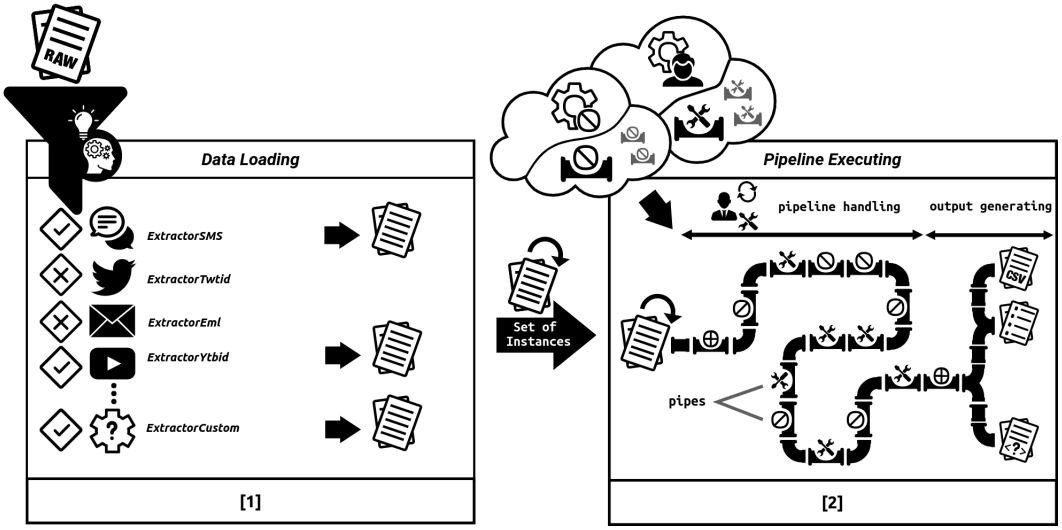


Figure 1: Description of **bdpar** two-stage execution process. The first stage is responsible for loading the dataset, while the second stage is in charge of executing the pipes comprising the user-defined preprocessing tasks.

As shown in Figure 1, the first stage comprises the loading of the required extractors according to the type of input data. By default, **bdpar** provides four different types of extractors: (i) `ExtractorSMS` is able to extract the textual contents exchanged through Short Message Service (SMS); (ii) `ExtractorTwtid` is capable of obtaining the text from Twitter entries (tweets); (iii) `ExtractorEML` can be used to gathering raw content from the body of email messages, and finally (iv) `ExtractorYtbid` allows extracting the comments published on the YouTube platform. Additionally, to increase the compatibility of **bdpar** with other data formats, the framework allows for the easy design and deployment of new customized extractors (by using a simple OOP inheritance relation).

Once the content is successfully extracted from the raw sources, the second stage is automatically initiated by **bdpar**. This stage comprises the execution of two steps: (i) pipeline handling; and (ii) output generating. The former is in charge of performing unified data processing by executing a specific set of pipes (also named pipeline) over the previously extracted content. It should be noted that each pipe included in the pipeline is represented as an object (inherited from `GenericPipe` class) responsible for performing a specific operation (task) over the input data. The second step (called output generating) transforms the preprocessed data into a specific output format (e.g., into a CSV structure). Moreover, **bdpar** allows users to develop new specific output-generation methods to achieve the desired output.

As can be realized from the pipeline handling stage shown in Figure 1, the pipe-based structure provides great flexibility and versatility to users since it allows users to easily (i) modify existing pipelines by adding or removing pipes, (ii) develop new pipes implementing additional tasks, or (iii) design and deploy new customized pipelines. To assist users in the creation and deployment of new pipelines, **bdpar** provides a set of 18 combinable pipes implementing basic preprocessing tasks for text sources. As can be seen in Table 1, tasks included in **bdpar** are divided into two different categories: (i) transformer tasks which are able to perform operations that successively alter the original content (such as lowercase conversion or emoticon finding), and (ii) maintainers which are responsible for executing operations that do not affect the current content (such as storing the extension of the input data).

Pipe name	Pipe type	Name of computed property	Description
`GuessDatePipe`	Transformer	"date"	Obtains the date and time based on the type and structure of the input data.
`File2Pipe`	Transformer	"source"	Obtains the source based on the type and structure of the input data.
`FindUserNamePipe`	Transformer	"userName"	Detects and extracts usernames from textual sources.
`FindHashtagPipe`	Transformer	"hashtag"	Detects and obtains hashtags from input data.
`FindUrlPipe`	Transformer	"URLs"	Uses regular expressions to find URLs in text.
FindEmoticonPipe	Transformer	"emoticon"	Identifies and extracts emoticons from textual sources.
`FindEmojiPipe`	Transformer	"Emojis"	Transforms emojis to its textual representation.
`GuessLanguagePipe`	Transformer	"language"	Tries to guess the language of a specific text.
`ContractionPipe`	Transformer	"contractions"	Transforms previously detected contractions.
`AbbreviationPipe`	Transformer	"abbreviation"	Expands detected abbreviations.
`SlangPipe`	Transformer	"langpropname"	Identifies slang words to its corresponding formal speech.
`ToLowerCasePipe`	Transformer	–	Converts the input source to lowercase characters.
`InterjectionPipe`	Transformer	"interjection"	Detects and extracts interjections from textual sources.
`StopWordPipe`	Transformer	"stopWord"	Recognizes and obtains stop words from textual sources.
`TargetAssigningPipe`	Maintainer	"target"	Identifies the target class of the data.
`StoreFileExtPipe`	Maintainer	"extension"	Guess the extension of the input data.
`MeasureLengthPipe`	Maintainer	"length"	Computes the length of a given text.
`TeeCSVPipe`	Maintainer	–	Transforms the final result into a CSV format file.

Table 1: Pipes provided by **bdpar** framework.

Additionally, each pipe provides a property name field where the computed property will be stored. To increase flexibility, property names can be specified by users or leave it by default (see names described in Table 1). Finally, pipes definition in Table 1 were sorted to match the execution order defined in the pipeline included by default in **bdpar** (named as `DefaultPipeline`)

Moreover, to increase the reliability of the pipeline, **bdpar** allows defining the execution order of each task comprising the specified pipeline. During the definition of a pipeline, we should specifically take into account the possible interdependence between pipes (e.g., language-dependent tasks should be executed after `GuessLanguagePipe`). To solve this situation and ensure the proper creation and execution of the pipelining process, **bdpar** provides a pipe-orchestration system. This mechanism is

automatically invoked when the pipeline is executed and traverses all tasks comprising the pipeline to evaluate two types of interdependencies: (i) always-before (or 'a priori dependence') and (ii) not-after (or 'a posteriori dependence'). The first constraint is used when a specific pipe requires the previous execution of other tasks to ensure its proper operation (for instance, `AbbreviationPipe` needs to know the text language, so it should not be executed before `GuessLanguagePipe`). Conversely, the second type of restriction is used to indicate the tasks that cannot be started after the execution of the current one (for instance, the recognition of contractions should not be run after changing text characters to lowercase or removing punctuation marks). Both restrictions are automatically managed through the `checkCompatibility` method included in the `Instance` class.

Furthermore, in order to ensure the proper execution of tasks implemented through `R6` classes within the pipeline, **bdpar** implements an object-oriented customized operator (denoted as `%>|%`) inspired by the implementation of the primitive forward-pipe operator (`%>%`) provided by the **magrittr** package (Bache and Wickham, 2020). Particularly, the execution of this new operator implies some inner operations such as (i) discarding an Instance (left-side operand) whenever an error occurs during the data processing flow, (ii) automatically manage pipes dependencies between pipes, (iii) simplifying the invocation of the associated pipe task (right-side operand) by hiding its explicit call, (iv) facilitate debugging issues by showing log messages with different levels of granularity, (v) the ability to display the intermediate computation results of each instance throughout the whole preprocessing flow and (vi) the capability to avoid the re-execution of previously processed pipelines. To this end, the operator transparently calls the `pipe` method defined in the pipe object. These functionalities allow improving the processing capabilities (in terms of speed, performance, and usability) of the application by preventing the problems derived from the (potential) existence of errors during the pipelining process and shorten pipelining definition by taking advantage of the customized operator capabilities. To increase the customization capabilities, **bdpar** allows easy development and deployment of new user-defined pipelines. To ensure full compatibility of new user-defined pipelines **bdpar** provides a reference class called `GenericPipeline`. Additionally, to simplify the use of the framework, **bdpar** provides a predefined pipeline (named `DefaultPipeline`) containing all the pipes included in Table 1.

Finally, once all `Instance` objects are processed, the output generation stage starts. As can be seen from Figure 1, this stage is responsible for storing the results achieved after executing the pipeline process over each (valid) `Instance`. Although this stage allows the use of customized storage and output-representation methods (implemented by user), **bdpar** provides two methods able to (i) save the achieved output into an external CSV file (using `TeeCSVPipe` pipe) or (ii) internally store in memory a set of preprocessed `Instance` objects (default output).

In order to exemplify the structure and operation of an object-based pipeline in **bdpar**, we have included below a code snippet comprising 13 different text processing tasks (implemented as pipe objects).

```
instance %>|%
   TargetAssigningPipe$new() %>|% StoreFileExtPipe$new() %>|%
   GuessDatePipe$new() %>|% File2Pipe$new() %>|%
   MeasureLengthPipe$new("length_before_cleaning_text") %>|%
   FindUrlPipe$new() %>|% FindEmojiPipe$new() %>|% GuessLanguagePipe$new() %>|%
   SlangPipe$new() %>|% ToLowerCasePipe$new() %>|%
   InterjectionPipe$new() %>|% StopWordPipe$new() %>|%
   MeasureLengthPipe$new("length_after_cleaning_text") %>|% TeeCSVPipe$new()
```

To facilitate the understanding of the pipelining process, the code included above assumes that each input data has been successfully loaded and stored in an `Instance` object (denoted as `instance`). As can be seen from the code snippet, each `instance` is processed through all the tasks comprising this pipeline. Specifically, the first 13 ones perform different preprocessing operations over each `instance`, while the latest one stores the achieved results into a CSV file.

Using bdpar package

The package can be installed and attached as described in the code included below (please refer to the 'README' file to access the latest and development versions).

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

Please note that the core functionalities of **bdpar** require the previous installation of six R packages (described in the '`Imports`' field included in the 'DESCRIPTION' file). In addition, some optional tasks (mainly belonging to specific data-processing pipes) used certain packages (indicated on the '`Suggest`'

field included in the 'DESCRIPTION' file) and should also be installed in order to ensure its proper operation. It should be taken into account that in case of needing all the dependencies, the argument `dependencies = TRUE` should be included in the command `install.packages`.

Executing bdpar framework

In order to guarantee a high level of flexibility, **bdpar** can be easily executed by using two different ways (i) following an OOP paradigm or (ii) using a classical function call approach. Below is include a code snippet describing both scenarios.

```
bdpar <- Bdpar$new()
bdpar$execute(path, extractors= ExtractorFactory$new(),
              pipeline= DefaultPipeline$new(), cache= TRUE,
              verbose= FALSE, summary= FALSE)
```

a) Executing **bdpar** using OOP paradigm

```
output <- line(path, extractors = ExtractorFactory$new(),
               pipeline= DefaultPipeline$new(), cache= TRUE,
               verbose= FALSE, summary= FALSE)
```

b) Executing **bdpar** following a function-based approach.

As can be depicted, both execution methods require the same six arguments since `runPipeline` is a wrapper function which encapsulates **bdpar** execution using the OOP paradigm. The first argument is mandatory since it is used to specify the directory or file(s) path where the raw input data is located. The second parameter indicates the extractors required to parse the input sources. If not defined, **bdpar** automatically invokes the default `ExtractorFactory$new()` object which initializes the four extractors provided by **bdpar** framework. Following, the third argument is used to determine the sequence of preprocessing tasks that should be executed (pipeline) to achieve the desired output (featured dataset). If the argument is not assigned, **bdpar** executes `DefaultPipeline$new()` object which implements an error-safe pipeline comprised of 18 tasks described in Table 1. The fourth one is used to enable (or disable) **bdpar** not-re-execution functionality (defined as `N-RE`). Particularly, this feature is able to detect which instances, tasks, and even pipelines were previously executed with a view to avoiding their re-execution. Moreover, the penultimate argument is used to indicate (if needed) the generation of a log output showing different levels of granularity (`DEBUG`, `INFO`, `WARN`, `ERROR`, `FATAL`). It should be noted that `DEBUG` level allows displaying the intermediate results achieved by each instance after being processed by the tasks comprising the pipeline. This is very useful to detect the location of possible errors. Finally, the latter argument allows showing (if needed) a detailed summary of all the operations and tasks performed during the pipeline execution.

Developing new functionalities

As previously stated, the design of the software architecture of **bdpar** is focused on facilitating the customization of any stage of the process (data loader and pipeline executor). Specifically, **bdpar** allows users to: (i) defining new types of input data parses, (ii) creating new pipes, and (iii) implementing and deploying new preprocessing tasks.

Regarding the first aspect, the development of new content parsers involves two stages: (i) the implementation of a customized extractor by overriding solely the methods of the `Instance` class that are necessary to load the input and (ii) the registration of the created extractor so it can be loaded by the **bdpar** framework. Two code fragments to detail the development and registration of a new customized parsers is included below.

```
ExtractorImage <- R6::R6Class(
  classname = "ExtractorImage",
  inherit = Instance,
  public = list(
    initialize = function(path) {
      super$initialize(path)
    },
    obtainSource = function() {
      source <- imager::load.image(super$getPath())
      super$setSource(source)
      super$setData(source)
    } ) )
```

a) Implementation of new `ExtractorImage` parser.

```
extractors <- ExtractorFactory$new()
extractors$registerExtractor(extension= c("jpeg","png"),
                             extractor= ExtractorImage)
```

b) Dynamic extractor registration operation in **bdpar**.

As can be seen, the first code snippet describes the formal structure of a new extractor. Particularly, `ExtractorImage` is able to load an image into an Instance object. To accomplish this task, the `obtainSource` method loads (by invoking `load.image()` method) an image from the file path received as a parameter of the class constructor (`super$getPath()`). Then, the loaded image is stored in the source variable of the `Instance` superclass (by invoking `super$setSource()` method) and is assigned to the data variable by calling to `super$setData()` method. The data field is used to store the result of each task comprising the pipeline.

Moreover, the second fragment of code exemplifies the registration of the previously created extractor in **bdpar** framework. As can be depicted, this operation is performed by a simple call to the `registerExtractor` method included in `ExtractorFactory` class. Due to the one-to-one dependency between each extractor and the different input formats, **bdpar** requires the definition of a specific extension (or set of extensions) to discern which type of extractor should execute. Also, `ExtractorFactory` provides two additional methods (i) `getAllExtractors`, which shows all registered extractors in **bdpar** framework and (ii) `removeExtractor`, which deletes a specific data extractor. Finally, to avoid parsing errors, unsupported input contents by registered extractors are automatically ignored by **bdpar**.

For the creation and deployment of new preprocessing tasks, **bdpar** provides an abstract class named `GenericPipe`. This type of class is very common in OOP to ensure all subclasses (pipes) follow the same structure and implement the methods defined in the superclass (`GenericPipe`). Particularly, `GenericPipe` defines two main methods that should be included in each subclass: (i) `initialize` and (ii) `pipe`. The former includes three optional parameters that are `propertyName`, which refers to the specific name to the output value computed in the task, `alwaysBeforeDeps`, and `notAfterDeps`, which handles two types of dependencies between pipes ("always-before" and "not-after", respectively). Finally, the `pipe` method is used to implement the behavior of the new task. Below we include a code snippet exemplifying how to develop a basic image-preprocess pipeline. Concretely, we design three pipes: (i) `Image2Pipe` responsible for invoking the `obtainSource` method provided in the `ExtractorImage` parser, (ii) `ImageCroppingPipe` in charge of halving the image, and (iii) `ImageRotatePipe`, which rotates the image 30 degrees clockwise.

```
Image2Pipe <- R6::R6Class(
  name = "Image2Pipe",
  inherit = GenericPipe,
  public = list(
    initialize = function(propertyName= "",
                          alwaysBeforeDeps= list(),
                          notAfterDeps= list()) {
                  super$initialize(propertyName, alwaysBeforeDeps, notAfterDeps)
    },
    pipe = function(instance) {
      instance$obtainSource()
      instance
    } ) )

ImageCroppingPipe <- R6::R6Class(
  "ImageCroppingPipe",
  inherit = GenericPipe,
  public = list(
    initialize = function(propertyName= "",
                          alwaysBeforeDeps= list("Image2Pipe"),
                          notAfterDeps= list()) {
                  super$initialize(propertyName, alwaysBeforeDeps, notAfterDeps)
    },
    pipe = function(instance) {
    data <- instance$getData()
    data <- imager::imsub(data, x > height/2)
    instance$setData(data)
      instance
    } ) )
```

```
ImageResizePipe <- R6::R6Class(
  "ImageResizePipe",
  inherit = GenericPipe,
  public = list(
    initialize = function(propertyName= "",
                          alwaysBeforeDeps= list("Image2Pipe"),
                          notAfterDeps= list()) {
                  super$initialize(propertyName, alwaysBeforeDeps, notAfterDeps)
    },
    pipe = function(instance) {
    data <- instance$getData()
    data <- imager::imrotate(data, 30)
    instance$setData(data)
      instance
    } ) )
```

As can be seen, the `Image2Pipe` class stores the loaded image into a new instance. Following, `ImageCroppingPipe` and `ImageResizePipe` class apply different image manipulation functions (`imager::imsub`, `imager::imrotate`) to halve and rotate the images, respectively. Following the result, of each pipe is stored into a specific field of the `Instance` object (by calling the `instance$setData()` method). To ensure proper operation of both tasks, image content should be previously loaded into an instance object (by invoking `Image2Pipe` associated task). To this end, a priori dependence with `Image2Pipe` has been defined in the `initialize` method of both pipes. As mentioned before, to facilitate the development and execution of pipes, dependencies between pipes are automatically managed by the object-oriented pipe operator (%>|%). Finally, in order to develop robust pipelines, `instance` objects should be invalidated when an error occurs, or the requirements are not satisfied (such as the storage of empty data or non-identification of textual language).

Finally, **bdpar** allows users to customize existing pipelines and develop new ones from scratch. In order to motivate the usage of **bdpar** regardless of user programming skills, the framework allows to manually or dynamically design new pipelines. The first method requires the creation of a new class (inheriting from `GenericPipeline`) which implements the `execute` method defined in the parent class. Moreover, to ensure proper management of (possible) execution errors (such as invalidated instances), a try-catch function should be included. Additionally, **bdpar** allows customizing the log messages (if needed) by calling the `bdpar.log()` function. Below we include an example showing how a customized pipeline is manually created.

```
TestPipeline <- R6::R6Class(
  classname = "TestPipeline",
  inherit = GenericPipeline,
  public = list(
    initialize = function() ,
    execute = function(instance)
      message("[TestPipeline][execute][Info] ", instance$getPath())
      tryCatch(
         instance %>|% Image2Pipe$new() %>|%
             ImageCroppingPipe$new() %>|% ImageResizePipe$new(),
          error = function(e)
               bdpar.log(message = paste0(instance$getPath()," :", paste(e)),
                 level= "ERROR",
                 className= class(self)[1],
                 methodName= "execute")
                 instance$invalidate()
         )
      return(instance)
     ) )
```

On the other hand, the dynamic method allows the creation of custom pipelines by simply indicating a list containing the pipe objects to be used. To achieve a higher level of flexibility dynamic, the model provides two ways of defining pipelines: (i) during the object instantiation or (ii) by calling the `add` function. A simple example describing how the previous pipeline is created following the dynamic method is included below.

```
pipeline <- DynamicPipeline$new(pipeline= list(Image2Pipe$new(),
```

```
                                                ImageCroppingPipe$new(),
                                                ImageResizePipe$new()))
```

a) Pipeline definition during object instantiation.

```
pipeline <- DynamicPipeline$new()
pipeline$add(list(Image2Pipe$new(), ImageCroppingPipe$new(),
                  ImageResizePipe$new()))
```

b) Pipeline creation using the add method.

Additionally, the dynamic method provides six methods able to extend the pipeline customization capabilities: (i) `add(pipe,pos=NULL)`, which adds new pipe object(s) to the pipeline flow at a certain position (or at the end if not defined), (ii) `removeByPos(pos)`, which removes a pipe object at a given position, (iii) `removeByPipe(pipe.name)` responsible for erasing a `pipe` object by name, (iv) `removeAll()` capable of releasing all pipe objects from the pipeline, (v) `get()` returning a list containing the pipe objects comprising the pipeline, and finally, (vi) `print()` which displays the pipes comprised in the pipeline.

As can be realized from both methods, the manual definition of pipelines allows users to have greater control and insight over the pipeline (such as personalizing error-handling methods or the inclusion of new user-defined object-oriented pipeline operators). Conversely, the dynamic mode enables users to define optimal pipelines without expert knowledge of `R6` and OOP concepts.

Managing bdpar configuration options

`bdpar.Options` included in **bdpar** allows managing configuration parameters to customize the behavior of available tasks and indicate parameters needed for the proper operation of pipes and/or content extractors (such as path locations for slang dictionaries or credentials required by Twitter or Youtube APIs, respectively). Moreover, to easily search and access the configuration, `bdpar.Options` stores parameters following a key-value pair structure. As can be deducted, the key parameter is used to uniquely identify a configuration entry. In order to facilitate the management of configuration parameters, `bdpar.Options` provides four main methods: (i) `bdpar.Options$add(key,value)`, which adds a new configuration entry, (ii) `bdpar.Options$set(key,value)` used to modify the value of an existing configuration parameter, (iii) `bdpar.Options$remove(key)`, which removes an entry matching a specific name, and finally, (iv) `bdpar.Options$reset()` used to restore `bdpar.Options` to its initial state (default options). Table 2 describes the configuration options included in **bdpar**.

Type	Key	Assigned by default	Description
API credentials	twitter.consumer.key twitter.consumer.secret twitter.access.token twitter.access.token.secret	x	Set of keys need to connect to Twitter and Youtube API
	youtube.app.id youtube.app.password	x	
API cache	cache.youtube.path cache.twitter.path	x	Path to temporary place extracted data
Pipe parameters	teeCSVPipe.output.path	√	Defines the output file path for `TeeCSVPipe`
Extractor options	extractorEML.mpaPartSelected	√	Indicates the content-type to parse on multipart emails
N-RE handler	cache.folder	√	Indicates the path to store the intermediate results
Parallel settings	numCores	√	Selects the number of cores used to execute the pipelines
Resource files	resources.abbreviations.path resources.contractions.path resources.interjections.path resources.slangs.path resources.stopwords.path	√	Location for the different language dictionaries (slang, contractions, . . .)

Table 2: Structure of **bdpar** configuration options.

As can be seen from Table 2, configuration options are divided into seven categories: (i) API credentials, (ii) API cache, (iii) pipe parameters (iv) Extractor options, (v) instance cache handler, (vi) parallel settings, and (vii) resource files. It is important to take into account that values for API credentials are not provided by default due to the inexistence of publicly available access keys for both Youtube and Twitter (only for private use with prior approval). Following, the optional API cache configuration entries are responsible for designating temporal locations to store information obtained after executing the content extractors. Defining cache paths API ensures that duplicated sources are executed only once (avoids parsing duplicated inputs).

Moreover, pipe parameters and extractor options allow specifying configuration values needed to guarantee the proper execution of pipes and extractors, respectively. Particularly, default "extractorEML.mpaPartSelected" entry allows defining which content-type (text/plain or text/html) should be extracted in multipart emails while "teeCSVPipe.output.path" indicates the location to store the CSV file generated by `TeeCSVPipe` pipe.

In addition, the not-re-execution handler allows defining the path to store the information required for the proper operation of the not-re-execution functionality. Despite this, the feature is very useful to reduce both unnecessary computation costs and time consumption that requires extra storage space. Therefore, **bdpar** allows the deletion of the intermediate results by invoking the specific `bdpar.Options$cleanCache()` method.

Following, parallel settings category is used to define the configuration values to handle parallelization in **bdpar**. Concretely, "numCores" entry allows defining the number of CPU cores to be used when a pipeline is executed. By default, **bdpar** is configured following a sequential paradigm (`numCores = 1`). Additionally, to ensure proper use of CPU resources **bdpar** provides a mechanism to verify whether the number of assigned CPU cores is compatible with the hardware specifications or not. If the assigned CPU cores are not valid, **bdpar** will be executed using the most optimal configuration according to the hardware specifications.

The last category comprises different dictionaries needed to perform multiple language-dependent operations (such as contraction detection or stopwords removal). Dictionaries provided by default ensure the compatibility of **bdpar** from 8 to 50 different languages (depending on the text-mining operation selected).

A case study

In order to illustrate the functionality of the **bdpar** package from a more realistic perspective, we developed a case study to show the most frequent words from a heterogeneous dataset collection (containing SMS and emails). The dataset comprises 20 emails (eml format) and 20 SMS in plain text from the nutritional and health domain. Moreover, to ensure straightforward reproducibility, (i) all the resources used are included in the package, and the pipeline provided by the package (`DefaultPipeline`) was selected to perform the content preprocessing flow, and (ii) resulting dataset was stored into a structured CSV file. Once the pipeline was executed, 18 new columns were generated from the outputs acquired after executing some pipeline tasks (labeled according to the property names described in Table 1). For instance, the execution of `FindEmojiPipe` forces the creation (if not exists) of a new column (named "emojis") containing the emojis found for each instance (or blank if not found).

To carry out the case of study, word frequencies were computed over the text content generated after executing the whole pipeline tasks (stored in the data column). Additionally, some previous text-cleaning operations were performed over the preprocessed text prior to executing the computation of the word frequencies (using word-cloud plots). Concretely, words were reduced to their stem form (stemDocument), punctuation marks were deleted, and numbers were removed (using the **tm** package (Feinerer et al., 2008)). Finally, for comparison purposes, frequencies were calculated both individually and jointly (see Figures 2 and 3)

```
#Execute bdpar framework
bdpar::runPipeline(path= system.file("example", package= "bdpar"),
                   cache= FALSE)
#Load CSV generated after executing bdpar
dataset <- read.csv(file= bdpar.Options$get("teeCSVPipe.output.path"),
                    sep= ";", stringsAsFactors= FALSE )

#Separate instances by type
sms <- dataset[dataset$extension == "tsms", ]
eml <- dataset[dataset$extension == "eml", ]
# Function to clean text and compute frequencies
word.frec <- function(data) {
  corpus <- tm::VCorpus(VectorSource(data))
  corpus <- tm::tm_map(corpus, removePunctuation)
  corpus <- tm::tm_map(corpus, removeNumbers)
  corpus <- tm::tm_map(corpus, stemDocument)
  sorted <- sort(rowSums(as.matrix(tm::TermDocumentMatrix(corpus))),
                 decreasing = TRUE)
  return(data.frame(word = names(sorted), freq = sorted))
}
sms.words <- word.frec(sms$data)
eml.words <- word.frec(eml$data)
all.words <- word.frec(dataset$data)
# Wordcloud for sms and emails
par(mfrow=c(1,2))
wordcloud::wordcloud(words= sms.words$word, freq= sms.words$freq,
                     min.freq= 1, max.words= 100, random.order= FALSE,
                     rot.per= .5, colors= RColorBrewer::brewer.pal(8, "Dark2"))
wordcloud::wordcloud(words= eml.words$word, freq= eml.words$freq,
                     min.freq= 1, max.words= 100, random.order= FALSE,
                     rot.per= .5, colors= RColorBrewer::brewer.pal(8, "Dark2"))
par(mfrow=c(1,1))
#Wordcloud for all instances (sms and email)
wordcloud::wordcloud(words= dataset.words$word, freq= all.words$freq,
                     min.freq= 1, max.words= 100, random.order= FALSE,
                     rot.per= .5, colors= RColorBrewer::brewer.pal(8, "Dark2"))
```

Figure 2 graphically represents the results achieved after performing an individualized analysis of each dataset (SMS and emails). Conversely, Figure 3 represents the frequencies achieved when analyzing both datasets together. Observing Figure 2, we see that frequencies of words included in SMS messages are lower than those included in emails. This is mainly due to the limited length of SMS messages (up to 160 characters). Therefore, the word frequency in Figure 3 has increased considerably, mainly owing to the joint evaluation of both datasets.

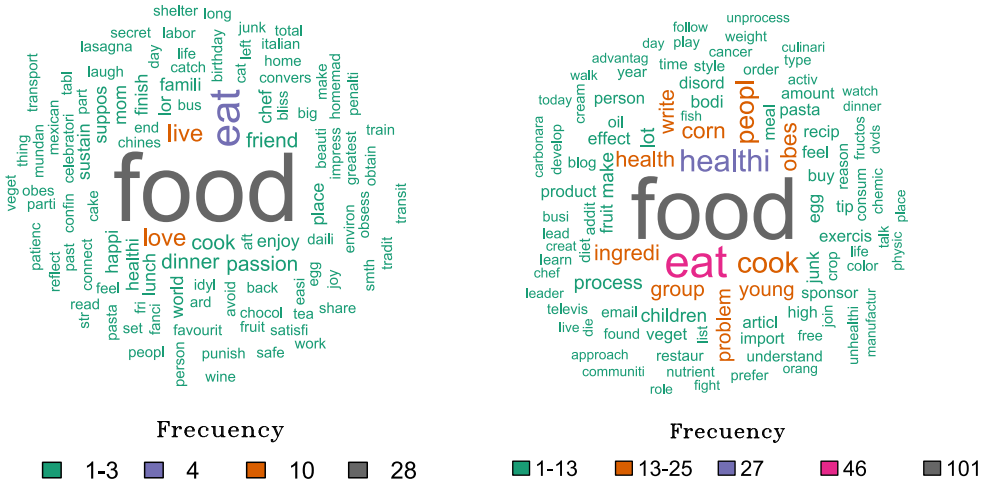


Figure 2: Wordcloud for SMS (left) and email (right).

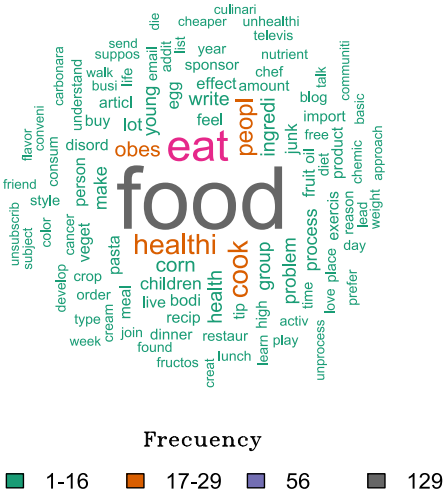


Figure 3: Wordcloud for SMS and email jointly.

Keeping in mind the functionality provided by **bdpar** (demonstrated through the current case study), it is easy to deduce that the application of data mining techniques over unstructured data could be easily addressed by taking advantage of the functionality of our framework. Some big data operations that could be addressed by taking advantage of **bdpar** are the clustering of documents using token features, the classification of documents, or the retrieval of documents for specific queries.

Conclusions and future work

In this work, we introduced **bdpar**, a pipe-based R framework to facilitate the creation of unified datasets from heterogeneous sources. Our framework allows users to (i) define new content extractors (data parsers), (ii) develop and deploy new preprocessing tasks (pipes), and (iii) define and build customized interconnected task flows (pipelines). Additionally, to save computational resources and

increase execution speed, **bdpar** provides an optimized pipe operator (noted as `%>I%`) capable of aborting the processing of an instance if an error was detected. Finally, a case study was developed to demonstrate the capability of the framework to preprocess and unify heterogeneous data into a single CSV file.

Future work is focused on two main aspects: (i) the development of semantic-based tasks able to explode the semantic relationships between synsets and; (ii) the capability to represent using a graph-based visualization the pipes comprising each pipeline, and (iii) the analysis of textual polarity and sentiment analysis.

Acknowledgements

The work of Tomás R. Cotos-Yáñez has been partially supported by the projects IN2017-84658-C2-1-R and PID2020-118101GB-I00 of the Spanish Ministry of Industry. David Ruano-Ordás has been supported by a post-doctoral fellowship from Xunta de Galicia (POSB-2021/024). Additionally, the work of José R. Méndez was partially funded by the project Semantic Knowledge Integration for Content-Based Spam Filtering (TIN2017-84658-C2-1-R) from the Spanish Ministry of Economy, Industry, and Competitiveness (SMEIC), State Research Agency (SRA) and the European Regional Development Fund (ERDF). SING group thanks CITI (Centro de Investigación, Transferencia e Innovación) from the University of Vigo for hosting its IT infrastructure. Finally, we thank the reviewers for their deep appropriate suggestions to improve the quality of the manuscript.

Bibliography

A. Amazon. AWS Amazon Data Pipeline. https://aws.amazon.com/es/datapipeline, 2019. Accesed: 2019-05-6. [p131]

S. M. Bache and H. Wickham. *magrittr: A Forward-Pipe Operator for R*, 2020. URL https://CRAN.R-project.org/package=magrittr. R package version 2.0.1. [p131, 134]

Brown, Brad, M. Chui, and J. Manyika. Are you ready for the era of 'big data'. *McKinsey Quarterly*, 4 (1):24–35, 2011. [p130]

W. Chang. *R6 and Reference Class performance tests*, 2019a. URL https://r6.r-lib.org/articles/Performance.html. R6 package version 2.4.1. [p131]

W. Chang. *R6: Encapsulated Classes with Reference Semantics*, 2019b. URL https://CRAN.R-project.org/package=R6. R package version 2.4.0. [p131]

P. Di Tommaso. Awesome pipeline. https://github.com/pditommaso/awesome-pipeline, 2019. [p130, 131]

Domo-Data. Domo-data never sleeps 6.0. https://www.domo.com/solution/data-never-sleeps-6, 2019. Accesed: 2019-04-05. [p130]

S. Dorogovtsev and J. Mendes. *Evolution of Networks: From Biological Nets to the Internet and WWW*. OUP Oxford, 2013. ISBN 9780191004407. URL https://books.google.es/books?id=FFL1AgAAQBAJ. [p130]

N. A. o. Engineering. *Cutting Edge Technologies*. The National Academies Press, Washington, DC, 1984. ISBN 978-0-309-03489-0. doi: 10.17226/286. URL https://doi.org/10.17226/286. [p130]

I. Feinerer, K. Hornik, and D. Meyer. Text mining infrastructure in R. *Journal of Statistical Software*, 25 (5):1–54, March 2008. URL http://www.jstatsoft.org/v25/i05/. [p140]

Google. Google cloud dataflow. https://cloud.google.com/solutions/big-data, 2019. Accesed: 2019-05-6. [p131]

M. Iansiti and T. Khansa. Technological Evolution, System Architecture and the Obsolescence of Firm Capabilities. *Industrial and Corporate Change*, 4(2):333–361, 03 1995. ISSN 0960-6491. doi: 10.1093/icc/4.2.333. URL https://doi.org/10.1093/icc/4.2.333. [p130]

IBM. IBM Big Data Success Stories. http://public.dhe.ibm.com/software/data/sw-library/big-data/ibm-big-data-success.pdf, 2019. Accesed: 2019-04-8. [p130]

IBM, P. Zikopoulos, and C. Eaton. *Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data*. McGraw-Hill Osborne Media, 1st edition, 2011. ISBN 0071790535, 9780071790536. [p130]

N. Kabeer. Introduction: The search for inclusive citizenship: Meanings and expressions in an interconnected world. 2005. [p130]

M. W. Kearney. *rtweet: Collecting Twitter Data*, 2019. URL https://cran.r-project.org/package=rtweet. R package version 0.6.9. [p131]

A. Labrinidis and H. V. Jagadish. Challenges and opportunities with big data. *Proc. VLDB Endow.*, 5 (12):2032–2033, Aug. 2012. ISSN 2150-8097. doi: 10.14778/2367502.2367572. URL https://doi.org/10.14778/2367502.2367572. [p130]

W. M. Landau. The drake R package: a pipeline toolkit for reproducibility and high-performance computing. *Journal of Open Source Software*, 3(21), 2018. URL https://doi.org/10.21105/joss.00550. [p131]

J. Mervis. Agencies rally to tackle big data. *Science*, 336(6077):22–22, 2012. ISSN 0036-8075. doi: 10.1126/science.336.6077.22. URL https://doi.org/10.1093/10.1126/science.336.6077.22. [p130]

D. Miner and A. Shook. *MapReduce Design Patterns: Building Effective Algorithms and Analytics for Hadoop and Other Systems*. O'Reilly Media, Inc., 1st edition, 2012. ISBN 1449327176, 9781449327170. [p130]

F. Napolitano. *repo: A Data-Centered Data Flow Manager*, 2020. URL https://CRAN.R-project.org/package=repo. R package version 2.1.5. [p131]

P. O'Donovan, K. Leahy, K. Bruton, and D. T. J. O'Sullivan. An industrial big data pipeline for data-driven analytics maintenance applications in large-scale smart manufacturing facilities. *Journal of Big Data*, 2(1):25, 2015. ISSN 2196-1115. doi: 10.1186/s40537-015-0034-z. URL https://doi.org/10.1186/s40537-015-0034-z. [p130]

B. Solutions. BDB Solutions for Big Data. https://www.bdb.ai/big-Data-Pipeline, 2019. Accesed: 2019-05-6. [p130]

G. Sood. *tuber: Access YouTube from R*, 2019. URL https://CRAN.R-project.org/package=tuber. R package version 0.9.8. [p131]

VCloud. Vcloud news. http://www.vcloudnews.com/every-day-big-data-statistics-2-5-quintillion-bytes-of-data-created-daily/, 2019. Accesed: 2019-04-17. [p130]

H. Wickham. *Advanced R, Second Edition*. CRC Press, 2019. ISBN 9780815384571. URL https://adv-r.hadley.nz/. [p131]

X. Wu, X. Zhu, G. Wu, and W. Ding. Data mining with big data. *IEEE Transactions on Knowledge and Data Engineering*, 26(1):97–107, Jan 2014. ISSN 1041-4347. doi: 10.1109/TKDE.2013.109. URL https://doi.org/10.1109/TKDE.2013.109. [p130]

Miguel Ferreiro-Díaz
Department of Computer Science
SING Research Group, University of Vigo, Spain
CINBIO – Centro de Investigaciones Biomédicas, University of Vigo, Campus Universitario Lagoas-Marcosende, 36310 Vigo, Spain
miguel.ferreiro.diaz@uvigo.es

Tomás R. Cotos-Yáñez
Department of Statistics and Operations Research
SiDOR Research Group, University of Vigo
CINBIO – Centro de Investigaciones Biomédicas, University of Vigo, Campus Universitario Lagoas-Marcosende, 36310 Vigo, Spain
ORCiD: 0000-0002-7732-6565
cotos@uvigo.es

José R. Méndez
Department of Computer Science
SING Research Group, Galicia Sur Health Research Institute (IIS Galicia Sur), SERGAS-UVIGO, Spain

CINBIO – Centro de Investigaciones Biomédicas, University of Vigo, Campus Universitario Lagoas-Marcosende, 36310 Vigo, Spain
ORCiD: 0000-0002-1935-4760
moncho.mendez@uvigo.es

David Ruano-Ordás
Department of Computer Science
SING Research Group, Galicia Sur Health Research Institute (IIS Galicia Sur), SERGAS-UVIGO, Spain
CINBIO – Centro de Investigaciones Biomédicas, University of Vigo, Campus Universitario Lagoas-Marcosende, 36310 Vigo, Spain
ORCiD: 0000-0002-6050-373X
drordas@uvigo.es

Unidimensional and Multidimensional Methods for Recurrence Quantification Analysis with crqa

by Moreno I. Coco, Dan Mønster, Giuseppe Leonardi, Rick Dale and Sebastian Wallot

Abstract Recurrence quantification analysis is a widely used method for characterizing patterns in time series. This article presents a comprehensive survey for conducting a wide range of recurrence-based analyses to quantify the dynamical structure of single and multivariate time series and capture coupling properties underlying leader-follower relationships. The basics of recurrence quantification analysis (RQA) and all its variants are formally introduced step-by-step from the simplest auto-recurrence to the most advanced multivariate case. Importantly, we show how such RQA methods can be deployed under a single computational framework in R using a substantially renewed version of our **crqa** 2.0 package. This package includes implementations of several recent advances in recurrence-based analysis, among them applications to multivariate data and improved entropy calculations for categorical data. We show concrete applications of our package to example data, together with a detailed description of its functions and some guidelines on their usage.

Introduction

In the current article, we present the updated 2.0 version of the R package **crqa** to perform many variants of recurrence-based analyses (Coco and Dale, 2014), including some very recent developments for the treatment of multivariate and categorical data. Recurrence-based techniques allow the quantification of temporal structure and generalized autocorrelation properties of individual time series (Webber and Zbilut, 1994; Zbilut and Webber, 1992), the quantification of bivariate relationships, and coupling between two time series (Zbilut et al., 1998; Marwan and Kurths, 2002), as well as the quantification of multidimensional dynamics of multivariate time series (Wallot et al., 2016b; Wallot and Mønster, 2018). Recurrence-based techniques originate from the description and analysis of dynamical systems (Marwan et al., 2007; Marwan and Kurths, 2002) and have been widely applied to data from physics (Alex et al., 2015; Ambrożkiewicz et al., 2019; Donner and Thiel, 2007; Hilarov, 2017; Zolotova and Ponyavin, 2006), physiology (Marwan et al., 2002; Langbein et al., 2004; Thomasson et al., 2001; Mestivier et al., 2001; Mønster et al., 2016; Timothy et al., 2017), and psychology (Abney et al., 2014; Coco et al., 2018, 2016; Shockley et al., 2003; Wallot et al., 2019; Wijnants et al., 2012; Pagnotta et al., 2020), to name a few fields.

The real success of recurrence-based analyses has revolved around their power of capturing the dynamics of complex and non-stationary time series data and of time series exhibiting qualitatively different patterns along with their temporal evolution (Marwan et al., 2007). This is because recurrence-based analyses are model-free techniques that make few assumptions and hence are well suited for the analysis of complex systems. Moreover, recurrence-based analyses are versatile and can be applied to interval-scale data as well as nominal data, continuously sampled data, and inter-event data alike (Dale et al., 2011; Zbilut et al., 1998).

The new version of the **crqa** package features the integration of major developments in recurrence analysis, such as its extension to multidimensional data, as well as a key simplification of its design and a marked improvement of the underlying computational procedures. It includes useful new functions, including a tool for mining the parameter settings for continuous data and piece-wise computation of recurrence plots to mitigate the computational cost of long time series.

The remainder of this article is divided into two broad sections. In the first section, we provide a concise introduction to the core concepts of recurrence analysis from the simplest case of auto-recurrence of a unidimensional time series to the most complex case of multidimensional cross-recurrence, which is now integrated into the new version of the **crqa** package. In the second section, we showcase example applications of the different analysis methods using empirical data, hence providing a hands-on tutorial for how to use the different functions of the package.

Methodological background

In section 2.2.1, we briefly introduce the framework of recurrence quantification analysis (RQA hereafter) with the simplest case of a unidimensional time series. Then, in section 2.2.2, we discuss

how RQA can be applied to two different unidimensional times series. Finally, in sections 2.2.3 and 2.2.4 we explain how RQA methods can be extended to multidimensional data.

Recurrence quantification analysis (RQA)

The concept of recurrence is at the heart of all recurrence-based analyses (Marwan and Kurths, 2002; Trulla et al., 1996), which mostly apply to time series or sequenced data (but see Wallot and Leonardi (2018b)). As we will see, recurrences are often defined in terms of phase space coordinates, not directly in terms of the values of a single time series, but the concept is easily demonstrated using this case—a single time series or sequence—as a starting point. Loosely speaking, a recurrence is the repetition of a value in a sequence of data points. More precisely, recurrence in a time series x, with n data points $x_1, x_2, \ldots, x_n$ is defined as:

$$R_{ij} = \begin{cases} 1 & : x_i = x_j \\ 0 & : x_i \neq x_j \end{cases} \qquad i, j = 1, 2, \ldots n \tag{1}$$

The above equation defines the elements R_{ij} of the recurrence matrix $\mathbf{R}$ in terms of identical repetitions. Such a definition is useful for a nominal sequence, where there are categorical elements that are either identical or not, and so no meaningful 'distance' norm among categories can be defined (see section 2.3.3.1 for an application to text data). In order to define recurrences for continuous data, we need to establish a threshold parameter (or radius parameter) ε, which provides the width of a tolerance band in the chosen distance norm within which similar but not identical values in a time series are counted as recurrent:

$$R_{ij} = \begin{cases} 1 & : |x_i - x_j| \leq \varepsilon \\ 0 & : |x_i - x_j| > \varepsilon \end{cases} \qquad i, j = 1, 2, \ldots n \tag{2}$$

Setting a threshold is necessary in most cases for empirical data because such data feature intrinsic fluctuations as well as measurement error (Marwan et al., 2007). Using recurrences as defined above, we can convert any unidimensional time series x into a recurrence plot (RP), which is a two-dimensional portrait of its dynamics expressed through its recurrence characteristics.

Figure 1 shows some examples of time series of various complexity (i.e., a sinusoidal, a chaotic attractor, and white noise) and their associated RPs, given some value for the threshold parameter ε.

If a time series x constitutes the one-dimensional measurement of an underlying multidimensional system, and the dynamics of the underlying dimensions are co-dependent, then these underlying dimensions can be recovered via the method of time-delayed embedding from the unidimensional time series (Packard et al., 1980; Takens, 1981). In these cases, the time series x is delayed (or lagged) by a certain number of data points, τ, and the number of times such delays are applied to x depends on an embedding dimension parameter m. The time-shifted copies of x, x_τ, $x_{2\tau}$,…,$x_{(m-1)\tau}$, can now be integrated into a single m-dimensional phase space, which shows the recovered multidimensional dynamics behind the measured unidimensional time series (Figure 2).

If the time series data comes from a multidimensional system, embedding the data into a higher dimensional phase space before the computation of an RP will improve the quantification of the dynamics of the systems from which that time series was recorded (Marwan et al., 2007). Now, with embedded data, recurrences are defined not in terms of the individual values of the original, unidimensional time series x but in terms of coordinates in m-dimensional phase space. It is possible to construct a total of $N = n - (m-1)\tau$ points in phase space, with the following coordinates:

$$\begin{aligned} \mathbf{X}_1 &= (x_1, x_{1+\tau}, x_{1+2\tau}, \ldots, x_{1+(m-1)\tau}) \\ \mathbf{X}_2 &= (x_2, x_{2+\tau}, x_{2+2\tau}, \ldots, x_{2+(m-1)\tau}) \\ &\vdots \\ \mathbf{X}_k &= (x_k, x_{k+\tau}, x_{k+2\tau}, \ldots, x_{k+(m-1)\tau}) \\ &\vdots \\ \mathbf{X}_N &= (x_N, x_{N+\tau}, x_{N+2\tau}, \ldots, x_n) \end{aligned} \tag{3}$$

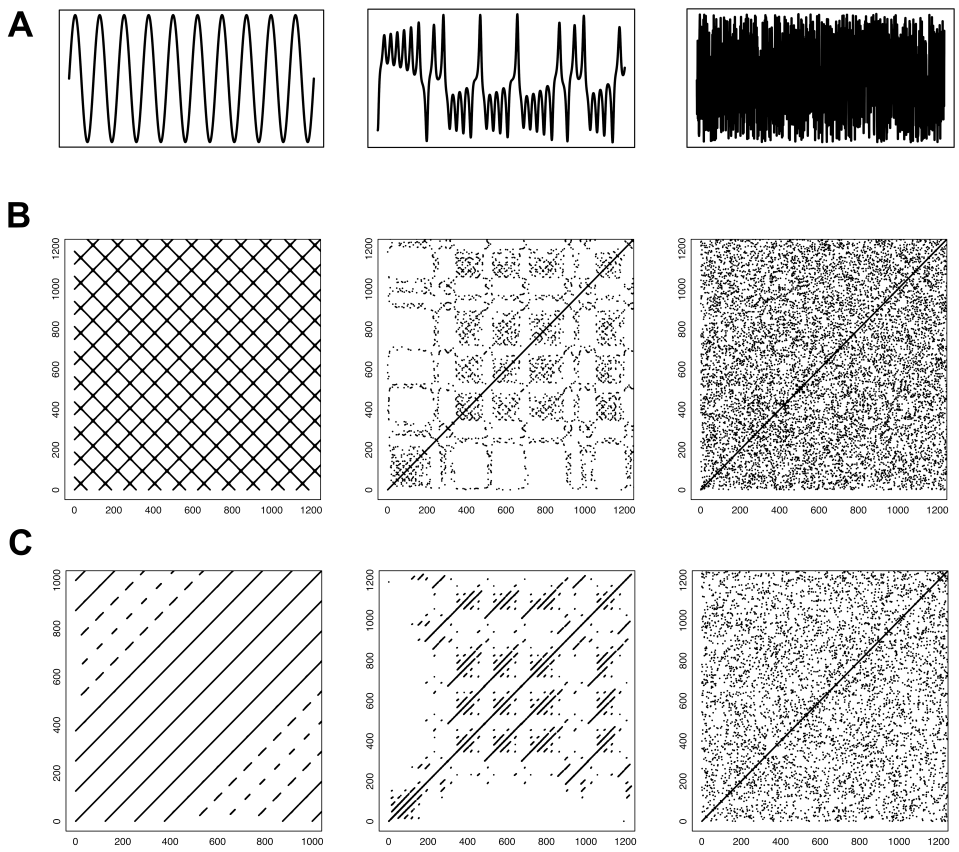


Figure 1: The three-time series in the first row (A) — a periodical sine wave, one of the dimensions of the chaotic Lorenz attractor and a white noise signal — were subjected to recurrence analysis without dimensional embedding. That is, the recurrence plots depict recurrences based on the values of the one-dimensional time series (B). The third row (C) shows their associated recurrence plots with dimensional embedding (the sine wave was embedded in 2 dimensions, the Lorenz attractor and the white noise signal in 3 dimensions).

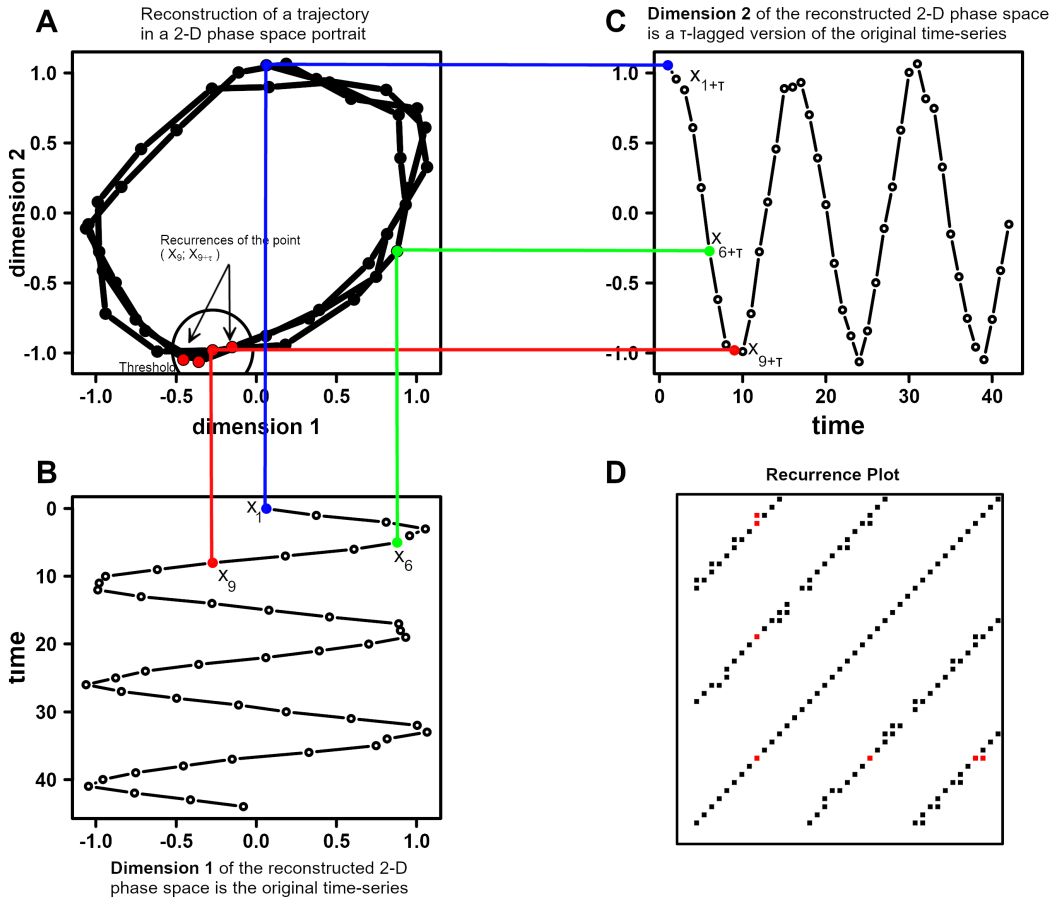


Figure 2: Graphical exemplification of the embedding method. A unidimensional time series (B)—here represented with time running along the y-axis—is embedded in two-dimensional phase space (A) by choosing as the second dimension (C) of every point in panel B the τ-lagged value (here $\tau = 3$) of the very same original unidimensional time series (B). After selecting an appropriate value of the threshold (or radius) parameter (here $\varepsilon = 0.25$), recurrence analysis can be applied to the trajectory in the two-dimensional phase space (A), and a recurrence plot is generated (D). In the recurrence plot (D), the recurrence points around the coordinates of $(X_9; X_{9+\tau})$ are highlighted in red color.

In terms of points in the m-dimensional phase space, the elements of the recurrence matrix are then given by

$$R_{ij} = \begin{cases} 1 & : ||\mathbf{X}_i - \mathbf{X}_j|| \leq \varepsilon \\ 0 & : ||\mathbf{X}_i - \mathbf{X}_j|| > \varepsilon \end{cases} \qquad i, j = 1, 2, \ldots N \tag{4}$$

where $||\cdot||$ is a distance norm in the m-dimensional phase space and ε is the threshold used to determine whether the points are close enough in phase space to be considered recurring or not. Examples of embedding and the resulting RPs are shown in Figure 1, lower panel, and in Figure 2, panel D. Note, however, that in the process of phase space reconstruction, the number of coordinates available in phase space is less than the number of data points in the original time series, the difference being equal to $n - N = \tau(m-1)$. Moreover, the resulting phase space portraits do not exactly reflect the true underlying multidimensional dynamics but are isomorphic to them (Garland et al., 2016). For continuous time series (i.e., not categorical), the delay τ, the embedding dimension m and the radius ε are usually unknown and have to be estimated from the data. τ can be obtained by examining the average mutual information function (AMI) of x (Fraser and Swinney, 1986), and the false-nearest-neighbor function (FNN) of x, given some value for τ, can, in turn, be used to determine m (Kennel et al., 1992). The radius ε is then chosen to achieve the desired proportion of recurrence points that is expected given the type of data at hand (Webber Jr and Zbilut (2005); see Wallot and Mønster (2018) for practical information on parameter estimation). The **crqa** package implements functions to carry out parameter estimation semi-automatically.

RPs are a very useful visualization to qualitatively explore the dynamics of a time series. However, their main advantage is that they provide the basis to quantify the dynamics of a time series based on the patterns of recurrence points found in the plot (Zbilut and Webber, 1992). There are various measures that can be computed from RPs. Here, we briefly describe a selection of measures, including the most common ones, that are implemented in the new version of the **crqa** package.

The most basic measure is the recurrence rate (RR) or percentage recurrence, which is defined as the sum of all recurrence points in an RP divided by the area of that RP. RR provides a measure for how many individual values of a time series—or its phase space coordinates—recur over time:

$$\mathrm{RR} = \frac{1}{N^2} \sum_{i,j=1}^{N} R_{ij} \tag{5}$$

All other measures characterize dynamics by exploiting the patterns of recurrences along with the vertical and diagonal structures of the RP (see Table 1 for a concise summary of the measures). In particular, measures based on diagonal-line structures reflect repetitions of the trajectories of the time series, whereas measures based on vertical-line structures focus on the states during which a time series slows down its dynamics. The entropy of the time series can also be computed on the basis of diagonal and vertical line structures of the RP. In version 2.0 of the **crqa** package, we include a novel entropy measure, which is based on the distribution of the areas of the rectangular structures in an RP generated from categorical time series. This measure provides a more accurate estimation of entropy over the classic diagonal-line entropy for categorical time series that predominantly evolve in terms of changes of states (Leonardi, 2018).

Cross-recurrence quantification analysis (CRQA)

Until now, we focused on quantifying the dynamics of a system by way of recurrences of a single time series. However, the concept of recurrence can be extended to that of cross-recurrence, which extends the univariate recurrence analysis to a bivariate analysis technique that allows quantification of the temporal coupling properties or similarity of two time series (Zbilut et al., 1998; Marwan et al., 2007). In other words, cross-recurrence is the recurrence of a value in a time series x, with data points $x_1, x_2, \ldots, x_n$ with the values of a time series y, with data points $y_1, y_2, \ldots, y_n$. Formally:

$$\mathrm{CR}_{ij} = \begin{cases} 1 & : |x_i - y_j| \leq \varepsilon \\ 0 & : |x_i - y_j| > \varepsilon \end{cases} \qquad i, j = 1, 2, \ldots n \tag{6}$$

As for Equation 2, a threshold parameter ε is applied to identify similar, but not necessarily identical, values that are recurrent across the two time series. As in the univariate case, this parameter can be set to values closer to 0, forcing cross-recurrences to be identical values, such as between two nominal (categorical) sequences. Also, in the case of cross-recurrence analysis, the two time series x and y can be embedded before computing the cross-recurrence plot (CRP):

Measure	Abbreviation	Definition
Recurrence Rate	RR	$\frac{1}{N^2}\sum_{i,j=1}^{N} R_{ij}$
Determinism	DET	$\sum_{l=l_{\min}}^{N} lP(l) \Big/ \sum_{l=1}^{N} lP(l)$
Average Diagonal Line Length	L	$\sum_{l=l_{\min}}^{N} lP(l) \Big/ \sum_{l=l_{\min}}^{N} P(l)$
Maximum Diagonal Line Length	maxL	$\max(\{l_i\}_{i=1}^{N_l}), \quad N_l = \sum_{l\geq l_{\min}} P(l)$
Diagonal Line Entropy	ENTR	$-\sum_{l=l_{\min}}^{N} p(l)\log p(l)$
Laminarity	LAM	$\sum_{v=v_{\min}}^{N} vP(v) \Big/ \sum_{v=1}^{N} vP(v)$
Trapping Time	TT	$\sum_{v=v_{\min}}^{N} vP(v) \Big/ \sum_{v=v_{\min}}^{N} P(v)$
Categorical Area-based Entropy	catH	$-\sum_{a>1}^{N_a} p(a)\log p(a)$

Table 1: Summary and definition of RQA measures. Here, l is some diagonal line length on the recurrence plot, i.e., the number of diagonally adjacent recurrence points; $P(l)$ is the histogram or frequency distribution of such diagonal line lengths; $p(l)$ is the probability of some diagonal line length; v is some vertical line length on the recurrence plot, i.e., the number of vertical adjacent recurrence points; $P(v)$ is the histogram or frequency distribution of such diagonal line lengths; $l_{\min}$ and $v_{\min}$ are the minimum diagonal and vertical line lengths included in the measures (≥ 2); a is the value of the area of a rectangular recurrence block generated in a categorical recurrence analysis; $p(a)$ is the probability of the recurrence blocks of area a.

$$\mathrm{CR}_{ij} = \begin{cases} 1 & : ||\mathbf{X}_i - \mathbf{Y}_j|| \leq \varepsilon \\ 0 & : ||\mathbf{X}_i - \mathbf{Y}_j|| > \varepsilon \end{cases} \qquad i, j = 1, 2, \ldots N \tag{7}$$

Commonly, it is expected that x and y have the same number of data points, and that the delay and embedding dimension parameters, τ and m, have to be the same too (see Wallot and Mønster (2018); Wallot and Leonardi (2018a) for practical aspects of parameter estimation)[1]. The recurrence measures obtained from cross-recurrence quantification analysis are calculated in the same way as for the univariate recurrence quantification analysis (see Table 1). However, the values for cross-recurrence now reflect the coupled dynamics of the two time series (Shockley et al., 2002) rather than the dynamics of an individual time series in univariate RQA. There is a key difference between RPs and CRPs. RPs always have recurrence points all along the main diagonal line of the plot (so-called line of identity, LOI), because a time series is by definition recurrent with itself at lag 0, which is what recurrences along the main diagonal reflect (i.e., $x_i = x_j$ when $i = j$). This is not necessarily the case for CRPs as two time series are not necessarily synchronized (x_i need not be the same as y_j when $i = j$). If the time series are not synchronized, cross-recurrences around the main diagonal are absent or sparse. The presence of a full LOI in a CRP implies that the dynamics of the two time series are identical or that they have a strictly linear relationship to each other.

Multidimensional recurrence quantification analysis (MdRQA)

Multidimensional recurrence quantification analysis (MdRQA) is one of the recent extensions of RQA (Wallot et al., 2016b) that is now also available in the **crqa** package. MdRQA allows the analysis of multidimensional time series $\mathbf{z}$ with samples $\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_n$, where each point (sample) in $\mathbf{z}$ has d dimensions:

[1] It is possible for x and y to be different lengths, and so to produce a rectangular CRP, but this is very rare in practice and it introduces some complications in computing synchrony measures

$$\mathbf{z}_k = (z_{k,1}, z_{k,2}, \ldots, z_{k,d}) \tag{8}$$

Here, recurrences are defined on the d-dimensional coordinate space made up of the points of $\mathbf{z}$:

$$R_{ij} = \begin{cases} 1 & : ||\mathbf{z}_i - \mathbf{z}_j|| \leq \varepsilon \\ 0 & : ||\mathbf{z}_i - \mathbf{z}_j|| > \varepsilon \end{cases} \qquad i, j = 1, 2, \ldots n \tag{9}$$

Like their unidimensional counterparts, multidimensional time series can be embedded into a higher dimensional space. The logic of estimating the delay and embedding dimension parameters, τ and m, are the same as with univariate RQA (Wallot, 2017; Wallot and Leonardi, 2018a). However, suppose one has multivariate time series and wants to estimate embedding parameters for those time series. In that case, one should use the multivariate embedding functions which are also provided with the new version of the **crqa** package, because they provide superior estimates of embedding parameters for multidimensional time series (Wallot and Mønster, 2018).

Depending on the underlying data, MdRQA can, in principle, be used for two different purposes. On the one hand, MdRQA can be used to quantify a multidimensional construct, such as physiological arousal, by simultaneously looking at its different measurable dimensions (e.g., heart rate, breathing, and body temperature). This would be the multivariate version of how univariate RQA quantifies the dynamics of a single unidimensional time series (e.g., breathing alone). On the other hand, MdRQA can be used to examine the shared dynamics of multiple individual time series, such as, for example, three electrodermal signals measured from three members of a team performing a collaborative task. Here, MdRQA variables would be interpreted as capturing higher-order inter-correlative properties between the three signals at the level of the group (Wallot et al., 2016b).

In either case, one has to make a decision about whether to normalize the different dimensions of the time series or not. If each dimension of the multidimensional time series is normalized, for example, z-scored, it would effectively give each time series equal weight for the definition of recurrence. In particular, if one does not know how the different time series interact, or if they are measured on different scales without regard to their potential effects on one another, then normalization is recommended. Otherwise, the risk is to assign a greater weight to the time series bearing greater variance. If one is certain that the values of each dimension of the multidimensional time series are already properly scaled with regard to each other, such as when simultaneously analyzing the three dimensions of the Lorenz system (Lorenz, 1963), then one needs not—and perhaps should not—normalize the dimensions.

Multidimensional cross-recurrence quantification analysis (MdCRQA)

Multidimensional cross-recurrence quantification analysis (MdCRQA) extends MdRQA in the same way that CRQA extends RQA. Effectively, MdCRQA allows for the computation of cross-recurrences between two multidimensional time series $\mathbf{x}$ and $\mathbf{y}$ (Wallot, 2019), where cross-recurrences are defined between the two d-dimensional coordinate spaces between the points of $\mathbf{x}$ and $\mathbf{y}$:

$$\mathbf{x}_i = (z_{i,1}, z_{i,2}, \ldots, z_{i,d}) \tag{10}$$
$$\mathbf{y}_j = (z_{j,1}, z_{j,2}, \ldots, z_{j,d}) \tag{11}$$

$$\mathrm{CR}_{ij} = \begin{cases} 1 & : ||\mathbf{x}_i - \mathbf{y}_j|| \leq \varepsilon \\ 0 & : ||\mathbf{x}_i - \mathbf{y}_j|| > \varepsilon \end{cases} \qquad i, j = 1, 2, \ldots n \tag{12}$$

It is important that the different dimensions of the two multivariate time series enter the analysis in the same order. For the example of physiological arousal, if the heart rate is the first dimension in $\mathbf{x}$, breathing is the second dimension in $\mathbf{x}$, and body temperature is the third dimension in $\mathbf{x}$, then heart rate also needs to be the first dimension in $\mathbf{y}$, breathing needs to be the second dimension in $\mathbf{y}$, and accordingly body temperature needs to be the third dimension in $\mathbf{y}$. Otherwise, the resulting MdCRQA measure will not be interpretable.

Unidimensional and multidimensional cross-recurrence analysis can also be performed in a time-dependent manner. This so-called windowed cross-recurrence analysis is conceptually very similar to windowed cross-correlation analysis as in Boker et al. (2002), and it can be used to track how cross-recurrence changes over the time course. To that end, one simply partitions the time series of interest into a number of overlapping or non-overlapping sub-series and calculates CRPs for each of the sub-series. For each CRP, i.e., each sub-series of the original time series, the recurrence measures are calculated, which allows tracking changes in cross-recurrence over time.

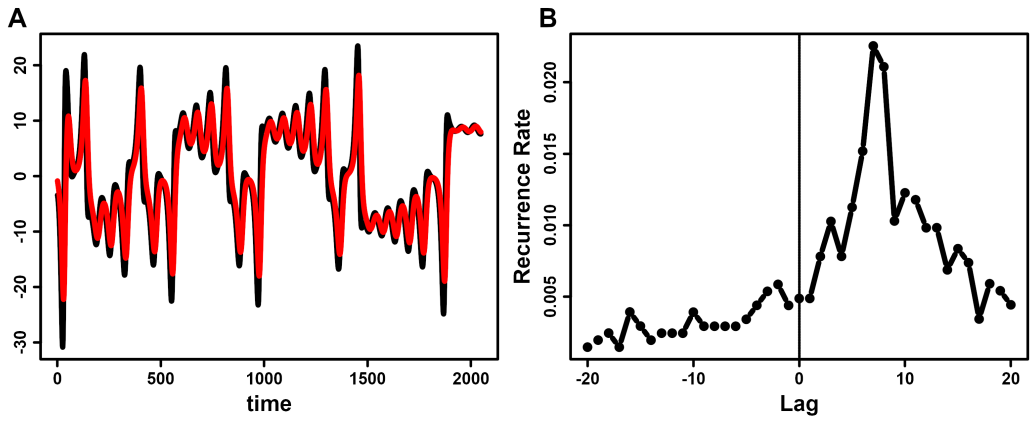


Figure 3: Illustration of DCRP. Time series of two of the three dimensions from the Lorenz system, one in red, the other one in black (A). After computation of their CRP, one can define their diagonal cross-recurrence profile (DCRP) in order to quantify their time-lagged coupling properties (B). As can be seen, the two time series are most strongly coupled at around the lags of 5 to 9, where a peak in the DCRP can be observed.

The diagonal cross-recurrence profile (DCRP)

Finally, from cross-recurrence plots of either two unidimensional time series (i.e., CRQA) or two multidimensional time series (i.e., MdCRQA), it is possible to extract the diagonal cross-recurrence profiles (DCRPs) and use them to capture leader-follower-relationships (Dale et al., 2011; Marwan et al., 2007). To that end, one has to specify a window size w for the number of lags on the recurrence plot that one wants to investigate. For example, a window-size of 10 data-points, i.e., $w = 10$ would span recurrences of $\pm$ 10 diagonals from the LOI. Specifically, this procedure determines the cross-recurrence rates of each diagonal, CR_w, by summing up all cross-recurrence points that fall along such diagonal and divide them by their length:

$$\mathrm{CR}_w = \frac{1}{N-w} \sum_{j-i=w}^{N-w} R_{i,j} \tag{13}$$

The DCRP permits quantification of the recurrence rate over different relative lags between two time series. If the peak of cross-recurrences falls along the central diagonal, the line of synchronization (LOS), then this suggests strong coupling at lag 0 between the two time series. If the peak of cross-recurrences falls instead on one of the diagonals off the LOS, it indicates that the dynamics of one time series follow the dynamics of the other time series by some lag equal to that diagonal position (Figure 3).

Note, however, interpreting the lags in terms of the sampling rate of the underlying measured time series only applies to CRPs based on unembedded (often categorical) time series. If the time series are embedded, this means that the observed lag is based on coordinates that are made up of several data points from the respective original time series, and hence introduces a degree of uncertainty with regard to the precise time interval of the lag when one tries to map particular recurrence points back to data points of the original time series. In addition, the leader-follower interpretation of the lags cannot be granted the status of a causal interpretation. For example, a parent can deliberately 'lag' their behavior behind that of their child. In such a case, it would be odd to say definitively that the child's behavior is causing the parent's behavior. For this reason, the DCRP has to be interpreted with caution and is best treated as a general description of relative temporal relationships.

Package design

The **crqa** package is available from the Comprehensive R Archive Network (CRAN) at `https://cran.r-project.org/web/packages/crqa/index.html`. The software is written in R with the exception of the `spdiags` function, which contains a section written in Fortran. The main function is `crqa`, which takes its name from the package. This function performs all types of recurrence quantification analyses discussed in the previous section, and it returns the actual recurrence plot, along with the measures listed in Table 1 extracted from it. Here is how to call this primary function, with arguments and defaults:

Argument	Description	Usage Notes
ts1 and ts2	The input time series, either unidimensional or multidimensional time series	If auto-recurrence is used (see method = rqa), then t1 and t2 should be the same time series.
delay	A constant indicating the number of time points used to lag the time series	It corresponds to the τ of the equations.
embed	A constant indicating the number of embedding dimensions applied to the time series	It corresponds to the m of the equations.
rescale	Whether the distance matrix on which recurrence is evaluated should be rescaled and how	If rescale = 0, keep the distance matrix as is; if rescale = 1, rescale the distance matrix to its mean; if rescale = 2, rescale to distance matrix to its maximum value.
radius	A constant used to decide whether the distance between two points is small enough to be considered recurrent	For categorical time series, the radius needs to be set at values smaller than 1. For continuous time series, the value of the radius needs to tailored to the type of data and its range.
normalize	Normalize the time series	if normalize = 0, keep the time series at their original scale; if normalize = 1, normalize the time series to unit interval; if normalize = 2, z-score the time series.
mindiagline	The minimum number of contiguous points along the diagonals to consider the system into a recurrent state	The default value is usually 2, as it takes a minimum of two points to define any line.
minvertline	The minimum number of contiguous points along the vertical lines to consider the system into a recurrent state	The default value is usually 2, as it takes a minimum of two points to define any line.
tw	The Theiler window parameter	It defines the number of diagonals off the line of identity that are excluded from recurrence quantification.
whiteline	A logical flag to calculate (TRUE) or not (FALSE) empty vertical lines.	The default is FALSE, as the calculation of such lines adds on the time of computation.
recpt	A logical flag indicating whether measures of cross-recurrence are calculated directly from a recurrent plot (TRUE) or not (FALSE)	It is mostly useful if the user wants to compute joint-recurrence analysis. The RP or CRP is supplied in place of ts1, and ts2 needs to be assigned as NA.
side	A string indicating the side of the recurrence plot on which recurrence measures should be calculated	For side = upper, recurrence measures are calculated on the upper triangle of the RP, for side = lower on the lower triangle of the RP, for size = both on the full RP. Note, the line of identity is automatically excluded for upper and lower setting.
method	A string to indicate the type of recurrence analysis to perform	For method = rqa, Auto-recurrence is calculated, i.e., a unidimensional series; for method = crqa, cross-recurrence is calculated; for method = mdcrqa, multidimensional recurrence is calculated. Note, the default value is crqa.
metric	A string to indicate the type of distance metric used	To see the list of all other possible metrics, see the help for the rdist function. Note, the default is euclidean.
datatype	A string (continuous or categorical) to indicate the nature of the data type	If the time series contain categorical information, it will automatically be recoded into a continuous integer-based time series with a warning sent to the user.

Table 2: Overview of the arguments that can be used in crqa to set up the recurrence quantification analysis.

```
crqa(ts1, ts2, delay, embed, rescale = 0, radius, normalize = 0, mindiagline = 2,
     minvertline = 2, tw = 0, whiteline = FALSE, recpt = NULL, side = "upper",
     method = "crqa", metric = "euclidean", datatype = "continuous")
```

Several arguments in this function allow the user to refine aspects of the computation. For example, the user can modify the metric to obtain the distance matrix when estimating recurrence or the settings of thresholds to accept contiguous points as recurring (see Table 2 for the list of arguments of crqa with a brief explanation of each). drpfromts can be used to obtain the diagonal cross-recurrence profile, and this function is built using the crqa at its core. In fact, most arguments stay exactly the same, and we will illustrate the additional arguments that are specific to this function when describing it. Likewise, the functions wincrqa and windowdrp are built on the main function crqa and are used to compute windowed cross-recurrence. The package also contains functions, such as optimizeParam, to automatically estimate the settings for the three main parameters of RQA analyses, i.e., radius ε, delay τ, and embedding dimension m, for continuous measures. As recurrence quantification analysis is heavy on memory requirements, the package features a function (piecewiseRQA) which can be used to break down the analysis of long time series into smaller and more manageable chunks that are computationally more tractable. Finally, the package provides the user with functions to simulate data from classic examples of dynamical systems, such as the Lorenz (lorenzattractor) or categorical series with different distributions (simts). In what follows, we briefly describe the data available with the package. These data are used to illustrate the different functionalities of the crqa package.

Related Packages

A search of CRAN shows very few other packages offering alternative methods to compute recurrence quantification analysis. In particular, **tseriesChaos**, **nonlinearTseries**, and **RHRV**. The function `recurr` in **tseriesChaos** computes a recurrence plot for a (section of) a single unidimensional time series, but it does not compute any derived measures from the plot characterizing the dynamics of the system, nor does it handle cross recurrence plots or any of the many extensions to simple recurrence plots provided by our **crqa** package. Going a little further, the function `rqa` in **nonlinearTseries** returns key measures from the recurrence plots (e.g., recurrence rate) and quick visualization of the recurrence plot (available also with the function `RecurrencePlot`). Finally, the function `RecurrencePlot` in **RHRV** is simply a wrapper built on the function in **nonlinearTseries** with the same name but specifically tailored to electrocardiogram data. The functions available in **nonlinearTseries** provide very basic functionality in terms of the range of metrics available and optimization routines. They do not allow the user to examine diagonal structures for leader-follower analyses, nor explore the evolution of recurrence rate using windowed methods. All such features are integrated into **crqa**, which, to the best of our knowledge, is the most comprehensive statistical package to perform recurrence quantification analysis in R.

Data

Different types of categorical and continuous time series, both unidimensional and multidimensional, are available with the package. The command `load(crqa)` will load the data into the R workspace. In particular, we include a nursery rhyme "The wheels on the bus" by Verna Hills to illustrate the most basic recurrence quantification analysis. This text is a vector of 120 strings (i.e., the words of the song), and as it is extremely simple and highly repetitive, it makes it a very good example to illustrate the core concept of recurrence. Then, we move on to cross-recurrence quantification analysis and include in the data object of the package eye-tracking data from the study by Richardson and Dale (2005). In this study, a narrator describes the characters of a TV series (Friends) to a listener, who will have to later answer some comprehension questions about them, while their eye-movement is co-registered. Here, we use a single trial of this study, which is stored as a data frame of 2,000 observations of six possible screen locations that are looked at by the narrator and the listener. These are numerically coded from 1 to 6, representing a 2x3 visual grid[2]. This data will also be used to illustrate diagonal and windowed cross-recurrence. Finally, we illustrate multidimensional cross-recurrence analysis using the hand movement data from the study by Wallot et al. (2016a). In this study, dyads were instructed to cooperate in a complex LEGO joint construction task under different conditions, while their hand movements and heart rates were co-registered. Again, we select only a single trial of hand-movement from the turn-taking condition. The data frame comprises 5,799 observations from two participants (P1 and P2) for the dominant (_d) and non-dominant (_n) hand.

Using the crqa package

RQA

As explained in section 2.2.1, RQA entails computing the auto-recurrence of a unidimensional time series. In the context of the nursery rhyme, we expect clear phases of recurrence to emerge because the words greatly repeat. In order to run this analysis, we use the main function `crqa` and specify in the argument `method` that we are running an RQA analysis (`method = "rqa"`). As the data that we use is categorical, we need to specify in the argument `datatype` that the nature of the data is categorical (`datatype = "categorical"`). This will automatically recode the categorical states of the series (i.e., the words) into unique numerical integers so that recurrence can be computed using a `radius` that has to be smaller than 1 (e.g., `radius = 0.01`) so that we can capture the recurrence of identical words. For categorical RQA, the delay and embedding dimension have to be set to 1 (`delay = 1; embed = 1`[3]`.`). We also need to set the Theiler window parameter to 1 (`tw` = 1) so that we can exclude the LOI from all recurrence measures. Finally, the same unidimensional time series has to be input both as `ts1` and `ts2` to obtain its auto-recurrence.

[2]There are two more states, 10 and 11, to indicate when the listener or the narrator blinked or looked outside of the screen or to identify possible blinks. They are coded with a different number so that these two states will not recur when the radius is set near 0.

[3]If embedding dimension is set to higher values,this becomes equivalent to doing recurrence on n-grams,where $m = n$. In fact,this interpretation of categorical recurrence creates bridges to traditional natural language processing,summarized in Dale et al. (2018)

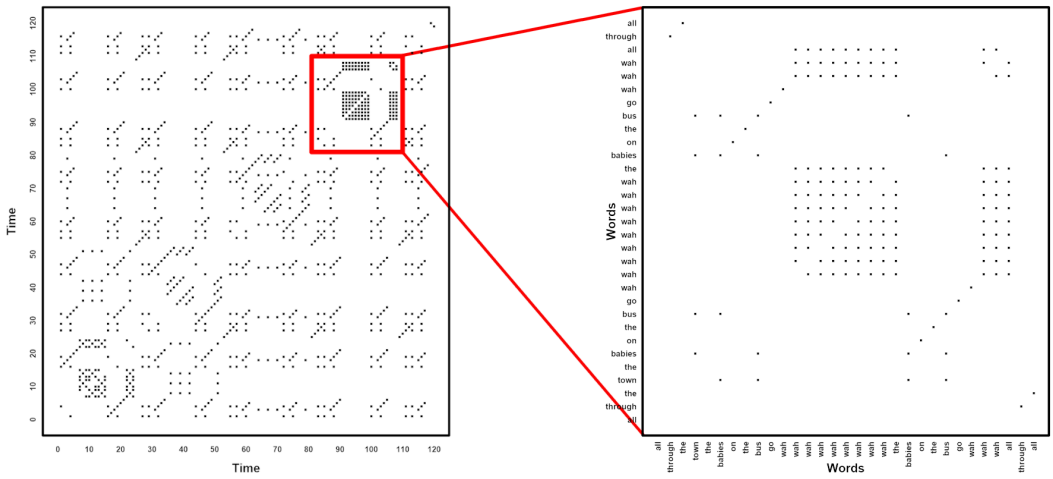


Figure 4: Recurrence quantification of the nursery rhyme 'wheels on the bus'. On the left panel, we show the full recurrence and the red box indicates the region of RP that is zoomed in on the right panel.

```
res <- crqa(text, text, delay = 1, embed = 1, rescale, radius = 0.01, normalize,
            mindiagline, minvertline, tw = 1, whiteline, recpt, side, method = "rqa",
            metric = "euclidean", datatype = "categorical")
```

We can use the plotting function `plotRP` to visualize the resulting recurrence plot. This function provides some basic arguments to change the size of the points in the plot (`pcex`), the color (`cols`), or their type (`pch`), which are taken verbatim from the generic `plot` function.

```
RP <- res$RP
parC <- list(unit = 10, labelx = "Time", labely = "Time", cols = "black", pcex = .5,
             pch = 15, las = 0, labax = seq(0, nrow(RP), 10),
             labay = seq(0, nrow(RP), 10))
plotRP(RP, parC)
```

In order to get a closer understanding of recurrences, we zoom in into a segment of the text, re-run the `crqa()` function, and visualize it (Figure 4). We add the labels of the axes (x,y), print the words vertically using the `las` argument, and decrease the temporal `unit` argument to print each individual word on the axes.

```
text_zoom <- text[81:110]
ans_zoom <- crqa(text_zoom, text_zoom, delay, embed, rescale, radius, normalize,
                 mindiagline, minvertline, tw, whiteline, recpt, side, method, metric,
                 datatype)
RP <- ans_zoom$RP
parC$labay <- parC$labax <- text_zoom
parC$las <- 2
parC$unit <- 1
parC$labelx <- parC$labely <- "Words"
plotRP(RP, parC)
```

As it can be clearly seen in Figure 4, the bulk of recurrence in that portion is driven by the repeated use of the single word *wah*. When looking at a few measures associated with the RP, we observe an overall determinism of 85.4%, which implies that the system is fairly repetitive, an average diagonal line length of 3.88, which implies that on average, there are sequences of four words that repeat, and a maximum diagonal length of 9, which means that the longest sequence repeating is made of 9 words. Some measures such as determinism or the average diagonal line length depending on the setting of the argument `mindiagline` (equivalent to $l_{\min}$ in Table 1). The default value of this argument is 2, as two contiguous points form a line, but it can depend on the type of data (e.g., words vs. eye-movement) or the sample rate at which it is acquired (55 Hz vs. 1,000 Hz). For example, if we have acquired data at 1,000 Hz, we would practically have one data point every 2 ms. This means that if we use a default `mindiagline` of 2, we would be considering as lines any states that contiguously repeat over a 4 ms window. This value would certainly be unrealistic for some type of responses that unfolds over a longer period of time (e.g., an eye-movement fixation lasts for an average of 200 ms).

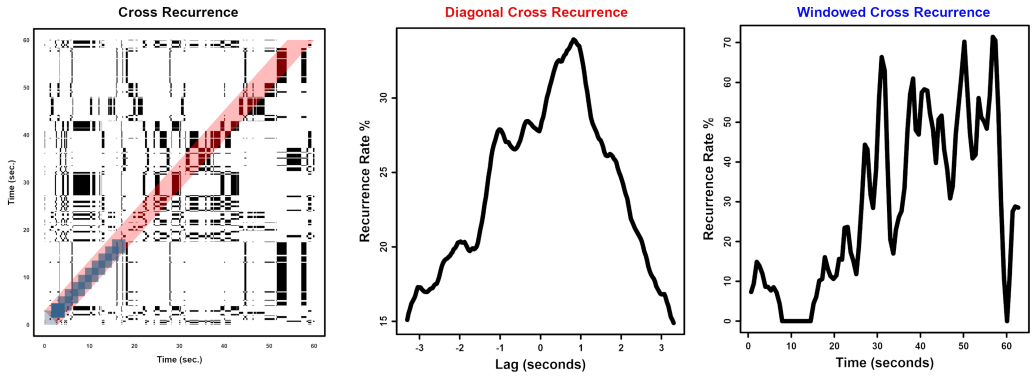


Figure 5: Cross-recurrence using a single trial of eye-movement data measured on a listener and a narrator. On the left panel, we visualize the full cross-recurrence plot. The transparent red band represents the lags around the line of coincidence that have been used to calculate the diagonal cross-recurrence plot (center panel), whereas the transparent blue squares within it are the overlapping windows used to compute the windowed cross-recurrence profile (right panel).

CRQA: `crqa`

As already explained in section 2.2.2, cross-recurrence is an extension of auto-recurrence to two different unidimensional time series. In order to run a cross-recurrence analysis with the **crqa** package, we simply need to change the `method` argument to `method = "crqa"`. Here, we illustrate its use through two time series of eye-movement data explained in section 2.3.2. Also, we input two different time series of eye-movement data, `narrator` and `listener`, rather than just one. An optional argument that is available in the `crqa` function is the `side` (upper, lower, or both) of the recurrence plot on which recurrence measures are computed. This may be useful, for example, for researchers interested in leader-follower dynamics (see Figure 5, left panel, for the visualization of the cross-recurrence plot).

Diagonal-CRQA: `drpfromts`

In section 2.2.2, we explained what diagonal cross-recurrence is and how it can be used. In the `crqa` package, this measure is computed by the function `drpfromts`, which utilizes the same arguments of the main `crqa` function plus an additional argument, `windowsize`, to define the number of lags (or diagonals) around the line of synchronization (LOS) of the CRP over which recurrence rate is computed. In the example visualized in Figure 5, center panel, we have chosen a window of 100 lags, which spans about ± 3 seconds around the LOS. We can clearly see that the peak recurrence is shifted by ≈ 1 second from the LOS, i.e., lag 0. This reflects the time taken by the listener to look at the same panel the narrator was looking at—namely, about 1 second for a listener to "catch up" to the speaker.

```
res <- drpfromts(narrator, listener, windowsize = 100, radius = 0.001,
                 delay = 1, embed = 1, rescale = 0, normalize = 0, mindiagline = 2,
                 minvertline = 2, tw = 0, whiteline = F, recpt = F, side = "both",
                 method = "crqa", metric = "euclidean", datatype = "continuous")
```

Windowed-CRQA: `windowdrp`

Windowed cross-recurrence captures the evolution of recurrence rate over time. In the context of the eye-movement data, this measure reflects how consistently listener and narrator are looking at the same scene location at any given point in time. More importantly, the windowed methodology reveals how recurrence changes over time (refer to section 2.2.5 for more details). We use the function `windowdrp` to compute this measure, which again shares the same arguments of `crqa`, plus three more that are specific to it. In particular, we have to set: (a) the size of the window that slides over the time-course, e.g., `windowsize = 100`, (b) the step that we want this window to move, e.g., `windowstep = 20`, and (c) the number of lags[4] within the window of interest over which recurrence rate is computed, e.g., `lagwidth = 50`. In Figure 5, right panel, we observe that the recurrence rate grows over time, which means that the narrator and the listener tend to look more and more at the same panels as the trial progresses.

[4]Note, that the number of lags cannot be greater than the size of the window.

```
res <- windowdrp(narrator, listener, windowstep = 20, windowsize = 100, lagwidth = 50,
                 radius = 0.001, delay = 1, embed = 1, rescale = 0, normalize = 0,
                 mindiagline = 2, minvertline = 2, tw = 0, whiteline = F, side = "both",
                 method = "crqa", metric = "euclidean", datatype = "continuous")
```

MdCRQA

Finally, we compute multidimensional cross-recurrence by setting the `method` argument, i.e., `method = "mdcrqa"`. Note, in order to compute multidimensional recurrence, the user needs to provide the same data frame as input to both `ts1` and `ts2`. This method is also available for `drpfromts` and `windowdrp`. Applied to the hand movement data, we first restructure the data of the two participants into independent sets. We re-use the same parameter settings of Wallot et al. (2016a) to compute MdCRQA and leave all other arguments with their default values.

```
P1 <- cbind(handset$P1_TT_d, handset$P1_TT_n)
P2 <- cbind(handset$P2_TT_d, handset$P2_TT_n)
res <- crqa(P1, P2, delay = 5, embed = 2, rescale = 0, radius = 0.1,
            normalize = 0, mindiagline = 10, minvertline = 10, tw = 0,
            whiteline, recpt, side, method = "mdcrqa", metric, datatype)
```

Breaking down the computation: `piecewiseRQA`

Often, researchers are interested in time series which contain several thousands of observations. Sometimes the dimensionality of these time series can be reduced without losing too much information, such as by down-sampling. This strategy may not always be possible or serve the researcher's purpose. Recurrence quantification analysis can require more RAM than is available in standard laptops or personal workstations, making it nearly impossible to run. In the new version of the **crqa** package, we provide the user with the `piecewiseRQA` function, which can be used to compute all different variants of recurrence quantification analysis described above on long time series. Conceptually, this function divides the time series into blocks, obtains a recurrence plot for each individual block, and then fills the original recurrence plot with all such sub-blocks, before computing the measures.[5]

For example, if we are handling a time series of 10,000 observations, we could divide it into 10 blocks of 1,000 observations each. `piecewiseRQA` has exactly the same arguments that we have already encountered in the main `crqa` function but has an additional two that are used to control the size of the block, e.g., `blockSize = 100`, and the argument `typeRQA`, which can take two options, either `full` or `diagonal`. If the value for `typeRQA` is `diagonal`, only the diagonal cross-recurrence will be computed; if `full`, the recurrence measures will be obtained out of the full plot[6]. In Figure 6, we visualize the computational speed (left panel) and memory demand (right panel) on simulated time series of sinusoids of increasing size and compare what happens if we run the `piecewiseRQA`, also with blocks of increasing size, as compared to running the main `crqa` function. We can clearly see that for time series of increasing length, memory demands are kept lower by the `piecewiseRQA` function as compared to the `crqa`. However, we can also see that there is a wide variance for blocks of different sizes. Therefore, it may be wise to explore the block sizes to find the one that can optimize the computational performance over a single trial before running the piecewise recurrence analysis on an entire dataset.

Estimating starting parameters: `optimizeParam`

The last function we showcase is `optimizeParam`, which helps the user exploring the space of values for the parameters of delay, embedding dimension and radius to compute recurrence quantification on continuous-valued time series. In particular, `optimizeParam` first estimates the average mutual information (AMI) of either the unidimensional or multidimensional time series and chooses the value that minimizes it. Then, it takes such a delay value and evaluates the embedding dimensions maximizing the false-nearest neighbors (FNN)[7]. As a last step, `optimizeParam` applies the values of delay and embedding dimension obtained to find a radius which returns a recurrence rate within a minimum and maximum value established by the user. We apply `optimizeParam()` to simulated

[5] This function is similar to the `crp_big` function in the long-standing CRP-toolbox for MATLAB by Marwan and colleagues: http://tocsy.pik-potsdam.de.

[6] Currently, the windowed cross-recurrence is not implemented to work with the `piecewiseRQA`.

[7] Users can also access the functions to compute delay (`MdDelay`) and embedding dimensions (`MdFnn`) independently of `optimizeParam`.

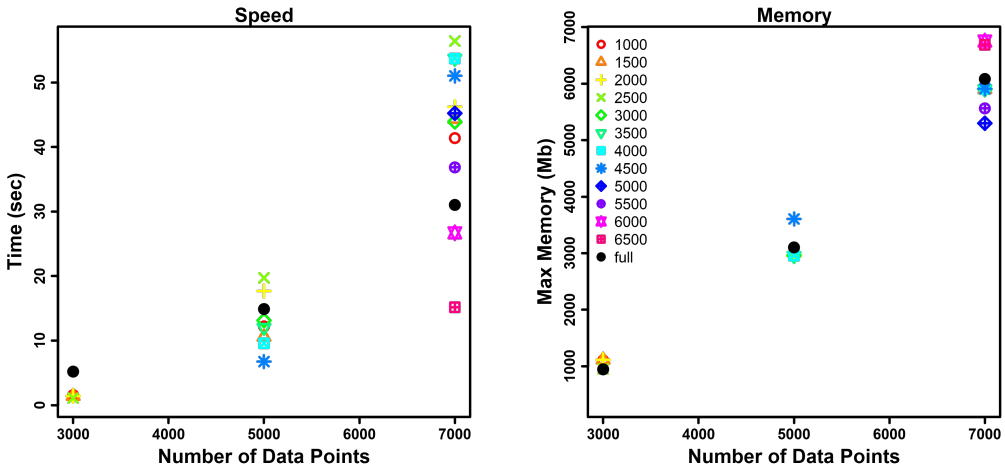


Figure 6: Evaluating the speed (time in seconds, left panel) and memory (peak RAM in MB, right panel) performance of crqa() and piecewiseRQA() for simulated data of increasing size (from 3000 to 7000 data points). We compare the use of blocks of different sizes (from 1000 to 6500 in increments of 500, coded using color and point type) with the case of running crqa() on the entire sequence of data points.

sinusoids to show the unidimensional case, and to the hand movement data of Wallot et al. (2016a) to show the multidimensional case.

In order to set up the estimation of the delay and embedding dimensions for unidimensional time series, the user has to decide how to choose the value of average mutual information (i.e., `typeami = mindip`, the lag at which minimal information is observed, or `typeami = maxlag`, the maximum lag at which minimal information is observed) and the relative percentage of information gained in FNN, relative to the first embedding dimension, when higher embeddings are considered (i.e., `fnnpercent`). Then, as `crqa` is integrated into the `optimizeParam` to estimate the radius, most of the arguments are the same (e.g., `mindiagline` or `tw`), except the number of values of that are considered (i.e., `radiusspan = 100`).

```
ts1 <- seq(0.1, 200, .1)
ts1 <- sin(ts1) + linspace(0, 1,length(ts1))
ts2 <- ts1
par <- list(method = "rqa", metric = "euclidean", maxlag =  20, radiusspan = 100,
            normalize = 0, rescale = 4,  mindiagline = 10, minvertline = 10, tw = 0,
            whiteline = FALSE, recpt = FALSE, side = "both", datatype = "continuous",
            fnnpercent  = 10,  typeami = "mindip")
results <- optimizeParam(ts1, ts2, par, min.rec = 2, max.rec = 5)
print(unlist(results))

radius emddim  delay
0.17   2       18
```

For multidimensional series, the user needs to specify the right RQA method (i.e., `method = "mdcrqa"`). Then, for the estimation of the delay via AMI: (1) `nbins`, which is the number of breaks used to define the bins within which the two-dimensional histogram (or frequency distribution) of the original and delayed time series are computed, and (2) the `criterion` to select the delay (`firstBelow` to use the lowest delay at which the AMI function drops below the value set by the `threshold` argument, and `localMin` to use the position of the first local AMI minimum). The estimation of the embedding dimensions instead needs the following arguments: (1) `maxEmb`, which is the maximum number of embedding dimensions considered, (2) `noSamples`, which is the number of randomly drawn coordinates from phase space used to estimate the percentage of false-nearest neighbors, (3) `Rtol`, which is the first distance criterion for separating false neighbors, and (4) `Atol`, which is the second distance criterion for separating false neighbors. The radius is estimated as before.

```
par$method <- "mdcrqa"
par$nbins <- 50
par$criterion <- "firstBelow"
par$threshold <- 1.6
par$maxEmb <- 20
```

```
par$numSamples <- 500
par$Rtol <- 10
par$Atol <- 2
results <- optimizeParam(P1, P2, par, min.rec = 2, max.rec = 5)
print(unlist(results))

radius  emddim  delay
0.032   11      2
```

Conclusion

This paper describes recurrence quantification analysis, a statistical method to characterize the nonlinear dynamics of a system. It has received a growing interest from researchers across very different fields from physiology to psychology because of its flexibility, ease of application, and explanatory power. In particular, we explain recurrence analysis from the simplest case of auto-recurrence of a unidimensional time series to the most complex case of multidimensional cross-recurrence. More importantly, we presented a significantly updated version of the **crqa** to perform all different variants of recurrence analysis described in the theoretical section of this manuscript. We showcased the different functions available in **crqa** with real data of categorical and continuous nature, and illustrated how starting parameters for continuous data could be obtained (i.e., radius, embedding dimension, and delay) as well as handling long time series in a memory-efficient way using additional functions available in **crqa**.

It is useful to end with some observations regarding the broader relevance of the package presented here. The RQA methodology and the updated `crqa` tap into a number of evolving problems in data analysis across various disciplines. For example, there is a drive to improve measures and models of multimodal data (Abawajy, 2015; Dale, 2015; Lahat et al., 2015), including multi-person measures (Cooke et al., 2013; López Pérez et al., 2017; Schilbach et al., 2013; von Zimmermann and Richardson, 2016; Wallot et al., 2016a). RQA and its multidimensional counterpart implemented in our package constitute an extraordinarily expansive analysis tool for exploring varied kinds of complex and multidimensional data. In addition, a demand for more dynamic quantitative analyses has now also penetrated into the social sciences (Chemero, 2011; Friedenberg, 2009; Spivey, 2008; Ward, 2002; Pagnotta et al., 2020). `crqa` is designed to be a comprehensive analysis package for studying the dynamics of diverse systems, especially systems that exhibit high degrees of interdependence, and that show signatures in their dynamics that are critical for understanding them.

Acknowledgments

SW acknowledges support from the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation)–WA 3538/4-1. MIC acknowledges support from the Fundaçâo para a Ciência e Tecnologia under grant agreement PTDC/PSI-ESP/30958/2017.

Bibliography

J. Abawajy. Comprehensive analysis of big data variety landscape. *International journal of parallel, emergent and distributed systems*, 30(1):5–14, 2015. [p159]

D. H. Abney, A. S. Warlaumont, A. Haussman, J. M. Ross, and S. Wallot. Using nonlinear methods to quantify changes in infant limb movements and vocalizations. *Frontiers in psychology*, 5:771, 2014. [p145]

P. Alex, S. Arumugam, K. Jayaprakash, and K. Suraj. Order–chaos–order–chaos transition and evolution of multiple anodic double layers in glow discharge plasma. *Results in Physics*, 5:235–240, 2015. [p145]

B. Ambrożkiewicz, Y. Guo, G. Litak, and P. Wolszczak. Dynamical response of a planetary gear system with faults using recurrence statistics. In *Topics in Nonlinear Mechanics and Physics*, pages 177–185. Springer, 2019. [p145]

S. M. Boker, J. L. Rotondo, M. Xu, and K. King. Windowed cross-correlation and peak picking for the analysis of variability in the association between behavioral time series. *Psychological methods*, 7(3): 338, 2002. [p151]

A. Chemero. *Radical embodied cognitive science*. MIT press, 2011. [p159]

M. I. Coco and R. Dale. Cross-recurrence quantification analysis of categorical and continuous time series: an r package. *Frontiers in psychology*, 5:510, 2014. URL https://doi.org/10.3389/fpsyg.2014.00510. [p145]

M. I. Coco, L. Badino, P. Cipresso, A. Chirico, E. Ferrari, G. Riva, A. Gaggioli, and A. D'Ausilio. Multilevel behavioral synchronization in a joint tower-building task. *IEEE Transactions on Cognitive and Developmental Systems*, 9(3):223–233, 2016. [p145]

M. I. Coco, R. Dale, and F. Keller. Performance in a collaborative search task: The role of feedback and alignment. *Topics in cognitive science*, 10(1):55–79, 2018. [p145]

N. J. Cooke, J. C. Gorman, C. W. Myers, and J. L. Duran. Interactive team cognition. *Cognitive science*, 37(2):255–285, 2013. [p159]

R. Dale. An integrative research strategy for exploring synergies in natural language performance. *Ecological Psychology*, 27(3):190–201, 2015. [p159]

R. Dale, A. S. Warlaumont, and D. C. Richardson. Nominal cross recurrence as a generalized lag sequential analysis for behavioral streams. *International Journal of Bifurcation and Chaos*, 21(04): 1153–1161, 2011. URL https://doi.org/10.1142/S0218127411028970. [p145, 152]

R. Dale, N. D. Duran, and M. Coco. Dynamic natural language processing with recurrence quantification analysis. *arXiv preprint arXiv:1803.07136*, 2018. [p154]

R. Donner and M. Thiel. Scale-resolved phase coherence analysis of hemispheric sunspot activity: a new look at the north-south asymmetry. *Astronomy & Astrophysics*, 475(3):L33–L36, 2007. [p145]

A. M. Fraser and H. L. Swinney. Independent coordinates for strange attractors from mutual information. *Phys. Rev. A*, 33:1134–1140, Feb 1986. URL https://doi.org/10.1103/PhysRevA.33.1134. [p149]

J. Friedenberg. *Dynamical psychology: Complexity, self-organization and mind*. ISCE Publishing, 2009. [p159]

J. Garland, E. Bradley, and J. D. Meiss. Exploring the topology of dynamical reconstructions. *Physica D: Nonlinear Phenomena*, 334:49 – 59, 2016. URL https://doi.org/10.1016/j.physd.2016.03.006. Topology in Dynamics, Differential Equations, and Data. [p149]

V. Hilarov. Detection of the fracture zone by the method of recurrence plot. *Physics of the Solid State*, 59 (12):2401–2406, 2017. [p145]

M. B. Kennel, R. Brown, and H. D. I. Abarbanel. Determining embedding dimension for phase-space reconstruction using a geometrical construction. *Phys. Rev. A*, 45:3403–3411, Mar 1992. URL https://doi.org/10.1103/PhysRevA.45.3403. [p149]

D. Lahat, T. Adali, and C. Jutten. Multimodal data fusion: an overview of methods, challenges, and prospects. *Proceedings of the IEEE*, 103(9):1449–1477, 2015. [p159]

J. Langbein, G. Nürnberg, and G. Manteuffel. Visual discrimination learning in dwarf goats and associated changes in heart rate and heart rate variability. *Physiology & behavior*, 82(4):601–609, 2004. [p145]

G. Leonardi. A method for the computation of entropy in the recurrence quantification analysis of categorical time series. *Physica A: Statistical Mechanics and its Applications*, 512:824 – 836, 2018. ISSN 0378-4371. URL https://doi.org/10.1016/j.physa.2018.08.058. [p149]

D. López Pérez, G. Leonardi, A. Niedźwiecka, A. Radkowska, J. Rączaszek-Leonardi, and P. Tomalski. Combining recurrence analysis and automatic movement extraction from video recordings to study behavioral coupling in face-to-face parent-child interactions. *Frontiers in psychology*, 8:2228, 2017. [p159]

E. N. Lorenz. Deterministic Nonperiodic Flow. *Journal of the Atmospheric Sciences*, 20(2):130–141, 1963. [p151]

N. Marwan and J. Kurths. Nonlinear analysis of bivariate data with cross recurrence plots. *Physics Letters A*, 302(5-6):299–307, 2002. [p145, 146]

N. Marwan, N. Wessel, U. Meyerfeldt, A. Schirdewan, and J. Kurths. Recurrence-plot-based Measures of Complexity and their Application to Heart-rate-variability Data. *Physical Review E*, 66(2):026702, aug 2002. doi: 10.1103/PhysRevE.66.026702. [p145]

N. Marwan, M. C. Romano, M. Thiel, and J. Kurths. Recurrence plots for the analysis of complex systems. *Physics Reports*, 438(5):237 – 329, 2007. ISSN 0370-1573. URL https://doi.org/10.1016/j.physrep.2006.11.001. [p145, 146, 149, 152]

D. Mestivier, H. Dabiré, and N. P. Chau. Effects of autonomic blockers on linear and nonlinear indexes of blood pressure and heart rate in shr. *American Journal of Physiology-Heart and Circulatory Physiology*, 281(3):H1113–H1121, 2001. [p145]

D. Mønster, D. D. Håkonsson, J. K. Eskildsen, and S. Wallot. Physiological evidence of interpersonal dynamics in a cooperative production task. *Physiology & Behavior*, 156:24 – 34, 2016. URL https://doi.org/10.1016/j.physbeh.2016.01.004. [p145]

N. H. Packard, J. P. Crutchfield, J. D. Farmer, and R. S. Shaw. Geometry from a time series. *Physical review letters*, 45(9):712, 1980. URL https://doi.org/10.1103/PhysRevLett.45.712. [p146]

M. Pagnotta, K. N. Laland, and M. I. Coco. Attentional coordination in demonstrator-observer dyads facilitates learning and predicts performance in a novel manual task. *Cognition*, 201:104314, 2020. [p145, 159]

D. C. Richardson and R. Dale. Looking to understand: The coupling between speakers' and listeners' eye movements and its relationship to discourse comprehension. *Cognitive science*, 29(6):1045–1060, 2005. [p154]

L. Schilbach, B. Timmermans, V. Reddy, A. Costall, G. Bente, T. Schlicht, and K. Vogeley. Toward a second-person neuroscience 1. *Behavioral and brain sciences*, 36(4):393–414, 2013. [p159]

K. Shockley, M. Butwill, J. P. Zbilut, and C. L. J. Webber. Cross recurrence quantification of coupled oscillators. *Physics Letters A*, 305:59–69, 2002. [p150]

K. Shockley, M.-V. Santana, and C. A. Fowler. Mutual interpersonal postural constraints are involved in cooperative conversation. *Journal of Experimental Psychology: Human Perception and Performance*, 29(2):326, 2003. [p145]

M. Spivey. *The continuity of mind*. Oxford University Press, 2008. [p159]

F. Takens. Detecting strange attractors in turbulence. In D. Rand and L.-S. Young, editors, *Dynamical Systems and Turbulence, Warwick 1980*, pages 366–381, Berlin, Heidelberg, 1981. Springer Berlin Heidelberg. URL https://doi.org/10.1007/BFb0091924. [p146]

N. Thomasson, T. J. Hoeppner, C. L. Webber Jr, and J. P. Zbilut. Recurrence quantification in epileptic eegs. *Physics Letters A*, 279(1-2):94–101, 2001. [p145]

L. T. Timothy, B. M. Krishna, and U. Nair. Classification of mild cognitive impairment eeg using combined recurrence and cross recurrence quantification analysis. *International Journal of Psychophysiology*, 120:86–95, 2017. [p145]

L. Trulla, A. Giuliani, J. Zbilut, and C. Webber Jr. Recurrence quantification analysis of the logistic equation with transients. *Physics Letters A*, 223(4):255–260, 1996. [p146]

J. von Zimmermann and D. C. Richardson. Verbal synchrony and action dynamics in large groups. *Frontiers in Psychology*, 7:2034, 2016. [p159]

S. Wallot. Recurrence quantification analysis of processes and products of discourse: A tutorial in r. *Discourse Processes*, 54(5-6):382–405, 2017. URL https://doi.org/10.1080/0163853X.2017.1297921. [p151]

S. Wallot. Multidimensional cross-recurrence quantification analysis (mdcrqa) – a method for quantifying correlation between multivariate time-series. *Multivariate Behavioral Research*, 54(2):173–191, 2019. URL https://doi.org/10.1080/00273171.2018.1512846. [p151]

S. Wallot and G. Leonardi. Analyzing multivariate dynamics using cross-recurrence quantification analysis (crqa), diagonal-cross-recurrence profiles (dcrp), and multidimensional recurrence quantification analysis (mdrqa) – a tutorial in r. *Frontiers in Psychology*, 9:2232, 2018a. ISSN 1664-1078. URL https://doi.org/10.3389/fpsyg.2018.02232. [p150, 151]

S. Wallot and G. Leonardi. Deriving inferential statistics from recurrence plots: A recurrence-based test of differences between sample distributions and its comparison to the two-sample kolmogorov-smirnov test. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 28(8):085712, 2018b. URL https://doi.org/10.1063/1.5024915. [p146]

S. Wallot and D. Mønster. Calculation of average mutual information (ami) and false-nearest neighbors (fnn) for the estimation of embedding parameters of multidimensional time series in matlab. *Frontiers in psychology*, 9, 2018. URL https://doi.org/10.3389/fpsyg.2018.01679. [p145, 149, 150, 151]

S. Wallot, P. Mitkidis, J. J. McGraw, and A. Roepstorff. Beyond synchrony: joint action in a complex production task reveals beneficial effects of decreased interpersonal synchrony. *PloS one*, 11(12): e0168306, 2016a. [p154, 157, 158, 159]

S. Wallot, A. Roepstorff, and D. Mønster. Multidimensional recurrence quantification analysis (mdrqa) for the analysis of multidimensional time-series: A software implementation in matlab and its application to group-level data in joint action. *Frontiers in psychology*, 7:1835, 2016b. URL https://doi.org/10.3389/fpsyg.2016.01835. [p145, 150, 151]

S. Wallot, J. T. Lee, and D. G. Kelty-Stephen. Switching between reading tasks leads to phase-transitions in reading times in l1 and l2 readers. *PloS one*, 14(2):e0211502, 2019. [p145]

L. M. Ward. *Dynamical cognitive science*. MIT press, 2002. [p159]

C. L. Webber and J. P. Zbilut. Dynamical assessment of physiological systems and states using recurrence plot strategies. *Journal of Applied Physiology*, 76(2):965–973, 1994. URL https://doi.org/10.1152/jappl.1994.76.2.965. [p145]

C. Webber Jr and J. Zbilut. Recurrence quantification analysis of nonlinear dynamical systems. national science foundation, washington dc, usa. chapter 2, methods for the behavioral sciences, eds. m. riley and g. van orden, 2005. URL https://nsf.gov/pubs/2005/nsf05057/nmbs/nmbs.jsp. [p149]

M. Wijnants, F. Hasselman, R. Cox, A. Bosman, and G. Van Orden. An interaction-dominant perspective on reading fluency and dyslexia. *Annals of dyslexia*, 62(2):100–119, 2012. [p145]

J. P. Zbilut and C. L. Webber. Embeddings and delays as derived from quantification of recurrence plots. *Physics Letters A*, 171(3):199 – 203, 1992. URL https://doi.org/10.1016/0375-9601(92)90426-M. [p145, 149]

J. P. Zbilut, A. Giuliani, and C. L. Webber. Detecting deterministic signals in exceptionally noisy environments using cross-recurrence quantification. *Physics Letters A*, 246(1):122 – 128, 1998. ISSN 0375-9601. URL https://doi.org/10.1016/S0375-9601(98)00457-5/. [p145, 149]

N. Zolotova and D. Ponyavin. Phase asynchrony of the north-south sunspot activity. *Astronomy & Astrophysics*, 449(1):L1–L4, 2006. [p145]

Moreno I. Coco
School of Psychology
University of East London
E154LZ, London, UK
E-mail: M.Coco@uel.ac.uk

Dan Mønster
School of Business and Social Sciences
Aarhus University
DK-8210, Aarhus V, Denmark
ORCiD: 0000-0001-9639-8823
E-mail: danm@econ.au.dk

Giuseppe Leonardi
Institute of Psychology
University of Economics and Human Sciences in Warsaw
01-043, Warsaw, Poland
E-mail: g.leonardi@vizja.pl

Rick Dale
Department of Communication
University of California, Los Angeles
CA 90005, Los Angeles, USA
E-mail: rdale@ucla.edu

Sebastian Wallot
Department of Language and Literature
Max Planck Institute for Empirical Aesthetics
GR-60322, Frankfurt a.M., Germany
E-mail: sebastian.wallot@ae.mpg.de

clustcurv: An R Package for Determining Groups in Multiple Curves

by Nora M. Villanueva, Marta Sestelo, Luis Meira-Machado and Javier Roca-Pardiñas

Abstract In many situations, it could be interesting to ascertain whether groups of curves can be performed, especially when confronted with a considerable number of curves. This paper introduces an R package, known as **clustcurv**, for determining clusters of curves with an automatic selection of their number. The package can be used for determining groups in multiple survival curves as well as for multiple regression curves. Moreover, it can be used with large numbers of curves. An illustration of the use of **clustcurv** is provided, using both real data examples and artificial data.

Keywords: multiple curves, number of groups, nonparametric, survival analysis, regression models, cluster

Introduction

A problem often encountered in many fields is the comparison of several populations through specific curves. Typical examples, considered by a number of authors, are given by the comparison of survival curves in survival analysis, children growth curves in pediatrics, or the comparison of regression curves in regression analysis. In many of these studies, it is very common to compare a large number of curves between groups, and methods of summarizing and extracting relevant information are necessary. A common approach is to look for a partition of the sample into a number of groups in such a way that curves in the same group are as alike as possible but as distinct as possible from those in other groups. This process is also known as curve clustering. A hypothesis test can be used to ascertain that the curves in the same group are equal. A fundamental and difficult problem in clustering curves is the estimation of the number of clusters in a dataset.

Traditionally, the comparison of these functions is performed using parametric models through the comparison of the resulting model parameters. This approach, however, requires the specification of the parametric model, which is often difficult and may be considered a disadvantage. Several nonparametric methods have been proposed in the literature to compare multiple curves. In the area of survival analysis, for example, several nonparametric methods have been proposed to test for the equality of survival curves for censored data. The most well-known and widely used to test the null hypothesis of no difference in survival between two or more independent groups was proposed by Mantel (1966). An alternative test that is often used is the Peto & Peto (Peto and Peto, 1972) modification of the Gehan-Wilcoxon test (Gehan, 1965). Several other variations of the log-rank test statistic using weights on each event time have been proposed in the literature (Tarone and Ware, 1977; Harrington and Fleming, 1982; Fleming et al., 1987) as well as other procedures to compare these survival curves based on different measures, as can be the medians (Chen and Zhang, 2016). There exists an extensive literature on curve comparison in the framework of regression analysis. In this context, several nonparametric tests have been proposed to test the equality of the mean functions, $H_0 : m_1 = \cdots = m_j$. Hall and Hart (1990) proposed a bootstrap test, while Härdle and Mammen (1993) Hardle and Marron (1990) suggested a semiparametric approach based on kernel smoothing. Other relevant papers on this topic are King et al. (1991) , Delgado (1993) , Kulasekera (1995) , Young and Bowman (1995), Dette and Neumeyer (2001), Pardo-Fernández et al. (2007) , Srihera and Stute (2010) , among others. A good review on this topic can be seen in the paper by Neumeyer and Dette (2003).

When the null hypothesis of equality of curves is rejected, leading to the clear conclusion that at least one curve is different, it can be interesting to ascertain whether curves can be grouped or if all these curves are different from each other. In this setting, one naïve approach would be to perform pairwise comparisons. In this line are the papers by Rosenblatt (1975), González-Manteiga and Cao (1993), Härdle and Mammen (1993), Dette and Neumeyer (2001) who proposed alternative tests of the null hypothesis of equality of curves obtained from pairwise comparisons of the estimators of the regression functions. A similar statistic was also considered by King et al. (1991). Pairwise comparisons between group levels with corrections for multiple testing are also possible in the framework of survival analysis. Among others, this can be achieved with the `pairwise_survdiff` of the package **survminer** (Kassambara et al., 2019). However, in any case, this approach would lead to a large number of comparisons without the possibility of determining groups with similar curves. Results for such a test can tell us that all combinations are different, or just one pair. When the number of curves to be compared increase, so does the difficulty of interpretation.

For partitioning a given set of curves into a set of K groups of curves (i.e., K clusters), Villanueva et al. (2019) propose an adaptation of the K-means methodology. K-means is probably the most

commonly used clustering method for splitting a dataset into a set of K groups with a very simple and fast algorithm. Furthermore, it can efficiently deal with very large data sets. One potential disadvantage of K-means clustering is that it requires the number of clusters to be pre-specified. A method is proposed by Villanueva et al. (2019) to determine the number of clusters.

The development of **clustcurv** R package has been motivated by recent contributions that account for these problems, in particular, the methods proposed by Villanueva et al. (2019) to determine groups in multiple survival curves and those introduced by Villanueva et al. (Manuscript submitted for publication, 2019) in the framework of regression curves. The **clustcurv** R package attempts to answer the following two questions: (i) given a potential large sample of curves, what is the best value for the number of clusters? (ii) What is the best subdivision of the sample curves into a given number of K clusters? To facilitate the task of selecting the optimal number of clusters as well as the composition of the clusters, it is essential to have software for implementing the proposed methods in an environment that researchers will find user-friendly and easily understandable. We believe that our package can answer this aim by providing several user-friendly functions. The package **clustcurv** is freely available from the Comprehensive R Archive Network (CRAN) at `https://cran.r-project.org/web/packages/clustcurv`

Three data sets were chosen for illustration of the software usage with real data. The first two datasets show the applicability of the proposed methods for obtaining clusters of survival curves. These applications were chosen to solve two real problems in the study of recurrence of breast cancer patients and survival of myeloma cancer. To illustrate the package usage in the regression context we used real data from a Barnacle's Growth study conducted in Galicia, Spain. Simulated data were also used to illustrate the package capabilities in a more complicated scenario.

The remainder of the paper is structured as follows: Section 2 briefly reviews methods for selecting the number of clusters and the nonparametric test used; Section 3 explains the use of the main functions and methods of **clustcurv**; Section 4 gives an illustration of the practical application of the package using real and simulated data; and finally, the last section concludes with a discussion and possible future extensions of the package.

An overview of the methodology

In this section, we briefly review the methodological background of the **clustcurv** package. As it solves problems addressed in the field of survival analysis and regression analysis, firstly, the notation and the nonparametric estimation procedures for both contexts are exposed. Then, the procedure for determining groups of curves is explained in detail, considering a general framework which includes the aforementioned contexts. Briefly, our procedure is described as follows. First, the J curves are estimated by nonparametric estimators. Second, given a number of K groups, the optimal possible assignment of J curves into K groups is chosen by means of a heuristic algorithm. Third, the optimal number of groups is determined using an automatic bootstrap-based testing procedure.

Notation and estimation procedure in the survival context

We will assume the J-sample general random censorship model where observations are made on n_j individuals from population $j (j = 1, \ldots, J)$. Denote $n = \sum_{j=1}^{J} n_j$ and suppose that the observations from the n individuals are mutually independent. Let T_{ij} be an event time corresponding to an event measured from the start of the follow-up of the i-th subject $(i = 1, \ldots, n_j)$ in the sample j, and assume that T_{ij} is observed subject to a (univariate) random right-censoring variable C_{ij} assumed to be independent of T_{ij}. Due to the censoring, rather than T_{ij}, we observe $(\widetilde{T}_{ij}, \Delta_{ij})$, where $\widetilde{T}_{ij} = \min(T_{ij}, C_{ij})$, $\Delta_{ij} = I(T_{ij} \leq C_{ij})$, where $I(\cdot)$ is the indicator function.

Since the censoring time is assumed to be independent of the process, the survival functions, $S_j(t) = P(T_j > t)$, may be consistently estimated by the Kaplan-Meier estimator (Kaplan and Meier, 1958) based on the $(\widetilde{T}_j, \Delta_j)$. The Kaplan-Meier estimator or the Product-Limit estimator is a nonparametric method frequently used to estimate survival for censored data. Let $t_1 < t_2 < \ldots < t_{m_j}$, $m_j \leq n_j$ denote the distinct ordered failure times from population j $(j = 1, \ldots, J)$, and let d_u be the number of events from population j at time t_u. Then, the Kaplan-Meier estimator of survival (for population j) is

$$\widehat{S}_j(t) = \prod_{u:t_u \leq t} \left(1 - \frac{d_u}{R_j(t_u)}\right),$$

where $R_j(t) = \sum_{i=1}^{n_j} I(\widetilde{T}_{ij} \geq t)$ denote the number of individuals at risk just before time t, among individuals from population j. The Kaplan-Meier estimate is a step function with jumps at event times.

The size of the steps depends on the number of events and the number of individuals at risk at the corresponding time. Under this setup, we will be interested to determine clusters in multiple survival curves.

Notation and estimation procedure in the regression context

Let (X_j, Y_j) be J independent random vectors, and assume that they satisfy the following nonparametric regression models, for $j = 1, \ldots, J$,

$$Y_j = m_j(X_j) + \varepsilon_j, \tag{1}$$

where the error variable ε_j has mean zero, and $m_j(X_j) = E(Y_j|X_j)$ is the unknown regression function. We do not make any assumptions about the error distribution.

The regression models in (1) can be estimated using several approaches, such as methods based on regression splines (de Boor, 2001), Bayesian approaches (Lang and Brezger, 2004), or local polynomial kernel smoothers (Wand and Jones, 1995; Fan and Gijbels, 1996). In this package, local linear kernel smoothers, as implemented in the **npregfast** package, are used.

Determining groups of nonparametric curves

As noted earlier, several authors have proposed different methods that can be used to compare estimates of nonparametric functions of multiple samples. The null hypothesis is that all the curves have identical functions, $H_0 : \mathcal{F}_1 = \ldots = \mathcal{F}_J$. However, if this hypothesis is rejected, there are no available procedures that let determine groups among these curves, that is, to assess if the levels $\{1, \ldots, J\}$ can be grouped in K groups $(\mathcal{G}_1, \ldots, \mathcal{G}_K)$ with $K < J$, so that $\mathcal{F}_i = \mathcal{F}_j$ for all $i, j \in \mathcal{G}_k$, for each $k = 1, \ldots, K$. Note that $(\mathcal{G}_1, \ldots, \mathcal{G}_K)$ must be a partition of $\{1, \ldots, J\}$, and therefore must satisfy the following conditions:

$$\mathcal{G}_1 \cup \ldots \cup \mathcal{G}_K = \{1, \ldots, J\} \quad \text{and} \quad \mathcal{G}_\mathrm{i} \cap \mathcal{G}_\mathrm{j} = \emptyset, \quad \forall \mathrm{i} \neq \mathrm{j} \in \{1, \ldots, \mathrm{K}\}. \tag{2}$$

We propose a procedure to test, for a given number K, the null hypothesis $H_0(K)$ that at least one partition exists $(\mathcal{G}_1, \ldots, \mathcal{G}_K)$ so that all the conditions above are verified. The alternative hypothesis $H_1(K)$ is that for any $(\mathcal{G}_1, \ldots, \mathcal{G}_K)$, exists at least a group $\mathcal{G}_k$ in which $\mathcal{F}_i \neq \mathcal{G}_j$ for some $i, j \in \mathcal{G}_k$.

The cited testing procedure is based on the J-dimensional process

$$\widehat{\mathbf{U}}(z) = (\widehat{U}_1(z), \widehat{U}_2(z), \ldots, \widehat{U}_J(z))^t,$$

where, for $j = 1, \ldots, J$,

$$\widehat{U}_j(z) = \sum_{k=1}^{K} [\widehat{\mathcal{F}}_j(z) - \widehat{\mathcal{C}}_k(z)] \; I_{\{j \in \mathcal{G}_k\}},$$

and $\widehat{\mathcal{C}}_k$ is the pooled nonparametric estimate based on the combined $\mathcal{G}_k$-partition sample.

The following test statistics were considered in order to test $H_0(K)$: a Cramér-von Mises type statistic

$$D_{CM} = \min_{\mathcal{G}_1, \ldots, \mathcal{G}_K} \sum_{j=1}^{J} \int_R \hat{U}_j^2(z) dz,$$

and a modification of it based on the L_1 norm proposed in the Kolmogorov-Smirnov test statistic

$$D_{KS} = \min_{\mathcal{G}_1, \ldots, \mathcal{G}_K} \sum_{j=1}^{J} \int_R |\hat{U}_j(z)| dz,$$

where R is the support of the lifetime distribution or the support of the independent variable in case of survival or regression, respectively.

In order to approximate the minimizers involved in the test statistics, we propose the use of clustering algorithms. Particularly, in the case of D_{CM}, defined in terms of the L_2-distance, we propose the use of the K-means (Macqueen, 1967). However, for obtaining the values of D_{KS}, defined in this case in terms of the L_1-norm, a variation of the K-means where instead of calculating the mean for each group to determine its centroid, it calculates the median, the k-medians —suggested by Macqueen (1967) and developed by Kaufman and Rousseeuw (1990)— would be more appropriate. In both cases, the carried-out procedure is equivalent: the functions $\mathcal{F}_j$ $(j = 1, \ldots, J)$ have to be estimated

in a common grid of size Q leading to a matrix of (J x Q) dimension, where each row corresponds with the estimates of the j curve in the Q positions of the grid. Then, this matrix will be the input of both heuristic methods, K-means and K-medians, and from these, the "best" partition $(\mathcal{G}_1, \ldots, \mathcal{G}_K)$ is obtained.

Finally, the decision rule based on D consists of rejecting the null hypothesis if D is larger than the $(1-\alpha)$-percentile obtained under the null hypothesis. To approximate the distributions of the test statistic under the null hypothesis, resampling methods such as the bootstrap introduced by Efron (1979) can be applied.

The testing procedure used here involves the following steps:

1. Using the original sample, for $j = 1, \ldots, J$ and $i = 1, \ldots, n_j$, estimate the functions $\mathcal{F}_j$ in a nonparametric way and in a common grid using each sample separately. Then, using the proposed algorithms, obtain the "best" partition $(\mathcal{G}_1, \ldots, \mathcal{G}_K)$. With it, obtain the estimated curves $\widehat{\mathcal{C}}_k$ using a pooled nonparametric estimator based on the combined partition samples (i.e., the estimator obtained by applying the nonparametric estimator to the combined partition samples).
2. Obtain the D value as explained before.
3. Draw bootstrap samples using a bootstrap procedure. In the survival context, follow step 3.(a), and in the regression context, follow step 3.(b):
   (a) For $b = 1, \ldots, B$ (e.g., $B = 1000$), and for each $j \in \mathcal{G}_k$, draw $(\widetilde{T}_{1j}^{*b}, \Delta_{1j}^{*b}), (\widetilde{T}_{2j}^{*b}, \Delta_{2j}^{*b}), \ldots, (\widetilde{T}_{n_jj}^{*b}, \Delta_{n_jj}^{*b})$ by independent sampling n_j times with replacement from the empirical distribution function, $\widehat{F}_k$, putting mass n_k^{-1} ($n_k = \sum_{j=1}^{J} n_j I_{\{j \in \mathcal{G}_k\}}$) at each point $(\widetilde{T}_{ij}, \Delta_{ij})$, with $j \in \mathcal{G}_k$. Note that this procedure is a pooled bootstrap, i.e., bootstrap from the pooled-combined partition sample given by the null hypothesis $H_0(K)$.
   (b) For $b = 1, \ldots, B$, and for each $j \in \mathcal{G}_k$, draw $\left\{\left(X_{i1}, Y_{i1}^{*b}\right)\right\}_{i=1}^{n_1}, \ldots, \left\{\left(X_{iJ}, Y_{iJ}^{*b}\right)\right\}_{i=1}^{n_J}$, where

$$Y_{ij}^{*b} = \sum_{k=1}^{K} \widehat{\mathcal{C}}_k(X_{ij}) I_{\{j \in \mathcal{G}_k\}} + \widehat{\varepsilon}_{ij} W_i^{*b}$$

   being $\widehat{\varepsilon}_{ij}$ the null errors under the $H_0(K)$ obtained as

$$\widehat{\varepsilon}_{ij} = \sum_{k=1}^{K} \left(Y_{ij} - \widehat{\mathcal{C}}_k(X_{ij})\right) I_{\{j \in \mathcal{G}_k\}},$$

   and the variables $W_1^{*b}, \ldots, W_n^{*b}$ are independent for the observed sample and i.i.d. with $E(W_i^{*b}) = 0$, $Var(W_i^{*b}) = 1$, and third moment equals to 1. A common choice is to consider a binary variable with probabilities $P\{W_i^{*b} = (1-\sqrt{5})/2\} = (5+\sqrt{5})/10$ and $P\{W_i^{*b} = (1+\sqrt{5})/2\} = (5-\sqrt{5})/10$, which corresponds to the *golden section*. Note that we have used the wild bootstrap here (Wu, 1986; Liu, 1988; Mammen, 1993) because this method is valid both for homoscedastic and for heteroscedastic models where the variance of the error is a function of the covariate.
4. Let D^{*b} be the test statistic obtained from the bootstrap samples after applying step 1 and 2 to the cited bootstrap samples.

Since in step 3 the bootstrap resamples are constructed under the null hypothesis of K groups, this mechanism approximates the distribution of the test statistic under the null hypothesis. If we denote $D^{*(b)}$ for the order statistics of the values $D^{*1}, \ldots, D^{*B}$ obtained in step 4, then $D^{*([(1-\alpha)B])}$ approximates the $(1-\alpha)$-quantile of the distribution of D under the null hypothesis.

It is important to highlight that repeating this procedure, testing $H_0(K)$, from $K = 1$ onwards until a certain null hypothesis is not rejected allows us to determine the number of K groups automatically. Note that however, that unlike the previous test decision, this latter one is not statistically significant (strong evidences for rejecting the null hypothesis are not given). The whole procedure is briefly described step by step in Algorithm 1.

Finally, note that, under survival and regression scenarios, the proposed procedure for the determination of groups in multiple curves may be translated as a test of multiple hypotheses where a set of K p-values corresponding to the K null hypotheses, $H_0(1), H_0(2), \ldots, H_0(K)$ are given. Even though several methods have been proposed to deal with this problem (see, e.g., Dudoit and van der Laan (2008) for an introduction to this area), there are still open challenges because there is no information about the minimum number of tests needed to apply these techniques. In any case, we have decided

Algorithm 1: k-nonparametric curves algorithm

1. With the original sample, for $j = 1, \ldots, J$ and $i = 1, \ldots, n_j$, and using the nonparametric estimator obtain $\widehat{\mathcal{F}}_j$.
2. Initialize with $K = 1$ and test $H_0(K)$:
   - 2.1 Obtain the "best" partition $\mathcal{G}_1, \ldots, \mathcal{G}_K$ by means of the k-means or k-medians algorithm.
   - 2.2 For $k = 1, \ldots, K$, estimate $\widehat{\mathcal{C}}_k$ and retrieve the test statistic D.
   - 2.3 Generate B bootstrap samples and calculate D^{*b}, for $b = 1, \ldots, B$.
   - 2.4 **if** $D > D^{*(1-\alpha)}$ **then**
     reject $H_0(K)$
     $K = K + 1$
     go back to 2.1
     else
     accept $H_0(K)$
     end
3. The number K of groups of equal nonparametric curves is determined.

to propose a possible approach to apply some of these well-known techniques as Bonferroni, Holm (Holm, 1979), etc. As the problem is still open, we feel that the final user must be able to decide to apply them by means of an argument included in the functions of the package. The challenge in the present context is that the number of hypotheses that are going to be tested is unknown in advance. In order to solve this we propose that, after having increased K in the algorithm, the null hypothesis for "smaller K's" has to be re-tested simultaneously with $H_0(K)$.

Package structure and functionality

The **clustcurv** package is a shortcut for "clustering curves" for being this its major functionality: to provide a procedure that allows users to determine groups of multiple curves with an automatic selection of their number. The package enables both survival and regression curves to be grouped, and it is designed along lines similarly into both contexts. In addition, in view of the high computational cost entailed in these methods, parallelization techniques are included to become feasible and efficient in real situations.

The functions within the **clustcurv** package are described in Table 1. Briefly, there are two main types of functionalities: (i) to determine groups of multiple curves with the automatic selection of their number with `regclustcurves` or `survclustcurves` functions and (ii) to determine groups of curves, given a number K, with `kregcurves` or `ksurvcurves` functions. The `S3` object obtained from whatever previous functions is the argument required as input for `autoplot`, which returns a graphical output based on **ggplot2** package. Numerical summaries of the fitted objects can be obtained by using `print` or `summary`.

Since the two most important functions in this package are `survclustcurv` and `regclustcurv`, the arguments of these functions are shown in Table 2. Note that the `ksurvcurves` `kregcurves` functions are just a simplified version of the previous two. Users can automatically obtain the optimal number of groups of multiple curves by means of `survclustcurves` and `regclustcurves`. Nevertheless, in those situations where the user knows in advance the number of groups, it is possible to obtain the assignment of the curves into the corresponding group, by means of the function `ksurvcurves` or `kregcurves`. In both functions, a common argument is the algorithm, which returns the best assignments of the curves into the groups to which they belong. At the moment, the algorithms to solve this optimization problem can be K-means or K-medians, through the argument `algorithm = 'kmeans'` or `algorithm = 'kmedians'`.

Furthermore, in order to address the high computational burden, the functions `survclustcurves`, `regclustcurves`, `ksurvcurves` and `kregcurves` have been programmed in parallel to compute the bootstrap-based testing procedure. The input command required for the use of parallelization is `cluster = TRUE`. The number of cores for parallel execution is fixed using the number of CPU

cores on the current host minus one unless it is specified by the user (`ncores = NULL`). Then, `registerDoParallel` of the **doParallel** package is used to register the parallel backend. The parallel computation is performed by the `foreach` function of **foreach** package.

Function	Description
`survclustcurves`	Main function for determining groups of multiple survival curves and selecting automatically the optimal number of them.
`regclustcurves`	Main function for determining groups of multiple regression curves ecting automatically the optimal number of them.
`ksurvcurves`	Main function for determining groups of survival curves, given a number of groups K.
`kregcurves`	Main function for determining groups of regression curves, given a number of groups K.
`summary`	Method of the generic summary function for `kcurves` and `clustcurves` objects (both survival and regression context), which returns a short summary.
`print`	Method of the generic print function for `kcurves` and `clustcurves` objects, which prints out some key components.
`autoplot`	Visualisation of `clustcurves` and `kcurves` objects with **ggplot2** (Wickham et al., 2019) graphics. Provides the plots for the estimated non-parametric curves grouped by color (optional) and their centroids (mean curve of the curves pertaining to the same group).

Table 1: Summary of functions in the **clustcurv** package.

Illustrative examples

In this section, we illustrate the use of **clustcurv** package using some real and simulated data. In the case of the survival context, the proposed methods were applied to the German breast cancer data included in the **condSURV** package and to the multiple myeloma data freely available as part of the **survminer** package. For the regression analysis, the **clustcurv** package includes a data set called `barnacle5` with measurements of rostro-carinal length and dry weight of barnacles collected from five sites of Galicia (northwest of Spain). Additionally, in order to show the behaviour of the method in a more complicated scenario, an example with simulated data is also provided.

Application to German Breast Cancer Study Data

In this study, a total of 686 patients with primary node-positive breast cancer were recruited between July 1984 and December 1989, and 16 variables were measured such as the age of the patient (`age`), menopausal status (`menopause`), hormonal therapy (`hormone`), tumour size (`size`, in mm), tumor grade (`grade`), and the number of positive nodes (`nodes`). In addition to these and other variables, the recurrence-free survival time (`rectime`, in days) and the corresponding censoring indicator (0 – censored, 1 – event) were also recorded.

We will use these data to illustrate the package capabilities to build clusters of survival curves based on the covariate nodes (grouped from 1 to > 13). An excerpt of the `data.frame` with one row per patient is shown below:

```
> library(condSURV)
> library(clustcurv)
> data(gbcsCS)
> head(gbcsCS[, c(5:10, 13, 14)])
  age menopause hormone size grade nodes rectime censrec
1  38         1       1   18     3     5    1337       1
2  52         1       1   20     1     1    1420       1
3  47         1       1   30     2     1    1279       1
4  40         1       1   24     1     3     148       0
5  64         2       2   19     2     1    1863       0
6  49         2       2   56     1     3    1933       0
```

The first three patients have developed a recurrence shown by `censrec` variable equals to 1, unlike the following three, which take the value of 0. This variable, along with the other two, `rectime` and

	`survclustcurves()` arguments
`time`	A vector with variable of interest, i.e. survival time.
`status`	A vector with censoring indicator of the survival time of the process; 0 if the total time is censored and 1 otherwise.
`x`	A vector with categorical variable indicating the population to which the observations belongs.
`kvector`	A vector specifying the number of groups of curves to be checking. By default it is NULL.
`kbin`	Size of the grid over which the survival functions are to be estimated.
`nboot`	Number of bootstrap repeats.
`algorithm`	A character string specifying which clustering algorithm is used, i.e., *K*-means (`'kmeans'`) or *K*-medians (`'kmedians'`).
`alpha`	A numeric value, particularly, the signification level of the hypothesis test.
`cluster`	A logical value. If `TRUE` (default) the code is parallelized. Note that there are cases without enough repetitions (e.g., a low number of initial variables) that R will gain in performance through serial computation. R takes time to distribute tasks across the processors also it will need time for binding them all together later on. Therefore, if the time for distributing and gathering pieces together is greater than the time needed for single-thread computing, it could be better not to parallelize.
`ncores`	An integer value specifying the number of cores to be used in the parallelized procedure. If `NULL`, the number of cores to be used is equal to the number of cores of the machine $-$ 1.
`seed`	Seed to be used in the procedure.
`multiple`	A logical value. If TRUE (not default), the resulted pvalues are adjusted by using one of several methods for multiple comparisons.
`multiple.method`	Correction method: `'bonferroni'`, `'holm'`, `'hochberg'`, `'hommel'`, `'BH'`, `'BY'`
	`regclustcurves()` arguments
`y`	A vector with variable of interest, i.e. response variable.
`x`	A vector with independent variable.
`z`	A vector with categorical variable indicating the population to which the observations belongs.
`kvector`	A vector specifying the number of groups of curves to be checking. By default it is NULL.
`kbin`	Size of the grid over which the survival functions are to be estimated.
`h`	The kernel bandwidth smoothing parameter.
`nboot`	Number of bootstrap repeats.
`algorithm`	A character string specifying which clustering algorithm is used, i.e., *K*-means (`'kmeans'`) or *K*-medians (`'kmedians'`).
`alpha`	A numeric value, particularly, the signification level of the hypothesis test.
`cluster`	A logical value. If `TRUE` (default) the code is parallelized. Note that there are cases without enough repetitions (e.g., a low number of initial variables) that R will gain in performance through serial computation. R takes time to distribute tasks across the processors also it will need time for binding them all together later on. Therefore, if the time for distributing and gathering pieces together is greater than the time needed for single-thread computing, it could be better not to parallelize.
`ncores`	An integer value specifying the number of cores to be used in the parallelized procedure. If `NULL`, the number of cores to be used is equal to the number of cores of the machine $-$ 1.
`seed`	Seed to be used in the procedure.
`multiple`	A logical value. If TRUE (not default), the resulted pvalues are adjusted by using one of several methods for multiple comparisons.
`multiple.method`	Correction method: `'bonferroni'`, `'holm'`, `'hochberg'`, `'hommel'`, `'BH'`, `'BY'`

Table 2: Arguments of `survclustcurves` and `regclustcurves`

nodes, will be taken into account for applying the methods described in Section 2.2. The number of positive nodes has been grouped from 1 to > 13 because of its low numbers onwards. Below, the steps for this preprocessed are shown:

```
> table(gbcsCS$nodes)

  1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20  21
187 110  79  57  41  33  36  20  20  19  15  13  11   3   5   8   5   5   5   3   1
 23  24  26  30  33  35  36  38  51
  1   2   1   1   1   1   1   1   1

> gbcsCS[gbcsCS$nodes > 13,'nodes'] <- 14
> gbcsCS$nodes <- factor(gbcsCS$nodes)
> levels(gbcsCS$nodes)[14]<- '>13'
> table(gbcsCS$nodes)

  1   2   3   4   5   6   7   8   9  10  11  12  13 >13
187 110  79  57  41  33  36  20  20  19  15  13  11  45
```

Estimates of the survival curves are obtained using the `survclustcurves` function. This function allows determining groups using the optimization algorithm *K*-means or *K*-medians. The function will verify if data has been introduced correctly and will create a `'clustcurves'` object. The first three arguments must be introduced, where `time` is a vector with event-times, `status` for their corresponding indicator statuses, and `x` is the categorical covariate.

As we mentioned, note that the proposed procedure may deal with the problem of testing multiple hypotheses, particularly relevant when the categorical variable has many levels. Thus, if the user wants to apply some correction, it is possible to specify `multiple = TRUE` and select some of the well-known techniques such as Bonferroni, Holm, etc., by means of the argument `multiple.method`.

The output of this function is the assignment of the survival curves to the group to which they belong and an automatic selection of their number. The following input commands provide an example of this output using the *K*-medians algorithm:

```
> fit.gbcs <- survclustcurves(time = gbcsCS$rectime, status = gbcsCS$censrec,
                x = gbcsCS$nodes, nboot = 500, seed = 300716, algorithm = 'kmedians',
                cluster = TRUE)
Checking 1 cluster...
Checking 2 clusters...
Checking 3 clusters...

Finally, there are 3 clusters.
> summary(fit.gbcs)

Call:
survclustcurves(time = gbcsCS$rectime, status = gbcsCS$censrec,
    x = gbcsCS$nodes, nboot = 500, algorithm = "kmedians", cluster = TRUE,
    seed = 300716)

Clustering curves in 3 groups

Number of observations:  640
Cluster method:  kmedians

Factor's levels:
 [1] "1"   "2"   "3"   "4"   "5"   "6"   "7"   "8"   "9"   "10"  "11"  "12"  "13"
[14] ">13"

Clustering factor's levels:
 [1] 1 1 1 3 3 3 3 2 3 2 2 2 2 2

Testing procedure:
  H0   Tvalue pvalue
1  1 95.68626  0.000
2  2 56.03966  0.018
3  3 33.63386  0.830
```

```
Available components:
 [1] "num_groups" "table"      "levels"     "cluster"    "centers"    "curves"
 [7] "method"     "data"       "algorithm"  "call"
```

The graphical representation of the fit can be easily obtained with the function `autoplot`. Specifying the argument `groups_by_color = FALSE`, the estimated survival curves for each level of the factor nodes by means of the Kaplan-Meier estimator can be drawn. The assignment of the curves to the three groups can be observed in Figure 1 simply typing `groups_by_color = TRUE`. As expected, the survival of patients can be influenced by the number of lymph nodes. The patients' recurrence time rises with the decrease of lymph nodes. Note that having 3 or fewer positive nodes seems to be related to higher recurrence-free probabilities. Patients with 9 or more positive nodes are more likely to develop a recurrence. The group of patients with 8 positive nodes was assigned to the group with highest recurrence probabilities. Though this was unexpected, further analysis confirm the poor and unexpected behavior.

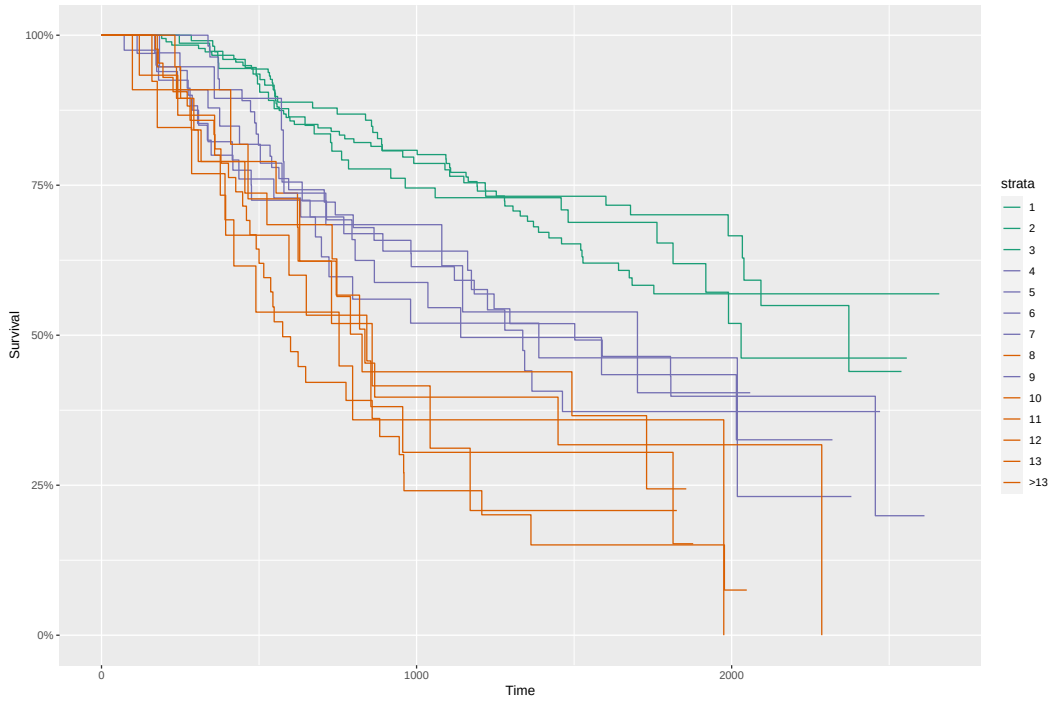


Figure 1: Estimated survival curves for each of the levels of the variable `nodes`. A specific color is assigned for each curve according to the group to which it belongs using the K-medians algorithm (in this case, three groups, $K = 3$).

Equivalently, the following piece of code shows the input commands and the results obtained with the `algorithm = 'kmeans'`. However, the number of groups and the assignments are different from those obtained with the `'kmedians'`. Although this situation is not so common, in some real applications, it can happen.

```
> fit.gbcs2 <- survclustcurves(time = gbcsCS$rectime, status = gbcsCS$censrec,
                 x = gbcsCS$nodes, nboot = 500, seed = 300716, algorithm = 'kmeans',
                 cluster = TRUE)
Checking 1 cluster...
Checking 2 clusters...

Finally, there are 2 clusters.


> fit.gbcs2

Call:
survclustcurves(time = gbcsCS$rectime, status = gbcsCS$censrec,
    x = gbcsCS$nodes, nboot = 500, algorithm = "kmeans", cluster = TRUE,
    seed = 300716)
```

```
Clustering curves in 2 groups

Number of observations:  607
Cluster method:  kmeans
```

The corresponding plot is shown in Figure 2. Note that having 9 or more positive nodes seems to be related to a lower recurrence-free survival than having 9 or less, with the exception of the survival curve for those patients with 8 positive nodes, which was assigned to the group with highest recurrence probabilities.

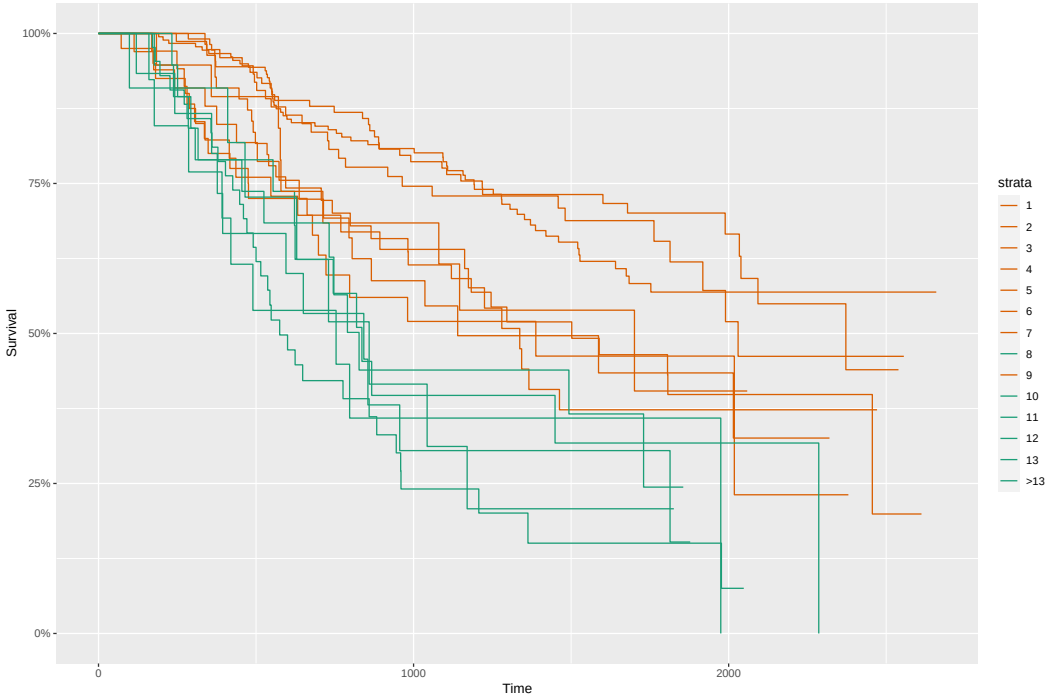


Figure 2: Estimated survival curves for each of the levels of the variable `nodes`. A specific color is assigned for each curve according to the group to which it belongs using the K-means algorithm (in this case, two groups, $K = 2$).

It is important to highlight that given a fixed value of K, one may also be interested in determining the group to which each survival function belongs. This is possible by means of the `ksurvcurves` function by considering, for example, the argument `k = 3`.

```
> ksurvcurves(time = gbcsCS$rectime, status = gbcsCS$censrec, x = gbcsCS$nodes,
                    seed = 300716, algorithm = 'kmedians', k = 3)

Call:
ksurvcurves(time = gbcsCS$rectime, status = gbcsCS$censrec, x = gbcsCS$nodes,
    k = 3, algorithm = "kmedians", seed = 300716)

Clustering curves in 3 groups

Number of observations:  640
Cluster method:  kmedians
```

More information related to the output above can be obtained running the `summary` function.

Application to Multiple Myeloma Study Data

In this case, a study of the survival in patients with multiple myeloma (MM) cancer was conducted. 256 individuals were included from the start of the follow-up to whom were analyzed and collected 16 variables. This data set is freely available in the **survminer** package. Below, it is shown the first rows of the `data.frame` with columns such as treatment (`treatment`), life state indicator (`event`; censored – 0; 1 – dead), survival time (`time`, in months), among others.

```
> library(survminer)
> data(myeloma)
> head(myeloma[,1:5])
         molecular_group chr1q21_status treatment event  time
GSM50986      Cyclin D-1       3 copies       TT2     0 69.24
GSM50988      Cyclin D-2       2 copies       TT2     0 66.43
GSM50989           MMSET       2 copies       TT2     0 66.50
GSM50990           MMSET       3 copies       TT2     1 42.67
GSM50991             MAF           <NA>       TT2     0 65.00
GSM50992    Hyperdiploid       2 copies       TT2     0 65.20
```

In this example, it is interesting to analyze if the survival in patients with MM disease is the same for the different molecular subgroups. If there is an effect of the molecular subgroups on the survival, future therapies that might exploit molecular insights should lead to an improvement in outcome for patients with these types of disease (Zhan et al., 2006).

Below, a summary of the results of the `survclustcurves` function obtained with `time`, `event`, and `molecular_group` as input variables and for both `kmedians` and `kmeans` algorithms are shown.

```
> fit.mye <- survclustcurves(time = myeloma$time, status = myeloma$event,
                      x = myeloma$molecular_group, nboot = 500, seed = 300716,
                      algorithm = 'kmedians', cluster = TRUE)
Checking 1 cluster...
Checking 2 clusters...

Finally, there are 2 clusters.
> summary(fit.mye)

Call:
survclustcurves(time = myeloma$time, status = myeloma$event,
    x = myeloma$molecular_group, nboot = 500, algorithm = "kmedians",
    cluster = TRUE, seed = 300716)

Clustering curves in 2 groups

Number of observations:  248
Cluster method:  kmedians

Factor's levels:
[1] "Cyclin D-1"       "Cyclin D-2"       "Hyperdiploid"     "Low bone disease"
[5] "MAF"              "MMSET"            "Proliferation"

Clustering factor's levels:
[1] 1 1 1 1 1 2 2

Testing procedure:
  H0   Tvalue pvalue
1  1 31.31603  0.026
2  2 14.94269  0.682

Available components:
 [1] "num_groups" "table"      "levels"     "cluster"    "centers"    "curves"
 [7] "method"     "data"       "algorithm"  "call"

> fit.mye2 <- survclustcurves(time = myeloma$time, status = myeloma$event,
                      x = myeloma$molecular_group, nboot = 500, seed = 300716,
                      algorithm = 'kmeans', cluster = TRUE)
Checking 1 cluster...
Checking 2 clusters...

Finally, there are 2 clusters.
```

```
> summary(fit.mye2)

Call:
survclustcurves(time = myeloma$time, status = myeloma$event,
    x = myeloma$molecular_group, nboot = 500, algorithm = "kmeans",
    cluster = TRUE, seed = 300716)

Clustering curves in 2 groups

Number of observations:  248
Cluster method:  kmeans

Factor's levels:
[1] "Cyclin D-1"       "Cyclin D-2"       "Hyperdiploid"     "Low bone disease"
[5] "MAF"              "MMSET"            "Proliferation"

Clustering factor's levels:
[1] 1 1 1 1 1 2 2

Testing procedure:
  H0   Tvalue pvalue
1  1 4.500272  0.032
2  2 1.108812  0.730

Available components:
 [1] "num_groups" "table"      "levels"     "cluster"    "centers"    "curves"
 [7] "method"     "data"       "algorithm"  "call"
```

When comparing the results obtained through the two methods (`kmeans`, `kmedians`), it is seen that the obtained number of clusters is the same (2 groups), even the assignment of the curves to the groups.

In particular, results obtained reveal that MMSET level and Proliferation level are associated with a high-risk or damage on the lifetime, while MAF, Low bone disease, Hyperdiploid, Cycline D-1, and Cycline D-2 have higher survival probabilities. This is observed in the plot shown in Figure 3, which can be obtained using the following input command:

```
> autoplot(fit.mye, groups_by_color = TRUE)
```

Application to Barnacle's Growth Study Data

This study was conducted on the Atlantic coast of Galicia (Northwest Spain), which consists of an approximately 1000km long shoreline with extensive rocky stretches exposed to tidal surges and wave action that are settled by the *Pollicipes pollicipes* (Gmelin, 1789) populations targeted for study. A total of 5000 specimens were collected from five sites of the region's Atlantic coastline and corresponded to the stretches of coast where this species is harvested: Punta do Mouro, Punta Lens, Punta de la Barca, Punta del Boy and Punta del Alba. Two biometric variables of each specimen were measured: RC (Rostro-carinal length, maximum distance across the capitulum between the ends of the rostral and carinal plates) and DW (Dry Weight). This data set (`barnacle5`) is available in the **clustcurv** package. The idea of this study is to know the relation between RC and DW variables along the coast, i.e., to analyze if the barnacle's growth is similar in all locations or by contrast, if it is possible to detect geographical differentiation in growth. A sample of the dataset is shown as follow:

```
> data("barnacle5")
> head(barnacle5)
    DW   RC    F
1 0.52 12.0 laxe
2 1.46 18.9 laxe
3 0.05  6.4 laxe
4 0.17  9.4 laxe
5 0.05  6.2 laxe
6 0.41 12.2 laxe
```

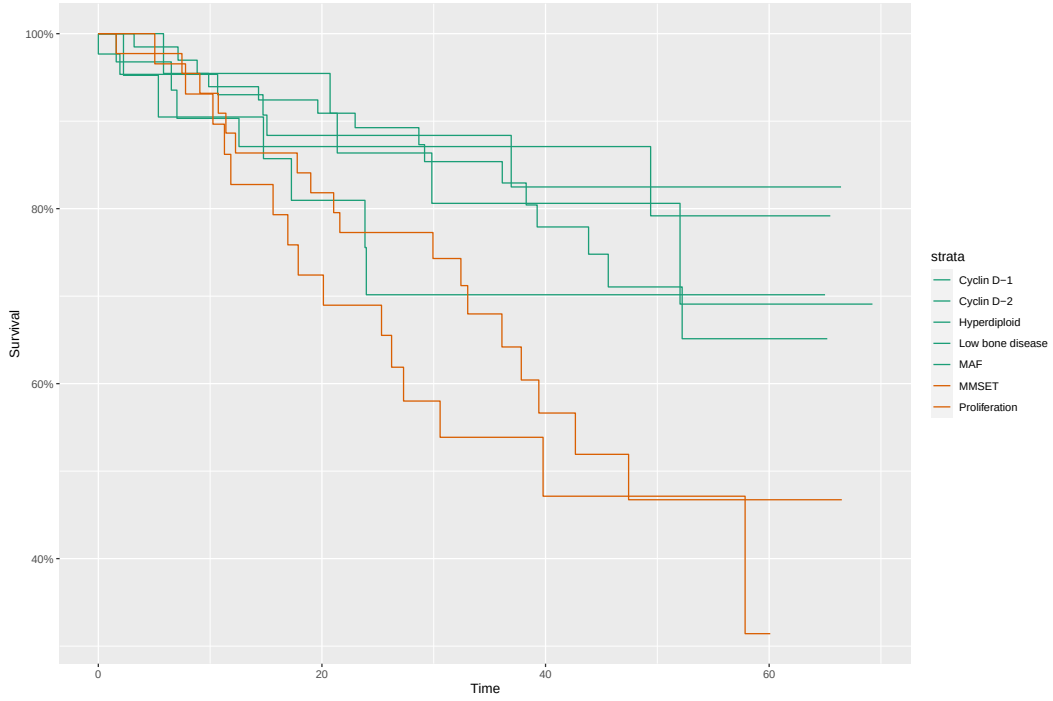


Figure 3: Estimated survival curves for each of the levels of the variable `molecular group`. A specific color is assigned for each curve according to the group to which it belongs using the K-medians algorithm, two groups, $K = 2$.

For each location (`F`), nonparametric regression curves were estimated to modeling the dependence between RC and DW. In order to determine groups, we used the proposed methodology in Subsection 2.2.2. Through executing the next piece of code, the following results can be obtained: one estimated curve was attributed to the first group (Punta Lens), two estimated curves were assigned to group 2 (Punta de la Barca and Punta del Boy), and the other two belong to group 3 (Laxe do Mouro and Punta del Alba) (Figure 4). In this example, the `regclustcurves` function was used with `algorithm = 'kmeans'` and the input variables y, x, z.

```
> fit.bar <- regclustcurves(y = barnacle5$DW, x = barnacle5$RC, z = barnacle5$F,
               nboot = 500, seed = 300716, algorithm = 'kmeans', cluster = TRUE)

Checking 1 cluster...
Checking 2 clusters...
Checking 3 clusters...

Finally, there are 3 clusters.

> summary(fit.bar)

Call:
regclustcurves(y = barnacle5$DW, x = barnacle5$RC, z = barnacle5$F,
    nboot = 500, algorithm = "kmeans", cluster = TRUE, seed = 300716)

Clustering curves in 3 groups

Number of observations:  5000
Cluster method:  kmeans

Factor's levels:
[1] "laxe"  "lens"  "barca" "boy"   "alba"

Clustering factor's levels:
[1] 2 1 3 3 2
```

```
Testing procedure:
  H0     Tvalue pvalue
1  1 0.94353014  0.000
2  2 0.15463483  0.034
3  3 0.02348982  0.422

Available components:
 [1] "num_groups" "table"      "levels"     "cluster"    "centers"    "curves"
 [7] "method"     "data"       "algorithm"  "call"
```

As can be seen, Figure 4 obtained using the following input command. The specimens from Punta de la Barca and Punta del Boy have similar morphology, wide and short. This is due to these zones present similar oceanographic characteristic, such as exposed rocky shore to waves and highly articulated. Unlike, the barnacles collected from Laxe do Moure and Punta del Alba are narrow and long because they are less exposed locations. Finally, Punta Lens is an intermediate coast, alternating sections of steep coast with large sand.

```
> autoplot(fit.bar, groups_by_color = TRUE)
```

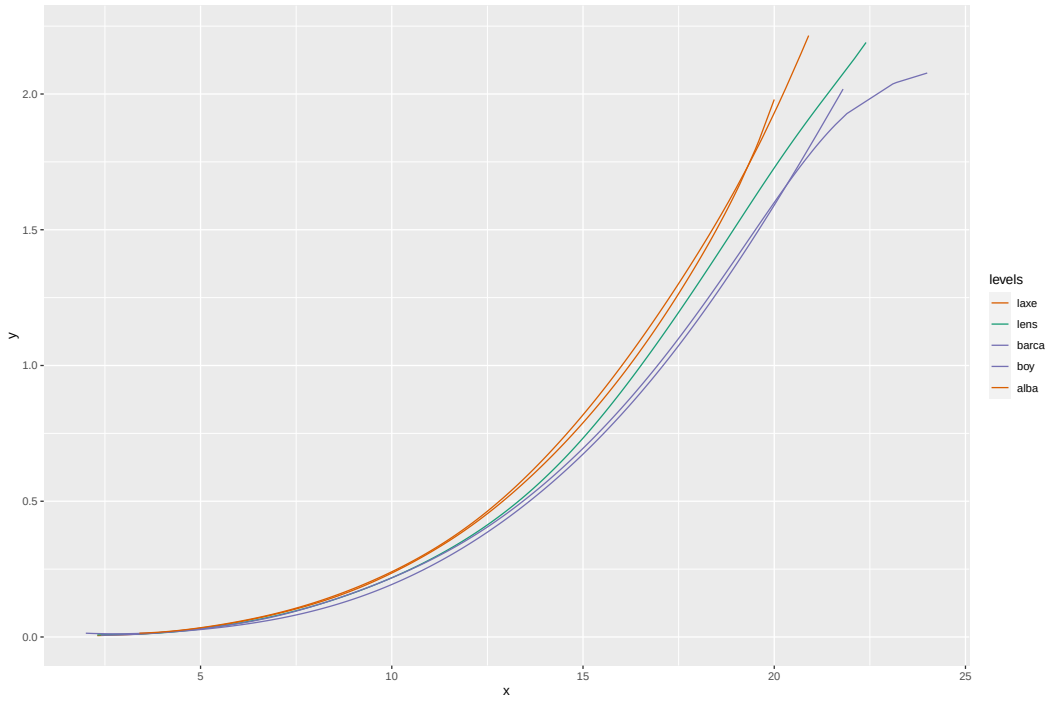


Figure 4: Estimated regression curves for each of the levels of the factor. A specific color is assigned for each curve according to the group to which it belongs using the *K*-means algorithm (in this case, three groups, $K = 3$).

Application to simulated data

Finally, this subsection reports the capabilities of the **clustcurv** package in a more complicated simulated scenario. We consider the regression models given in (1) for $j = 1, \ldots, 30$, with

$$m_j(X_j) = \begin{cases} X_j + 1 & \text{if} \quad j \leq 5 \\ X_j^2 + 1 & \text{if} \quad 5 < j \leq 10 \\ 2\,\sin(2\,X_j) & \text{if} \quad 10 < j \leq 15 \\ 2\,\sin(X_j) & \text{if} \quad 15 < j \leq 20 \\ 2\,\sin(X_j) + a\,\mathrm{e}^{X_j} & \text{if} \quad 20 < j \leq 25 \\ 1 & \text{if} \quad j > 25, \end{cases} \tag{3}$$

where a is a real constant, X_j is the explanatory covariate drawn from a uniform distribution $[-2, 2]$, and ε_j is the error distributed in accordance to a $N(0, \sigma_j(x))$. We have considered the heteroscedastic scenario where $\sigma_j(x) = 0.5 + 0.05 m_j(x) \cdot N(0, 1.5)$.

We explore the methodology considering the null hypothesis $H_0(5)$ of assignment of the m_j curves into five groups ($K = 5$). To show the performance of our procedure, two values were considered for a, 0 and 0.4. It should be noted that the value $a = 0$ corresponds to the null hypothesis, while if $a = 0.4$ the number of groups is six. Particularly, we have defined an unbalanced scenario with unequal sample sizes for each j curve, particularly, $(n_1, n_2, \ldots, n_J) \sim \text{Multinomial}(n; p_1, p_2, \ldots, p_J)$ being $p_j = p_j^* / \sum_{j=1}^{J} p_j^*$, with p_j^* randomly drawn from $\{1, 1.5, 2, 2.5, 3\}$ and $n = 5000$. Note that we propose this procedure for generating the n_j in order to obtain a completely unbalanced study.

The code for the generation of this dataset with $a = 0$ can be found below:

```
> m <- function(x, j) {
   y <- numeric(length(x))
   y[j <= 5] <- x[j <= 5] + 1
   y[j > 5 & j <= 10] <- x[j > 5 & j <= 10] ^ 2 + 1
   y[j > 10 & j <= 15] <- 2 * sin(2 * x[j > 10 & j <= 15]) #- 4
   y[j > 15 & j <= 20] <- 2 * sin(x[j > 15 & j <= 20])
   y[j > 20 & j <= 25] <- 2 * sin(x[j > 20 & j <= 25]) + a * exp(x[j > 20 & j <= 25])
   y[j > 25] <- 1
   return(y)
 }

> seed <- 300716
> set.seed(seed)
> n <- 5000
> a <- 0.0
> x <- runif(n, -2, 2)
> prob <- sample(c(1, 1.5, 2, 2.5, 3), 30, replace = TRUE)
> prob <- prob/sum(prob)
> f <- sample(1:30, n, replace = TRUE, prob = prob)
> N <- length(unique(f))
> error <- rnorm(n,0,1.5)
> y <- m(x, f) + (0.5 + 0.05 * m(x, f)) * error
> data <- data.frame(x, y, f)
```

In order to determine groups of the generated curves, the user has to execute the next piece of code. As expected, when $a = 0$, the number of groups selected is five.

```
> fit.sim <- regclustcurves(x = data$x, y = data$y,  z = data$f, nboot = 500,
                      algorithm = 'kmedians', cluster = TRUE, seed = 300716)
Checking 1 cluster...
Checking 2 clusters...
Checking 3 clusters...
Checking 4 clusters...
Checking 5 clusters...

Finally, there are 5 clusters.
> fit.sim

Call:
regclustcurves(y = data$y, x = data$x, z = data$f, nboot = 500,
    algorithm = "kmedians", cluster = TRUE, seed = 300716)

Clustering curves in 5 groups

Number of observations:  5000
Cluster method:  kmedians
> autoplot(fit.sim, groups_by_colour = TRUE, centers = TRUE)
```

Additionally, for different values of a (a > 0), our procedure should determine 6 groups. For instance, for $a = 0.4$, it selects the true number of groups ($K = 6$) typing the commands below:

```
> seed <- 300716
> set.seed(seed)
> n <- 5000
> a <- 0.4
> x <- runif(n, -2, 2)
```

```
> prob <- sample(c(1, 1.5, 2, 2.5, 3), 30, replace = TRUE)
> prob <- prob/sum(prob)
> f <- sample(1:30, n, replace = TRUE, prob = prob)
> N <- length(unique(f))
> error <- rnorm(n,0,1.5)
> y <- m(x, f) + (0.5 + 0.05 * m(x, f)) * error
> data2 <- data.frame(x, y, f)
> fit.sim2 <- regclustcurves(x = data2$x, y = data2$y,  nboot = 500, seed = 300716,
                          z = data$f, algorithm = 'kmedians', cluster = TRUE)
Checking 1 cluster...
Checking 2 clusters...
Checking 3 clusters...
Checking 4 clusters...
Checking 5 clusters...
Checking 6 clusters...

Finally, there are 6 clusters.
> fit.sim2

Call:
regclustcurves(y = data2$y, x = data2$x, z = data$f, nboot = 500,
    algorithm = "kmedians", cluster = TRUE, seed = 300716)

Clustering curves in 6 groups

Number of observations:  5000
Cluster method:  kmedians
> autoplot(fit.sim2, groups_by_colour = TRUE, centers = TRUE)
```

Figures 5 and 6 show the results with the simulated data with $a = 0$ and $a = 0.4$, respectively. In this situation, the true number of groups is equal to 5 and 6. As can be appreciated, our method seems to perform reasonably well for both values of a. For $a = 0$, the null hypothesis $H_0(5)$ is accepted, curves assigned to each group are plotted with the same color. In the case of $a = 0.4$, the null hypothesis $H_0(6)$ is accepted. Therefore, there are 6 groups of regression curves. Note that in both plots, the centroids are colored in black because in the `autoplot` function, the argument `centers = TRUE`.

Conclusion and further extensions of the R package

This paper discussed the implementation of some methods developed for determining groups of multiple nonparametric curves in the R package **clustcurv**. In particular, the methods proposed are focused on the framework of regression analysis and in the framework of survival analysis. In the context of survival analysis, we restrict ourselves to survival curves. Hopefully, future versions of the package will extend the methodology to determine groups in risk functions, cumulative hazard curves, or density functions. The current version of the package implements two optimization algorithms, the well-known *K*-means and *K*-medians. It can be interesting to let the user choose far from those, such as Means-Shift or *K*-medoids algorithms.

Acknowledgements

The authors acknowledge financial support by the Spanish Ministry of Economy and Competitiveness (MINECO) through project MTM2017-89422-P and MTM2017-82379-R (funded by (AEI/FEDER, UE). Thanks to the Associate Editor and the referee for comments and suggestions that have improved this paper.

Bibliography

Z. Chen and G. Zhang. Comparing survival curves based on medians. *BMC Medical Research Methodology*, 16(1):33, 2016. ISSN 1471-2288. doi: 10.1186/s12874-016-0133-3. URL http://dx.doi.org/10.1186/s12874-016-0133-3. [p164]

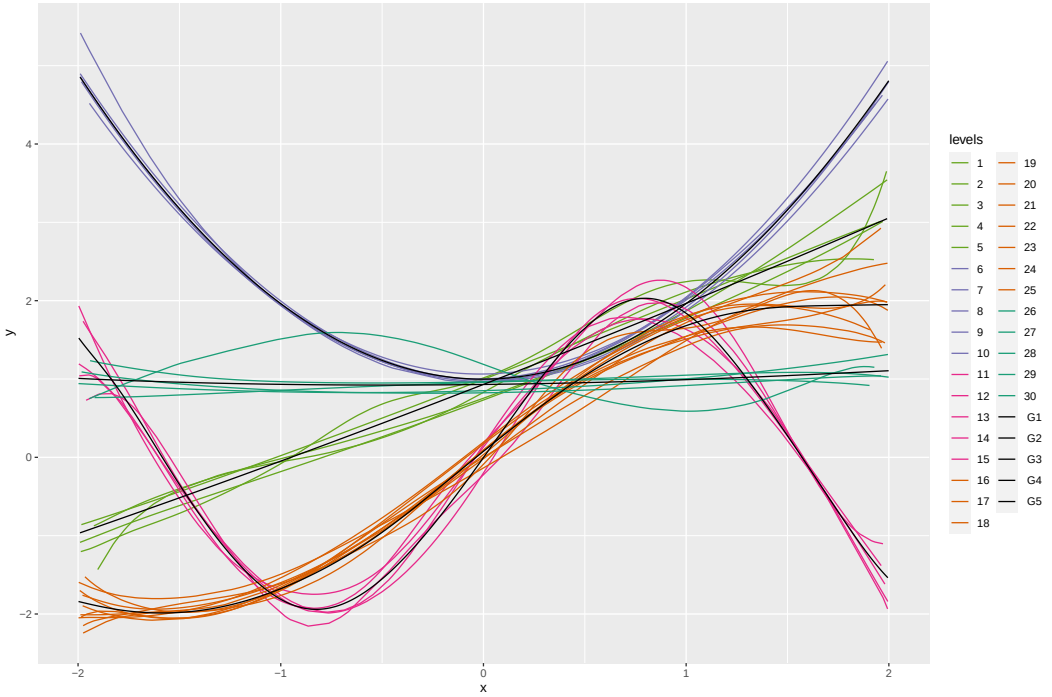


Figure 5: Estimated regression curves for each of the levels of the variable f with $a = 0$. A specific color is assigned for each curve according to the group to which it belongs using the K-means algorithm (in this case, five groups, $K = 5$). Black curves are the centroids of each group.

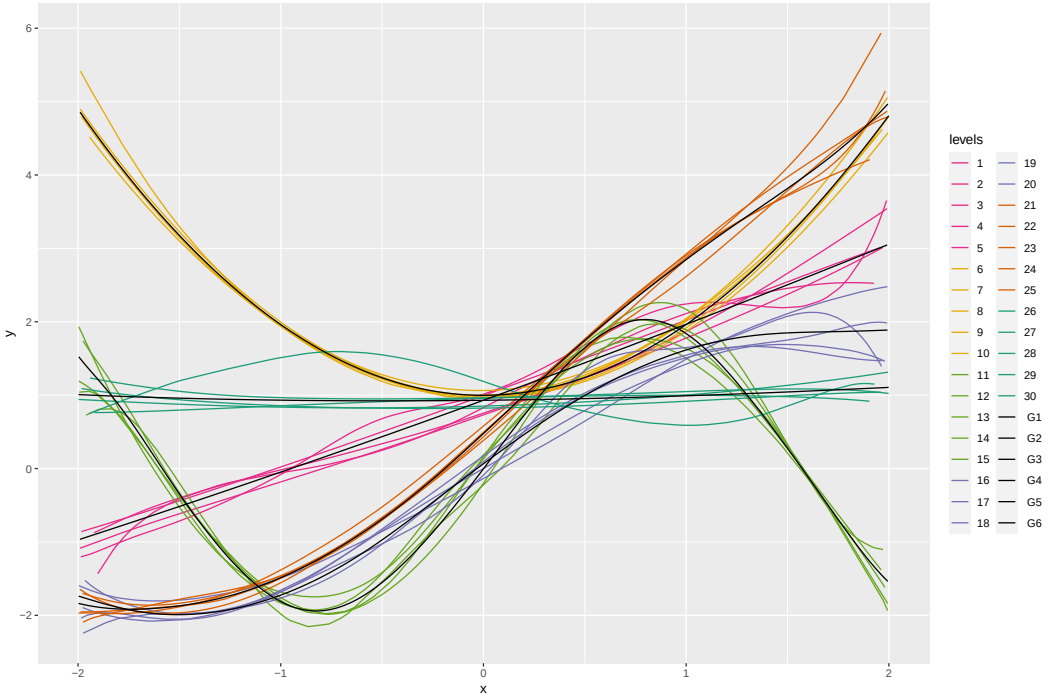


Figure 6: Estimated regression curves for each of the levels of the variable f with $a = 0.4$. A specific color is assigned for each curve according to the group to which it belongs using the K-means algorithm (in this case, six groups, $K = 6$). Black curves are the centroids of each group.

C. A. de Boor. *A Practical Guide to Splines*. Springer Verlag, New York, 2001. [p166]

M. A. Delgado. Testing the equality of nonparametric regression curves. *Statistics and Probability Letters*, 17:199–204, June 1993. [p164]

D. Dette and N. Neumeyer. Nonparametric analysis of covariance. *The Annals of Statistics*, 29:1361–1400, 2001. [p164]

S. Dudoit and M. J. van der Laan. *Multiple Testing Procedures with Applications to Genomics*. Springer Series in Statistics. Springer, New York, 2008. [p167]

B. Efron. Bootstrap methods: another look at the jackknife. *The Annals of Statistics*, 7:1–26, 1979. [p167]

J. Fan and I. Gijbels. *Local polynomial modelling and its applications*. Number 66 in Monographs on statistics and applied probability series. Chapman & Hall, 1996. [p166]

T. R. Fleming, D. P. Harrington, and M. O'Sullivan. Supremum versions of the log-rank and generalized wilcoxon statistics. *Journal of the American Statistical Association*, 82(397):312–320, 1987. ISSN 01621459. URL http://www.jstor.org/stable/2289169. [p164]

E. A. Gehan. A generalized wilcoxon test for comparing arbitrarily singly censored samples. *Biometrika*, 52:203–223, 1965. ISSN 00063444. URL http://www.jstor.org/stable/2333825. [p164]

W. González-Manteiga and R. Cao. Testing the hypothesis of a general linear model using nonparametric regression estimation. *Test*, 2(1):223–249, 1993. [p164]

P. Hall and J. D. Hart. Bootstrap test for difference between means in nonparametric regression. *Journal of the American Statistical Association*, 85(412):1039–1049, Dec. 1990. [p164]

W. Härdle and E. Mammen. Testing parametric versus nonparametric regression. *Annals of Statistics*, 21:1926–1947, 1993. [p164]

D. P. Harrington and T. R. Fleming. A class of rank test procedures for censored survival data. *Biometrika*, 69(3):553, 1982. doi: 10.1093/biomet/69.3.553. URL +http://dx.doi.org/10.1093/biomet/69.3.553. [p164]

S. Holm. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6: 65–70, 1979. [p168]

E. L. Kaplan and P. Meier. Nonparametric estimation from incomplete observations. *Journal of the American Statistical Association*, 53:457–481, 1958. [p165]

A. Kassambara, M. Kosinski, P. Biecek, and S. Fabian. survminer: Drawing Survival Curves using 'ggplot2'. R package version 0.4.6, 2019. URL https://cran.r-project.org/web/packages/survminer. [p164]

L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis.* John Wiley, 1990. [p166]

E. King, J. D. Hart, and T. E. Wehrly. Testing the equality of two regression curves using linear smoothers. *Statistics and Probability Letters*, 12(3):239–247, 1991. ISSN 0167-7152. [p164]

K. B. Kulasekera. Comparison of regression curves using quasi-residuals. *Journal of the American Statistical Association*, 90(431):1085–1093, 1995. ISSN 01621459. [p164]

S. Lang and A. Brezger. Bayesian p-splines. *Journal of Computational and Graphical Statistics*, 13:183–212, 2004. [p166]

R. Y. Liu. Bootstrap Procedures under some Non-I.I.D. Models. *The Annals of Statistics*, 16(4):1696–1708, 1988. URL http://www.jstor.org/stable/2241788. [p167]

J. B. Macqueen. *Some methods of classification and analysis of multivariate observations*, volume 1. Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability (Univ. of Calif. Press), 1967. [p166]

E. Mammen. Bootstrap and Wild Bootstrap for High Dimensional Linear Models. *The Annals of Statistics*, 21(1):255–285, 1993. URL http://www.jstor.org/stable/3035590. [p167]

N. Mantel. Evaluation of survival data and two new rank order statistics arising in its consideration. *Cancer Chemotherapy Reports*, 50(3):163–170, 1966. [p164]

N. Neumeyer and H. Dette. Nonparametric comparison of regression curves: An empirical process approach. *The Annals of Statistics*, 31(3):880–920, 2003. ISSN 00905364. [p164]

J. C. Pardo-Fernández, I. Van Keilegom, and W. González-Manteiga. Testing for the equality of k regression curves. *Statistica Sinica*, 17:1115–1137, 2007. [p164]

R. Peto and J. Peto. Asymptotically efficient rank invariant test procedures (with discussion). *Journal of the Royal Statistical Society, Series A*, 135:185–206, 1972. [p164]

M. Rosenblatt. A quadratic measure of deviation of two-dimensional density estimates and a test of independence. *The Annals of Statistics*, 3(1):1–14, 01 1975. [p164]

R. Srihera and W. Stute. Nonparametric comparison of regression functions. *Journal of Multivariate Analysis*, 101:2039–2059, October 2010. ISSN 0047-259X. [p164]

R. E. Tarone and J. Ware. On distribution-free tests for equality of survival distribution. *Biometrika*, 64: 156–160, 1977. [p164]

N. M. Villanueva, M. Sestelo, and L. Meira-Machado. A Method for Determining Groups in Multiple Survival Curves. *Statistics in Medicine*, 38:366–377, 2019. [p164, 165]

N. M. Villanueva, M. Sestelo, C. Ordoñez, and J. Roca-Pardiñas. An Approach to Determine Groups of Multiple Regression Curves. Manuscript submitted for publication, 2019. [p165]

M. P. Wand and M. C. Jones. *Kernel Smoothing*. Chapman & Hall: London, 1995. [p166]

H. Wickham, W. Chang, L. Henry, T. L. Pedersen, K. Takahashi, C. Wilke, K. Woo, and H. Yutani. ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics. R package version 3.2.1, 2019. URL https://cran.r-project.org/web/packages/ggplot2. [p169]

C. F. J. Wu. Jackknife, Bootstrap and other resampling methods in regression analysis. *The Annals of Statistics*, 14(4):1261–1295, 1986. doi: 10.2307/2241454. URL http://dx.doi.org/10.2307/2241454. [p167]

S. G. Young and A. W. Bowman. Nonparametric analysis of covariance. *Biometrics*, 51:920–931, 1995. [p164]

F. Zhan, Y. Huang, S. Colla, J. P. Stewart, I. Hanamura, S. Gupta, J. Epstein, S. Yaccoby, J. Sawyer, B. Burington, E. Anaissie, K. Hollmig, M. Pineda-Roman, G. Tricot, F. van Rhee, R. Walker, M. Zangari, J. Crowley, B. Barlogie, and J. Shaughnessy, John D. The molecular classification of multiple myeloma. *Blood*, 108(6):2020–2028, 09 2006. ISSN 0006-4971. doi: 10.1182/blood-2005-11-013458. [p174]

Nora M. Villanueva
Department of Statistics and OR,
SiDOR research group & CINBIO
University of Vigo, Spain
ORCiD: 0000-0001-8085-2745
http://noramvillanueva.github.io
nmvillanueva@uvigo.es

Marta Sestelo
Department of Statistics and OR,
SiDOR research group & CINBIO
University of Vigo, Spain
ORCiD: 0000-0003-4284-6509
http://sestelo.github.io
sestelo@uvigo.es

Luís Meira-Machado
Department of Mathematics
Centre of Mathematics
University of Minho, Portugal
lmachado@math.uminho.pt

Javier Roca-Pardiñas
Department of Statistics and OR,
SiDOR research group & CINBIO
University of Vigo, Spain
roca@uvigo.es

Benchmarking R packages for Calculation of Persistent Homology

by Eashwar V. Somasundaram, Shael E. Brown, Adam Litzler, Jacob G. Scott, and Raoul R. Wadhwa

Abstract Several persistent homology software libraries have been implemented in R. Specifically, the Dionysus, GUDHI, and Ripser libraries have been wrapped by the **TDA** and **TDAstats** CRAN packages. These software represent powerful analysis tools that are computationally expensive and, to our knowledge, have not been formally benchmarked. Here, we analyze runtime and memory growth for the 2 R packages and the 3 underlying libraries. We find that datasets with less than 3 dimensions can be evaluated with persistent homology fastest by the GUDHI library in the **TDA** package. For higher-dimensional datasets, the Ripser library in the TDAstats package is the fastest. Ripser and **TDAstats** are also the most memory-efficient tools to calculate persistent homology.

Introduction

Topological data analysis (TDA) is a broad set of methodologies that characterizes structural features of datasets inspired by topological principles. It has a broad range of usage, from viral evolution to physical chemistry (Chan et al., 2013; Offroy and Duponchel, 2016). Within the umbrella of TDA, persistent homology represents an algebraic approach to understanding the number, characteristics, and persistence of structural features in an n-dimensional point cloud. In the basic workflow of persistent homology, a series of simplicial complexes are generated on point clouds to characterize topological features. There are several methods to generate these complexes on point clouds. In this paper, we focus on persistent homology of the Vietoris-Rips and alpha complexes, which use simplicial complexes to approximate topologic relationships in point clouds. The exact method of constructing these complexes is described in the Mathematics section. Essentially, we measure features that are discovered by the algorithm at a particular stage and disappear at a later stage. The difference between these stages is persistence. Features with larger persistence more likely represent real geometric patterns rather than noise.

There are several C++ libraries available to researchers that calculate alpha and Vietoris-Rips complexes, such as Dionysus, GUDHI, and Ripser (Morozov, 2018; Maria et al., 2016; Bauer, 2019). These libraries have been wrapped in R by the **TDA** and **TDAstats** packages (Fasy et al., 2019; Wadhwa et al., 2018). Although useful, calculating persistent homology for large datasets is often limited due to computational complexity (Otter et al., 2017). As a result, researchers often limit persistent homology analysis to lower dimensions. However, ignoring features in higher dimensions may cause significant information loss, underutilizing persistent homology's capabilities. Here, we aim to benchmark two R packages - **TDA** and **TDAstats** - and enable researchers to most efficiently calculate persistent homology in R.

Mathematics of Persistent Homology

An n-dimensional simplex is the convex hull of $n+1$ points in a Euclidean space. More intuitively, an n-dimensional simplex is the simplest n-dimensional object (e.g., a 0-simplex is a point, a 1-simplex is a line, a 2-simplex is a triangle, 3-simplex is a tetrahedron). These simplices can be glued together on common sub-simplices to form a simplicial complex (e.g., two triangles sharing a common side). In a simplicial complex, topological features will arise that can be characterized by Betti numbers. Each Betti number, denoted by B_k, k counts the number of features in dimension k. B_0 counts the number of connected components, B_1 counts refer to loops, B_2 counts the number of voids, and so on (Edelsbrunner and Harer, 2008).

There are several different methods to construct a simplicial complex on a given point cloud S, but this paper focuses on the Vietoris-Rips and alpha complexes. The Vietoris-Rips complex is perhaps the most common method for constructing a simplicial complex to calculate persistent homology (Hausmann, 1996). In a point cloud of k points in 2 dimensions, a distance parameter, $\delta > 0$, can be used to draw a circle of diameter δ around every point in S. For point clouds in 3 dimensions, spheres of diameter δ are drawn around each point. For dimensions k greater than 3, a k-dimensional hypersphere is drawn around each point. The remainder of this explanation will focus on the 2-dimensional case. If δ is sufficiently large, then some of the resulting circles may intersect. In this case, a line is drawn to connect the points at the center of the intersecting circles. When a triple of points is connected, we add a triangle (2-simplex). When a quadruple of points are connected, we add a

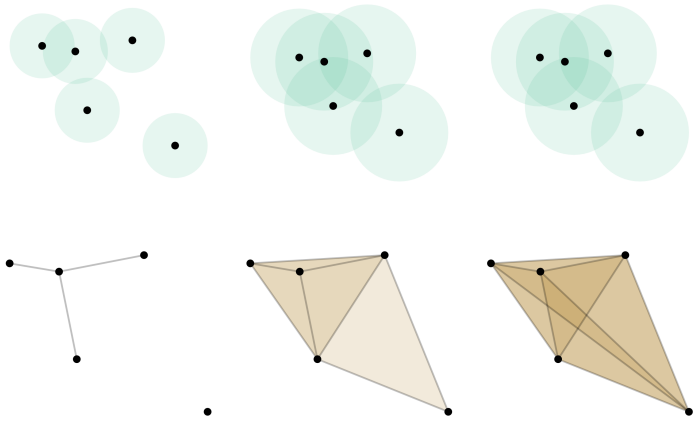


Figure 1: Basic Visualization of the Vietoris-Rips Complex. For a given parameter, δ, δ-diameter circles are drawn around each point. If two circles intersect, a point is drawn between their centers. As δ continues to grow, more circles intersect, filling out the simplicial complex. Features on the simplicial complex appear and die as δ increases. These features' dimensions, birth, and death are recorded in an nx3 matrix. Eventually, the full convex hull is drawn, ending the "filtration" process.

tetrahedron (3-simplex) and so forth. However, we only add simplices at most of the dimension of the space of the point clouds (e.g., only up to 3-simplices are added in a 3-dimensional point cloud). This group of points and lines form the skeleton of a simplicial complex. For each distance parameter, δ, there will be a single simplicial complex associated with it. As δ increases, different topological features may appear, persist, and eventually disappear.

Once δ reaches the maximum Euclidean distance between any pair of points in the point cloud, a convex hull will form around all k points creating a $(k-1)$-dimensional simplex. A 3-column matrix can be created recording the dimension of each feature, the δ at which that feature appeared, and the δ at which it disappeared. This matrix characterizes the persistent homology of that point cloud.

Alpha complexes provide another method to generate simplicial complexes on the point cloud S. For alpha complexes, we partition the whole space in which the data resides into cells such that each cell contains exactly one data point x, and the cell of that data point is the set of all points closer to x than any other data points. Such a partition is also known as a Voronoi diagram. The nerve of a Voronoi diagram is equivalent to the Delaunay Triangulation (Edelsbrunner and Mücke, 1994). Alpha complexes are simplicial complexes that are subsets of the Delaunay Triangulation. The parameter, α, can describe the radius of a ball (dimension matches dimension of the space) of each point in the point cloud S much, like δ describes the diameter of a circle in the Vietoris-Rips complex. We first intersect the α radius balls with their own Voronoi cell and then search for intersections of these subsetted balls to form simplices. Once α is large enough, the full Delaunay Triangulation is formed. In between these stages, the birth and death of features at certain values of α can be captured in a 3-column persistent homology matrix much like the Vietoris-Rips complex. One key difference from the Vietoris-Rips complex is that edges can only form between neighboring points in the alpha complex.

In both methods, the boundary matrix records all simplicial complexes for each parameter value (δ for Vietoris-Rips complexes and α for alpha complexes). Calculating persistent homology is divided

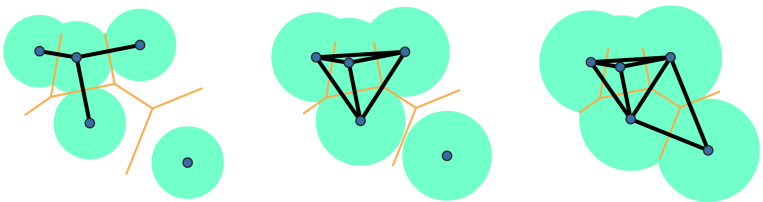


Figure 2: Basic visualization of the Alpha complex. For a given α, α-radius balls are drawn around each point, and the union of the balls is taken. Then, an intersection between this union of α-balls and the Vornoi diagram is taken. A connecting segment is drawn between points in adjacent Voronoi cells once the α-ball fills out the Voronoi diagram. As α grows, more circles fill out the Voronoi cells. Once α is large enough, the Delaunay Triangulation is formed.

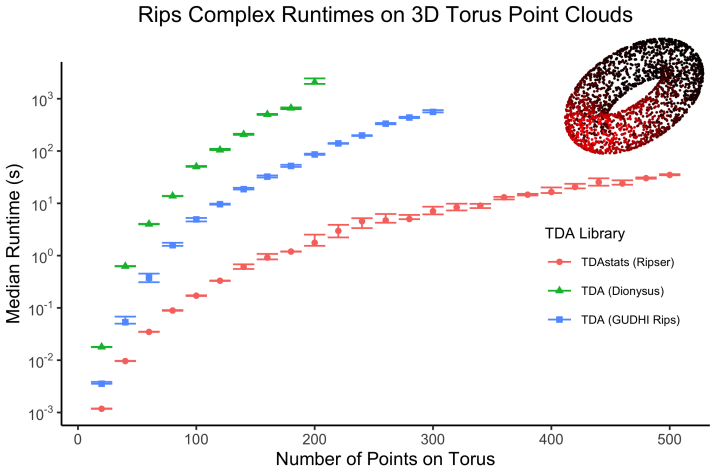


Figure 3: Calculating persistent homology of a torus with three TDA libraries. Median runtime (min to max, $n = 10$ iterations per data point) for each TDA library (denoted by color) is plotted against point cloud size. Homological features of up to 2 dimensions were calculated. Time complexity follows a power law for all three libraries (see GitHub repo for regression details). Although the libraries have similar runtimes for smaller point clouds, Dionysus has a clear disadvantage when the number of points exceeds 100. When the number of points exceeds 200, Ripser has a clear advantage over GUDHI, which maintains its advantage over Dionysus.

into two steps: (1) forming the boundary matrix and (2) reducing the boundary matrix to be able to read off the topological features of each dimension and their birth/death values of the parameter. The second step can be computed in at most O(k^3) steps, where k is the number of rows (and columns) of the boundary matrix. The size of the boundary matrix can describe the memory complexity on the random access memory (RAM) for persistent homology calculations. We compare memory complexity between alpha and Vietoris-Rips complexes in this paper.

Alpha complex calculations have a run time complexity of $O(n^{d/2})$, and Vietoris-Rips complex calculations have a run time complexity of $O(2^n)$, where n is the number of points and d is the point cloud dimension (Otter et al., 2017). Vietoris-Rips's run time and memory are exponential with regards to point number (but constant with data dimension) in contrast to alpha complexes where run time and memory are polynomial with point number (but exponential with data dimension). Therefore, we can predict that low dimensional point clouds favor alpha complexes, but fewer points in higher dimension favor Vietoris-Rips complexes.

Methods

We use **readr** v1.3.1 to read rectangular data (Wickham et al., 2018), **ggplot2** v3.2.1 (Wickham, 2016), **scatterplot3d** v0.3-41 (Ligges and Mächler, 2003), **recexcavAAR** v0.3.0 (Schmid and Serbe, 2017), **deldir** v0.1 (Turner, 2020), ggtda v0.1 (Brunson et al., 2020), and **magick** v2.2 (Ooms, 2019) to visualize data, **bench** v1.0.4 to collect benchmark data (Hester, 2019), **TDA** v1.6.9 (Fasy et al., 2019) and **TDAstats** v0.4.1 (Wadhwa et al., 2018) to calculate persistent homology of Vietoris-Rips and alpha simplicial complices, and **pryr** v0.1.4 for calculations involving R objects (Wickham, 2018). Median runtime calculations are shown along with the minimum and maximum of 10 iterations per benchmark. Datasets were generated by sampling functions in base R to generate points uniformly distributed over circles (dimension = 2), spheres (dimension = 3), filled squares (dimension = 2), filled cubes (dimension = 3, 4), and tori (dimension = 3). The number of points per point cloud varied from 25 to 500 along with intervals of 25 points, which were empirical limits chosen after considering available computational resources. For consistency between software libraries, the minimum and maximum simplicial complex radii were predetermined for each point cloud and provided as parameters to the **TDA** and **TDAstats** R packages. Within the **TDA** package, benchmark data was collected for the GUDHI (Maria et al., 2016) and Dionysus (Morozov, 2018) libraries; within the **TDAstats** package, benchmark data was collected for the Ripser (Bauer, 2019) library. As alpha complex calculation was only implemented in GUDHI, alpha complex benchmark data was naturally only collected for the single library. Measuring memory usage proved challenging since all the libraries calculating persistent homology were implemented in either C++ or Java and then wrapped in R as part of a CRAN package. Thus, memory burden was indirectly measured by using boundary matrix size as a

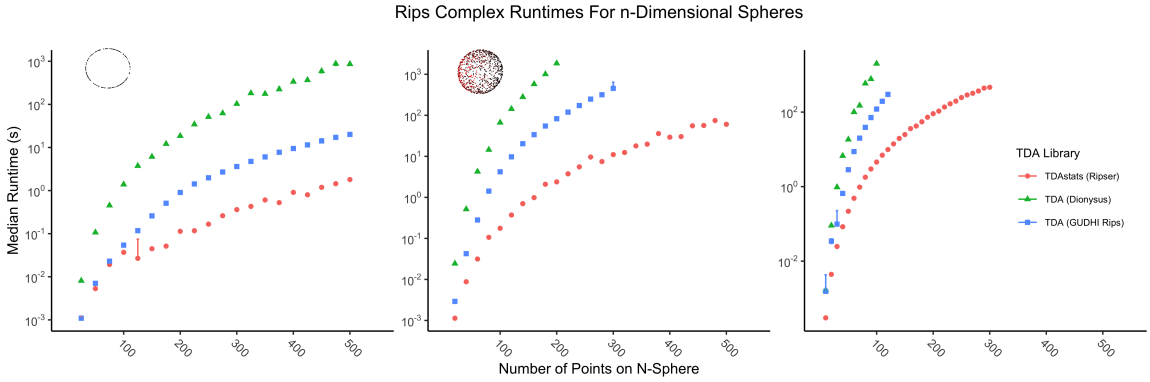


Figure 4: Calculating persistent homology of round point clouds of varying dimensions with three TDA libraries. Median runtime (min to max, $n = 10$ iterations per data point) for each TDA library (denoted by color) is plotted against point cloud size and faceted by data dimension. The left panel compares library performance for a 2-dimensional circular point cloud, the center panel for a 3-dimensional spherical point cloud, and the right panel for a 4-dimensional hyperspherical point cloud. Maximum feature dimensions (one less than the data dimension) were calculated in each case.

proxy. Given that Ripser optimizes computation of persistent homology by avoiding calculation of a boundary matrix, memory use benchmarks are not provided for Ripser and, consequently, **TDAstats**.

Benchmark data were collected twice - once on a local machine and once on a remote computing node, each of which featured 16 GB RAM. Both datasets were compared for consistency and are publicly available at the repository linked below. Data from the remote computing node is visualized in this report. The larger point clouds required more than 16 GB of RAM to calculate persistent homology using a subset of the libraries; attempts to compute results resulted in runtime errors, and the corresponding output is missing from the corresponding figures and tables. Fully reproducible code for all numerical results and figures can be found at `https://github.com/eashwarsoma/TDA-benchmark`. This GitHub repository also contains instructions for generating the Supplement referenced in this report's results. Video explanations of TDA concepts and reproducing all results in this report can be found at `https://tinyurl.com/TDABench`.

Results

Computing persistent homology of a canonical torus grants quick insight into efficiency of each library (Figure 3). Dionysus exhibits the longest median runtime, and, although Ripser and GUDHI have similar runtimes for smaller point clouds, as the number of points increases Ripser eventually has a significant lead. Next, we compare library performance with multiple canonical datasets to ensure that the noted pattern generalizes.

Tori do not trivially generalize to other dimensions, but circles do. Benchmarking on a circular point cloud permits confirmation of the pattern in Figure 3 while also revealing how the libraries compare as the dataset dimension increases. Figure 4 exhibits the resulting data for a 2-dimensional circle (left panel), 3-dimensional sphere (center panel), and 4-dimensional hypersphere (right panel). When the dataset dimension equals 2, GUDHI practically matches Ripser's performance in outpacing Dionysus. However, in the case of the 3-dimensional sphere, the pattern visualized in Figure 3 for the 3-dimensional torus is again present. By the 4th dimension, the gap between Ripser and GUDHI widens. Of note, missing points for larger datasets in Figure 4 are not plotted if and only if calculating persistent homology caused an error due to insufficient RAM. Thus, for the hypersphere, Ripser was able to calculate persistent homology for a dataset with approximately 3 times as many points as Dionysus and over 2 times as many as GUDHI. Interestingly, all curves plotted in Figure 4 grow polynomially with respect to the number of points (see Supplement for regression details).

Large data and feature dimensionality often restrict persistent homology calculations to small point clouds due to computational limits. When calculating persistent homology on a high-dimensional point cloud, as Vietoris-Rips feature dimension increases, there is a corresponding increase in runtime (Figure 5). Dionysus is clearly outmatched by GUDHI and Ripser as feature dimension increases, with the difference being clearest for larger point clouds; by feature dimension 5, Ripser outpaces GUDHI as well (Figure 5). It is unclear whether runtime for each library grows polynomially or exponentially (see Supplement for regression details).

Even with a constant feature dimension, the underlying data dimension could play a role in

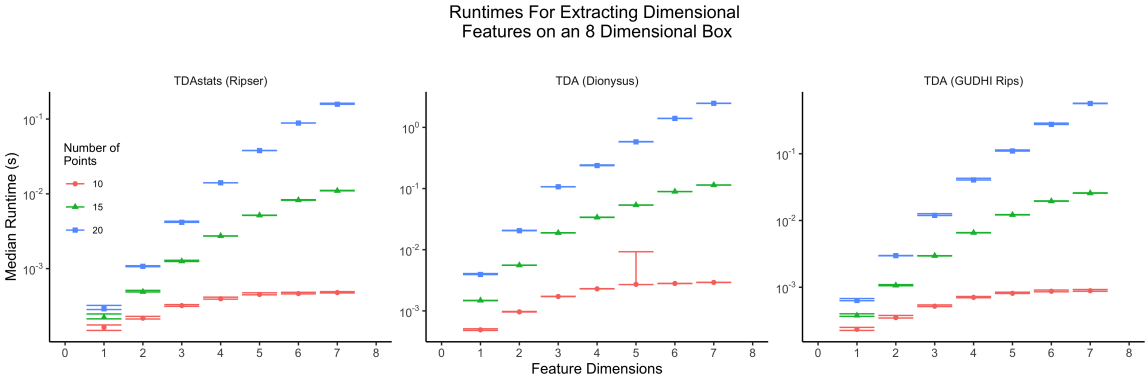


Figure 5: Comparison of Vietoris-Rips complex persistent homology calculation as a function of feature dimension. Median runtime (min to max, $n = 10$ iterations per data point) for various point cloud sizes (denoted by color) is plotted against the calculated feature dimension and faceted by TDA library. Persistent homology was calculated on a uniformly distributed random sample of points contained within a 1 unit, 8-dimensional cube. Computational limitations of calculating persistent homology for a large number of feature dimensions restricted point clouds to relatively small sizes.

the runtime of persistent homology calculation. Figure 6 compares the handling of this issue by the Vietoris-Rips complex and the alpha complex. Since GUDHI is the only library implementing functionality with an alpha complex, we compare its implementations of the Vietoris-Rips and alpha complices. Due to computational limitations, an alpha complex could not be calculated for any point clouds with data dimensions exceeding 3. Two notable aspects of Figure 6 stand out. First, the alpha complex calculation clearly runs faster than the Vietoris-Rips complex calculation, a trend that becomes clearer as point cloud size increases. Second, although the Vietoris-Rips complex calculation runtime appears to be independent of the underlying data dimension, the alpha complex calculation is dependent on it. Figure 6 shows a subtle difference between data dimensions 2 and 3 as point cloud size increases. Although unconcerning for a data dimension up to 3, failure to run any alpha complex calculations with a data dimension of 4 could be cause for concern.

In addition to runtime differences, the three Vietoris-Rips homology engines differ in memory use. All three engines appeared to follow power law growth, with a linear trend on log-log plots (Figure 7). However, for nearly all combinations of point cloud dimension and shape, TDAstats used the least memory, and Dionysus used the most, with TDAstats also growing with the smallest power law exponent as the number of points increased for most point clouds. For most point clouds, runtime and memory complexity for TDAstats (Ripser) grew with a power function at least one degree less than the other engines (Figure 8).

Discussion

As persistent homology calculations continue to become a more popular tool to analyze complex multidimensional data, it will be important to understand from a computational perspective which method to use. In this paper, we examined two forms of persistent homology complexes: Vietoris-Rips and alpha complexes. Both algorithms describe topological features through the generation of simplicial complexes. The advantage in saving computational time by choosing a particular algorithm depends on point cloud characteristics.

Figure 9 shows that at high point cloud sizes, GUDHI's alpha complex outperforms Ripser. Theoretically, alpha complexes gain polynomial run time complexity as the number of points increases, whereas Vietoris-Rips complexes gain exponential run time complexity (Otter et al., 2017). Specifically, alpha complexes are $O(n^{d/2})$, and Vietoris-Rips complexes are $O(2^n)$, where n is the number of points on a point cloud and d is the dimensions on the point cloud. For the conditions in our paper, Vietoris-Rips and alpha complexes both performed better than their theoretical maximums. Vietoris-Rips complex calculations consistently had a polynomial growth for both runtime and memory, while alpha complexes had linear runtime growth.

Based on the theoretical complexity and our results, alpha complexes are superior for point clouds with 3 or fewer dimensions. This advantage becomes especially clear at a high number of points. This difference in performance is clear in both runtime and memory use. Interestingly, while alpha complexes had overall less memory use, the memory use varied depending on the shape. Alpha complexes seem to require more memory for noisier data sets such as the annulus when compared to

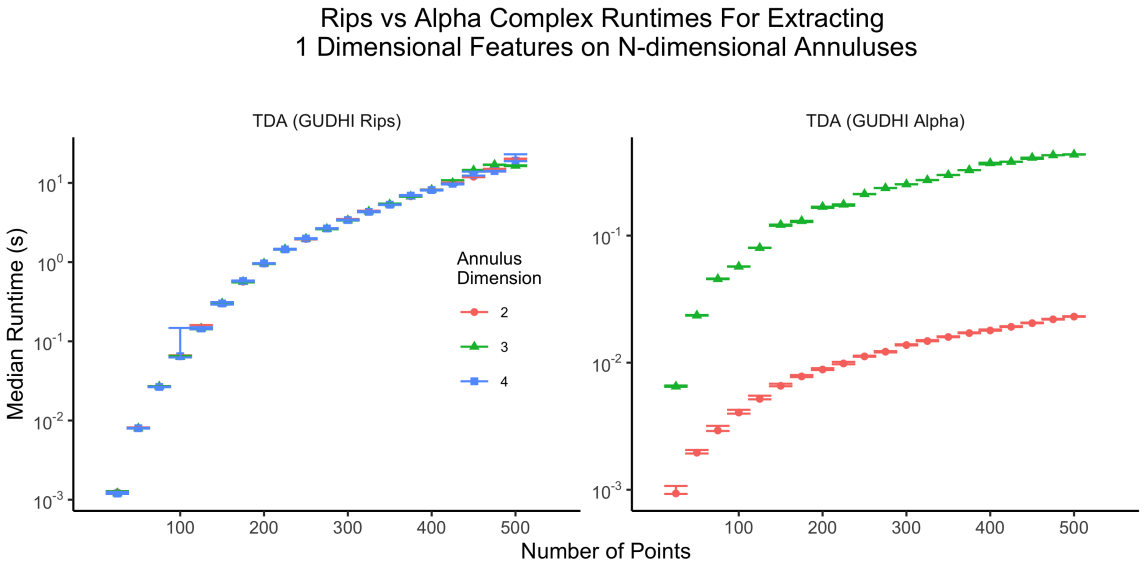


Figure 6: Comparing persistent homology calculation between Vietoris-Rips and alpha complices. Median runtime (min to max, $n = 10$ iterations per data point) for various data dimensions (denoted by color) are plotted against point cloud size and faceted by type of simplicial complex. Maximum of feature dimension was kept constant at 1. Alpha complex runtimes are linear, in contrast to polynomial Vietoris-Rips runtimes (see Supplement for regression details).

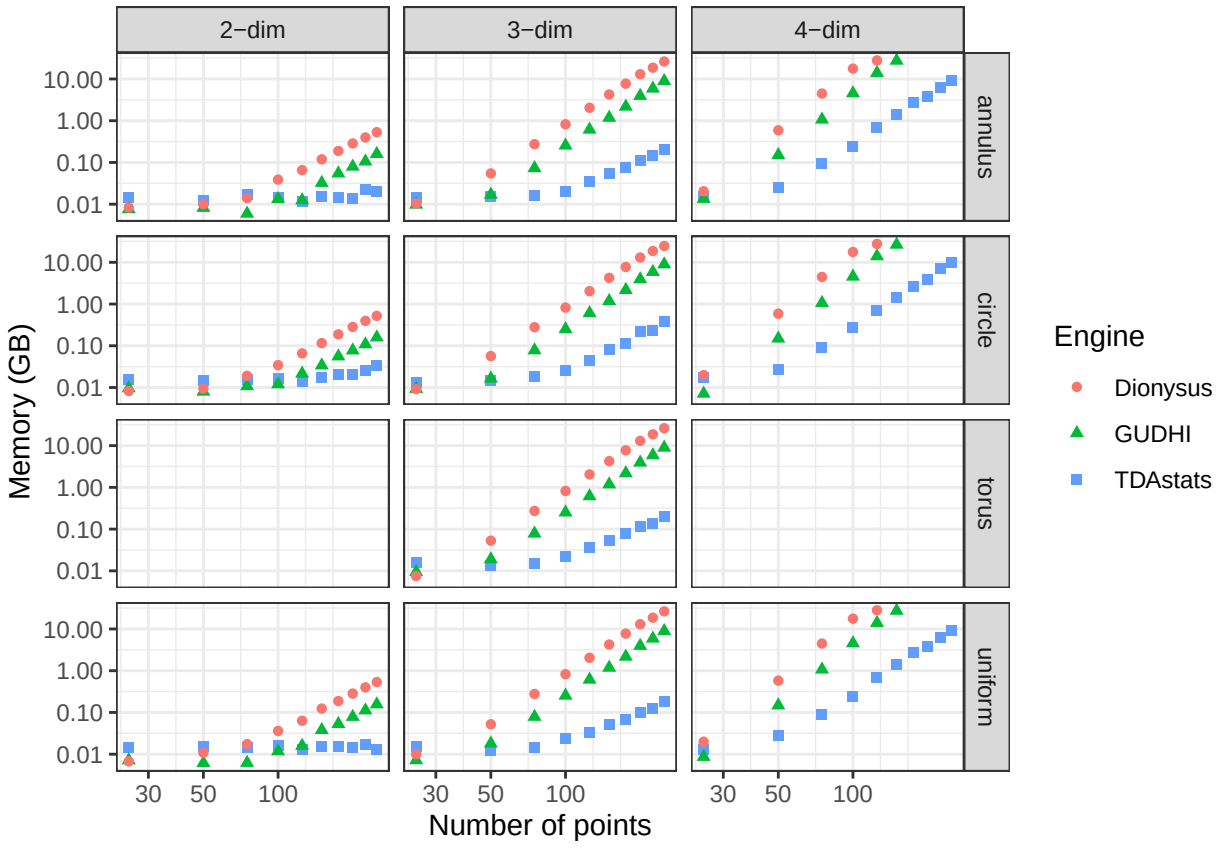


Figure 7: Comparing memory use of Vietoris-Rips persistent homology engines. Each column title corresponds to point cloud dimension; each row title lists point cloud shape; each persistent homology engine is represented by points of a distinct shape and color. For point clouds containing more than 50 points, there appears to be a linear trend on the log-log axes. Data for 2- and 4-dimensional tori were not collected because a torus does not trivially generalize to dimensions other than 3. Missing points for GUDHI and Dionysus in the 4-dim plots indicate that persistent homology calculation was terminated since memory requirement exceeded available RAM (32 GB).

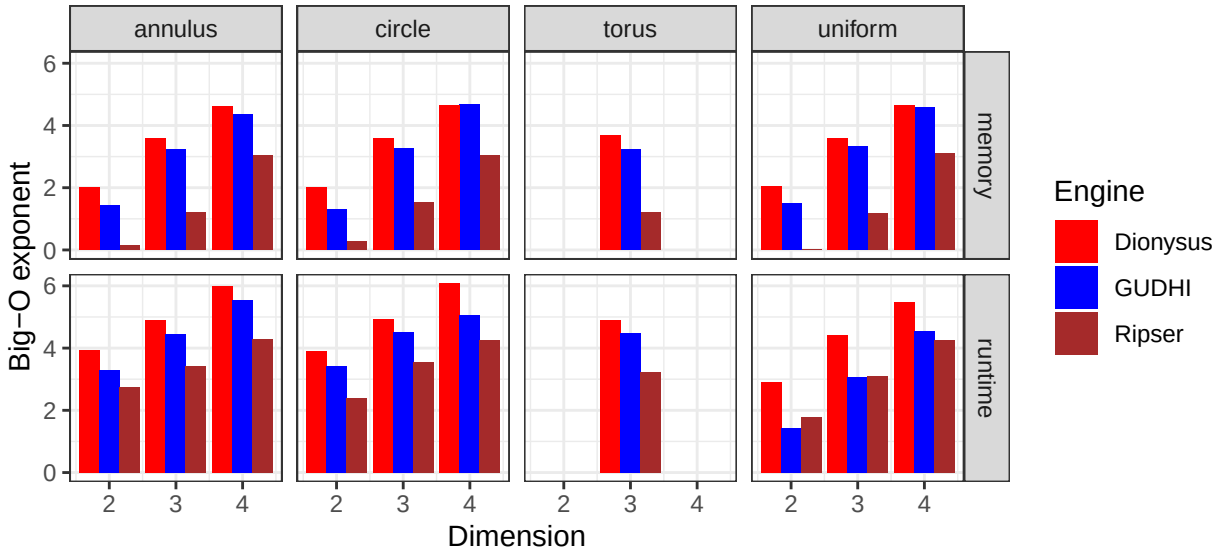


Figure 8: Big-O exponents for runtime and memory complexity as point cloud size varies. In the majority of cases, Ripser (and consequently, TDAstats) has the lowest exponent, indicating the slowest growth in complexity as point cloud size increases. Due to shape constraints, torus only has data available in the third dimension.

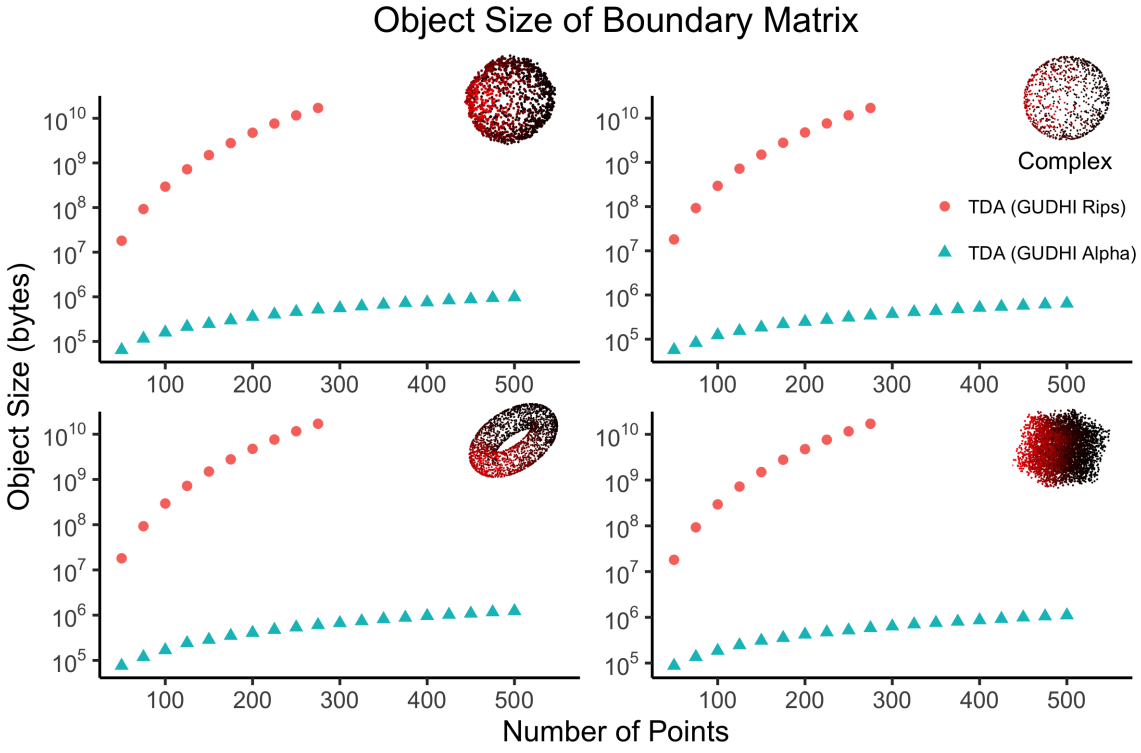


Figure 9: Runtime comparison of persistent homology calculation between Ripser's Vietoris-Rips and GUDHI's alpha complex functionality. Median runtime (min to max $n = 10$ iterations per data point) for various 3-dimensional point cloud structures (facet) plotted against point cloud size for each library (color). Benchmarking was conducted on an annulus (top-left), a sphere (top-right), a torus (bottom-left), and a cube (bottom-right). Data was not collected for data dimensions greater than 3 due to computational limitations of calculating alpha complices.

the sphere.

However, without sufficient computational resources, alpha complexes were not usable for point cloud dimensions greater than 3. If a point cloud has more than 3 dimensions, then it could undergo pre-processing with dimension reduction before using alpha complexes. Note, it is possible an algorithm will eventually be developed to enable alpha complex computation of higher-dimensional data. However, if data dimension cannot be reduced without significant information loss, then Vietoris-Rips complexes should be used. It should be stated that if a point cloud is compatible with both complexes, both analyses should be performed as there may be a variation in the persistent homology matrix. This is because alpha complexes satisfy the Nerve Theorem (Edelsbrunner and Mücke, 1994), which implies that they are topologically equivalent to the true underlying topology of the dataset; in contrast, Vietoris-Rips complexes only approximate the underlying topology (Hausmann, 1996). Among the tested Vietoris-Rips engines, Ripser (wrapped by **TDAstats**) has the fastest calculation time. GUDHI and Dionysus (wrapped by **TDA**) significantly fall behind as feature dimension and number of points increase.

On average, Ripser computed the persistent homology of a Vietoris-Rips complex with less memory than either GUDHI or Dionysus. Thus, when efficiency is critical, useRs would likely find **TDAstats** the appropriate library. However, **TDA** contains an entire library of features not available in **TDAstats**. Specifically, **TDA** allows kNN density estimation, kernel density estimation, density-based clustering, and dendrogram visualization, among other useful functionality. When computational resources are plenty, when point clouds are small and low-dimensional, or when the aforementioned functionality is required, **TDA** will likely be more appropriate than **TDAstats**. Both packages are hosted on CRAN.

While Vietoris-Rips complexes can handle high-dimensional data well, the calculation still significantly slows down when looking for higher dimension features. This is evidenced by the big-O polynomial growth for runtime and memory that have degree less than 4 for most 2-dimensional point clouds, but degree between 4 and 6 for most 4-dimensional point clouds (Figure 8); even higher degree complexities should be expected as point cloud dimension increases. Thus, finding high-dimensional topological features in high-dimensional point clouds remains a challenge. Methods to calculate persistent homology do exist for other simplicial complexes, such as the Delaunay complex and the Witness complex, but, to our knowledge, they are not currently implemented in CRAN packages. Future challenges would be creating and implementing algorithms that reduce the computational complexity of higher-dimensional topological feature calculations for R.

Acknowledgments

The authors thank Emily Dolson, PhD for assistance with using a high-performance computing node for data collection. JGS was supported by NIH grant R37 CA244613.

Bibliography

U. Bauer. Ripser: efficient computation of vietoris-rips persistence barcodes. *arXiv*, math.AT (1908.02518), 2019. [p184, 186]

J. C. Brunson, R. R. Wadhwa, and J. G. Scott. *ggtda: ggplot2-Compatible Visualization of Persistent Homology*, 2020. URL https://github.com/rrrlw/ggtda. R package version 0.1.0. [p186]

J. M. Chan, G. Carlsson, and R. Rabadan. Topology of viral evolution. *Proceedings of the National Academy of Sciences*, 110(46):18566—-18571, 2013. doi: 10.1073/pnas.1313480110. [p184]

H. Edelsbrunner and J. Harer. Persistent homology — a survey. *Discrete & Computational Geometry*, 453, 1 2008. doi: 10.1090/conm/453/08802. [p184]

H. Edelsbrunner and E. P. Mücke. Three-dimensional alpha shapes. *ACM Transactions on Graphics*, 13 (1):43–72, Jan 1994. doi: 10.1145/174462.156635. [p185, 191]

B. T. Fasy, J. Kim, F. Lecci, C. Maria, D. L. Millman, and V. Rouvreau. *TDA: Statistical Tools for Topological Data Analysis*, 2019. URL https://CRAN.R-project.org/package=TDA. R package version 1.6.5. [p184, 186]

J.-C. Hausmann. On the vietoris-rips complexes and a cohomology theory for metric spaces. In F. Quinn, editor, *Prospects in Topology (AM-138): Proceedings of a Conference in Honor of William Browder*, pages 175–188. Princeton University Press, Princeton, NJ, 1996. doi: 10.1515/9781400882588-013. [p184, 191]

J. Hester. *bench: High Precision Timing of R Expressions*, 2019. URL https://CRAN.R-project.org/package=bench. R package version 1.0.4. [p186]

U. Ligges and M. Mächler. Scatterplot3d - an r package for visualizing multivariate data. *Journal of Statistical Software*, 8(11):1–20, 2003. URL http://www.jstatsoft.org. [p186]

C. Maria, P. Dlotko, V. Rouvreau, and M. Glisse. Rips complex. In *GUDHI User and Reference Manual*. GUDHI Editorial Board, 2016. URL http://gudhi.gforge.inria.fr/doc/latest/. [p184, 186]

D. Morozov. *Dionysus*, 2018. URL http://mrzv.org/software/dionysus2. [p184, 186]

M. Offroy and L. Duponchel. Topological data analysis: A promising big data exploration tool in biology, analytical chemistry and physical chemistry. *Analytica Chimica Acta*, 910:1–11, 2016. doi: 10.1016/j.aca.2015.12.037. [p184]

J. Ooms. *magick: Advanced Graphics and Image-Processing in R*, 2019. URL https://CRAN.R-project.org/package=magick. R package version 2.2. [p186]

N. Otter, M. A. Porter, U. Tillmann, P. Grindrod, and H. A. Harrington. A roadmap for the computation of persistent homology. *EPJ Data Science*, 6(1), Sep 2017. doi: 10.1140/epjds/s13688-017-0109-5. [p184, 186, 188]

C. Schmid and B. Serbe. *recexcavAAR: 3D Reconstruction of Archaeological Excavations*, 2017. URL https://CRAN.R-project.org/package=recexcavAAR. R package version 0.3.0. [p186]

R. Turner. *deldir: Delaunay Triangulation and Dirichlet (Voronoi) Tessellation*, 2020. URL https://CRAN.R-project.org/package=deldir. R package version 0.1-25. [p186]

R. R. Wadhwa, D. F. Williamson, A. Dhawan, and J. G. Scott. TDAstats: R pipeline for computing persistent homology in topological data analysis. *Journal of Open Source Software*, 3(28):860, 2018. doi: 10.21105/joss.00860. URL https://doi.org/10.21105/joss.00860. [p184, 186]

H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. ISBN 978-3-319-24277-4. URL https://ggplot2.tidyverse.org. [p186]

H. Wickham. *pryr: Tools for Computing on the Language*, 2018. URL https://CRAN.R-project.org/package=pryr. R package version 0.1.4. [p186]

H. Wickham, J. Hester, and R. Francois. *readr: Read Rectangular Text Data*, 2018. URL https://CRAN.R-project.org/package=readr. R package version 1.3.1. [p186]

Eashwar V. Somasundaram
School of Medicine
Case Western Reserve University
Cleveland, OH 44106, United States

eashwar.somasundaram@case.edu

Shael E. Brown
Department of Quantitative Life Sciences
McGill University
Montreal, QC H3A 0G4, Canada

shael.brown@mail.mcgill.ca

Adam Litzler
Department of Translational Hematology and Oncology Research
Cleveland Clinic Foundation
Cleveland, OH 44195, United States

adli4611@colorado.edu

Jacob G. Scott
Department of Translational Hematology and Oncology Research
Cleveland Clinic Foundation
Cleveland, OH 44195, United States

ScottJ10@ccf.org

Raoul R. Wadhwa
Cleveland Clinic Lerner College of Medicine
Case Western Reserve University
Cleveland, OH 44195, United States

raoul.wadhwa@case.edu

Statistical Quality Control with the qcr Package

by Miguel Flores, Rubén Fernández-Casal, Salvador Naya, Javier Tarrío-Saavedra

Abstract The R package **qcr** for Statistical Quality Control (SQC) is introduced and described. It includes a comprehensive set of univariate and multivariate SQC tools that completes and increases the SQC techniques available in R. Apart from integrating different R packages devoted to SQC (**qcc**, **MSQC**), **qcr** provides nonparametric tools that are highly useful when Gaussian assumption is not met. This package computes standard univariate control charts for individual measurements, $\bar{x}$, S, R, p, np, c, u, EWMA, and CUSUM. In addition, it includes functions to perform multivariate control charts such as Hotelling T^2, MEWMA and MCUSUM. As representative features, multivariate nonparametric alternatives based on data depth are implemented in this package: r, Q and S control charts. The **qcr** library also estimates the most complete set of capability indices from first to the fourth generation, covering the nonparametric alternatives, and performing the corresponding capability analysis graphical outputs, including the process capability plots. Moreover, Phase I and II control charts for functional data are included.

Introduction

Throughout the last decades, there has been an increasing interest to measure, improve, and control the quality of products, services, and procedures. This is connected to the strong relationship between quality, productivity, prestige, trust, and brand image. In fact, implementing procedures of statistical quality control (SQC) is currently related to increasing companies' competitiveness.
The concept of quality control has been extended from the first definitions based on the idea of adjusting production to a standard model to satisfy customer requirements and include all participants. Nowadays, SQC is not only applied to manufactured products but to all industrial and service processes.

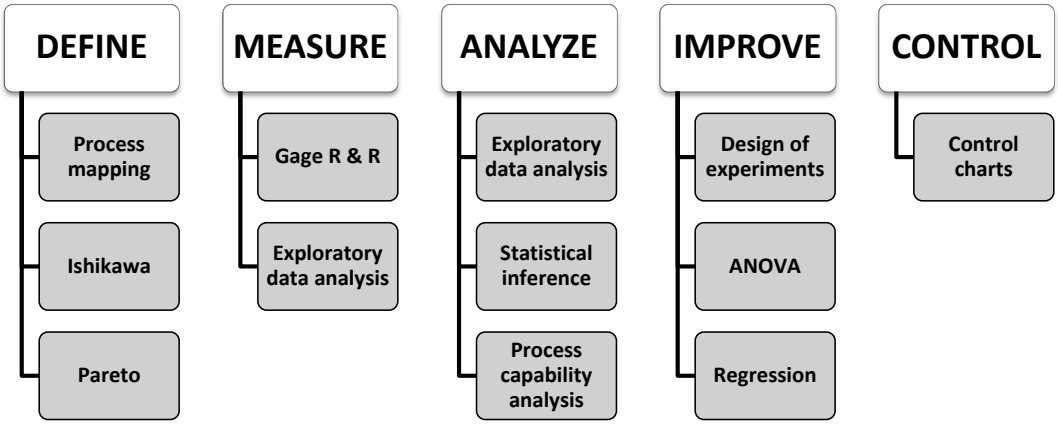


Figure 1: Statistical tools applied in each steps of the Six Sigma methodology.

The use of different SQC techniques was standardized With the development of the Six Sigma method by Motorola in 1997 (Pande et al., 2000). Six Sigma is a methodology or even philosophy focused on variability reduction that promotes the use of statistical methods and tools in order to improve processes in industry and services. The Six Sigma application is composed of five stages: Define, Measure, Analyze, Improve, and Control (DEMAIC). Figure 1 shows some representative statistical techniques applied in each of the Six Sigma stages. The two most representative statistical tools of SQC are the control charts and the process capability analysis (Montgomery, 2009). Therefore, the proposed **qcr** package has been developed in order to provide users a comprehensive and free set of programming tools to apply control charts and perform capability analysis in the SQC framework.

The control stage is characterized by the use of tools based on anomaly detection and correction (Montgomery, 2009). The most representative techniques of this stage and the primary tool of the Statistical Process Control (SPC) are the control charts (Champ and Woodall, 1987). They have been

developed to evaluate the process performance and at any time. The use of control charts prevents the process from getting out of control, and helping to detect the assignable causes corresponding to variations of the critical-to-quality features (CTQs), thus performing process changes when actually required. Furthermore, control charts provide estimates of the natural range of process variation (natural control limits), allowing us to compare this range with those limits specified by standards, company managers, or customers (specification limits). Hence, the process monitoring can be carried out by comparing each new observation with these natural limits, preventing defects in the final product. Briefly, a control chart is a two-dimensional graph whose axis represents the variable or attribute that is being monitored (CTQ variables). The estimation of natural control limits of the CTQ variables is developed by a process composed of two phases: In Phase I, the natural control limits are estimated using a preliminary sample (calibration sample) where we assume that the causes of variation are only random. In Phase II, each new observation is plotted on the control chart along with the natural limits obtained in the previous step. The set of new observations (twhich are not used to calculate the natural control limits) make up the so-called monitoring sample. Patterns, observations of out of control limits, runs of more than six observations on one side of the central line, among others, are some of the different criteria to identify out of control states in a specific process, providing also valuable information about the detection of any assignable causes of variation in the monitoring.

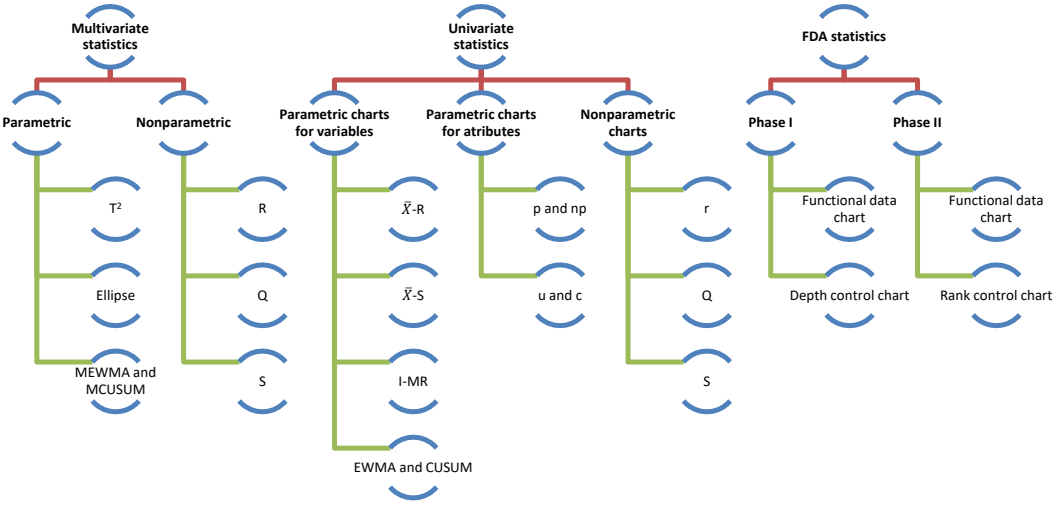


Figure 2: Control charts implemented in the qcr package.

The most used control charts are based on the assumptions of normality and independence of the studied CTQ variables. These charts are used to control the position and dispersion of CTQ attributes and variables. Figure 2 shows some of the most important types of control charts. These can be classified according to the type of feature that is being controlled (attribute or variable), the variable dimension (univariate or multivariate), and assuming or not a parametric distribution of the variable (parametric or nonparametric). The qcr package provides charts for the mean ($\bar{x}$), standard deviation (s), range (R), individual measurements (I), moving ranges (MR), proportion of nonconforming units (p), number of nonconforming units (np), number of defects per unit (c), mean number of defects per control unit (u), exponentially weighted moving average (EWMA), and cumulative sum control chart (CUSUM). The last two techniques are also called memory control charts, and they are specially designed to detect shifts of less than two standard deviations, both when using rational samples or individual measurements. On the other hand, new control charts based on the concept of data depth and developed by Liu (1995) are implemented in qcr. Those are the r, Q, and S control charts, the nonparametric alternatives for individual measurements, mean control chart, and CUSUM control chart, respectively. When more than one variable defines the process quality, multivariate control charts are applied. If the Gaussian assumption is met, the Hotelling T^2 control chart can be applied. If we want to detect small deviations, multivariate EWMA (MEWMA) and multivariate CUSUM (MCUSUM) can be implemented. When no parametric distribution is assumed, r, Q, and S charts can be used.

Another interesting SQC tool, which is very useful in the industry, is the Process Capability Analysis (PCA). It estimates how well a process meets the tolerances defined by the company, customers, standards, etc., by comparing the specification tolerances with respect to the natural range of variation of CTQ features. The capability of the process is measured using capability

indicators. Process Capability Ratio (PCR) is a numerical score that helps the manufacturers know whether the output of a process meets the engineering specifications. Large PCR values show that the industrial or service process is capable of meeting the customer requirements. There have been many different PCRs developed in the last four decades that require the Gaussian assumption for the CTQ variable (Boyles, 1991). However, many processes in industry and real applications do not meet this hypothesis. Thus, we could innacuratelly estimate the capability using PCR. Hence, many authors have studied different nonparametric alternatives to traditional PCR (Polansky, 2007).

The **qcr** package has been developed in R (R Core Team, 2021) under the GNU license. Nowadays, there are other R packages that currently provide quality control tools for users. The use of each one is shown in Figure 3.

The **qcc** package (Scrucca, 2004) was developed by Professor Luca Scrucca of the Department of Economics, Finance, and Statistics at the University of Perugia. It enables us to perform Shewhart quality control charts for variables and attributes, as well as the CUSUM and EWMA charts for detecting small changes in the CTQ variable. Multivariate analysis is performed applying the Hotelling T^2 control chart. Additionally, it has functions implemented to obtain the operating characteristic curves (OC) and to estimate process capability analysis indices. Pareto and Ishikawa diagrams are also implemented. Otherwise, the **IQCC** package (Barros, 2017) is maintained by Professor Emanuel P. Barbosa of the Institute of Mathematics in the State University of Campinas. It has a smaller number of control charts implemented, but it incorporates multivariate graphics. The **qualityTools** package (Roth, 2016) was developed to aid learning in quality sciences. Figure 3 shows some of its utilities, e.g., capability analysis (providing a comprehensive set of parametric distributions) and design of experiments. In addition, the **SixSigma** library (Cano et al., 2012, 2015) provides alternative functions to **qualityTools** and **qcc** packages and the possibility of implementing process maps.

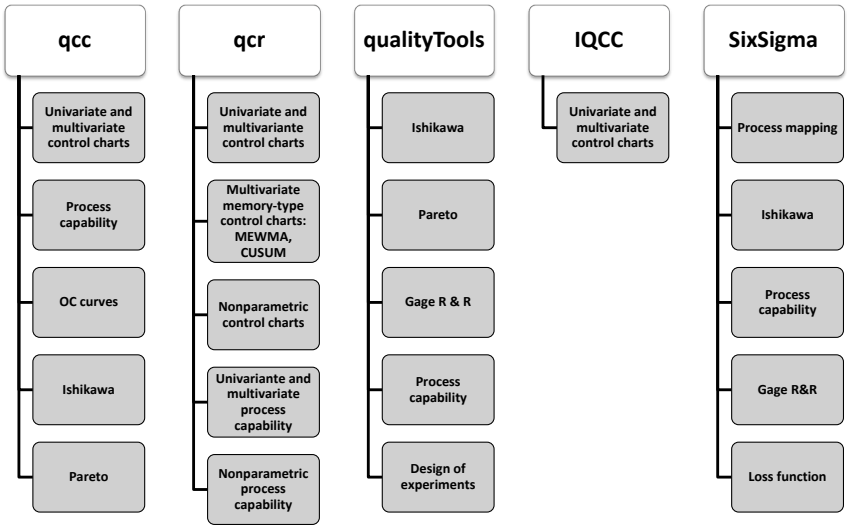


Figure 3: Comparison between the main packages in R devoted to Statistical Quality Control and the **qcr** package.

Furthermore, there are other libraries specifically focused on control chart applications. Namely, the **spcadjust** (Gandy and Kvaloy, 2013) that allows us to estimate the calibration control limits of Shewhart, CUSUM, and EWMA control charts, and the **spc** (Knoth, 2021) which provides tools for the evaluation of EWMA, CUSUM, and Shiryaev-Roberts control charts by using Average Run Length and RL quantiles criteria. Moreover, the **MSQC** package (Santos-Fernandez, 2013) is a set of tools for multivariate process control, mainly control charts. It contains the main alternatives for multivariate control charts such as Hotelling (T^2), Chi-squared, MEWMA, MCUSUM, and Generalized Variance control charts. It also includes some tools to evaluate the multivariate normal assumption. The corresponding multivariate capability analysis can be performed using the **MPCI** library (Santos-Fernández and Scagliarini, 2012) that provides different multivariate capability indices. It is also interesting to mention the **edcc** package (Zhu and Park, 2013) for its economic design of control charts by minimizing the expected cost per hour of the studied process.

It is important to emphasize that the **qcr** package also includes new applications such as nonparametric approaches of control charts and capability indices (also covering the capability plots),

which are currently unavailable in the R software.

Datasets in the qcr package

The **qcr** package contains new databases (see table 1) based on study cases tackled by the authors during their professional activity as well as well-known datasets implemented on other packages focused on statistical quality control such as:

- **archery1:** It consists of a stage in which the archer shoots 72 arrows. The information is given in x and y coordinates. It is implemented in the **MSQC** package (Santos-Fernandez 2013).
- **circuit:** Number of nonconformities observed in 26 successive samples of 100 printed circuit boards. It is implemented in the **qcc** package (Scrucca 2004).
- **dowel1:** Diameter and length of a dowel pin. It is implemented in the **MSQC** package (Santos-Fernandez 2013).
- **orangejuice:** Frozen concentrated orange juice is packed in 6-oz cartons. These cartons are formed on a machine by spinning them from a cardboard stock and attaching a metal bottom panel. A can is then inspected to determine whether, when filled, the liquid could possibly leak either on the side seam or around the bottom joint. If this occurs, a can is considered nonconforming. The data were collected as 30 samples of 50 cans each at half-hour intervals over a three-shift period in which the machine was in continuous operation. It is implemented in the **qcc** package (Scrucca 2004).
- **pcmanufact:** A personal computer manufacturer counts the number of nonconformities per unit on the final assembly line. He collects data on 20 samples of 5 computers each. It is implemented in the **qcc** package (Scrucca 2004).
- **pistonrings:** Piston rings for an automotive engine are produced by a forging process. The inside diameter of the rings manufactured by the process is measured on 25 samples, each of size 5, drawn from a process being considered 'in control'. It is implemented in the **qcc** package (Scrucca 2004).

Univariate and multivariate parametric control charts in qcr

The construction of a control chart is equivalent to the plotting of the acceptance regions of a sequence of hypothesis tests over time. Namely, the $\bar{x}$ chart is a control chart used to monitor the process mean μ. It plots the sample means, $\bar{X}$'s, corresponding to subgroups of the $\{X_1, X_2, ...\}$ observations and is equivalent to test the hypotheses $H_0 : \mu = \mu_0$ versus $H_\alpha : \mu \neq \mu_0$ (for some target value μ_0) conducted over time, using $\bar{x}$ as the test statistic. Here we assume that $\{X_1, X_2, ...\}$ are the sample measurements of a particular CTQ feature that follows the F distribution with mean μ and standard deviation σ. When there is insufficient evidence to reject H_0, we can state that the process is under control; otherwise, the process is out of control. In other words, processes are under control when their sources of variation are only the sources common to the process (Brown and Wetherill, 1990). The decision to reject or not H_0 is based on the value of the sample mean $\bar{x}$ observed at each time interval (Liu and Tang, 1996). The control charts are easy to construct, visualize, and interpret, and most important, have proven their effectiveness in practice since the 1920's.

Control charts are defined, on the one hand, by a center line that represents the average value of the CTQ feature corresponding to the in-control state and, on the other hand, two horizontal lines, called the upper control limit (UCL) and the lower control limit (LCL). The region between the control limits corresponds to the region where H_0 is not rejected (defined in the previous section). As a consequence, the process will be out of control when an observed rational sample or an individual measurement falls outside the limits. Let w be a sample statistic that measures a quality characteristic of interest, and suppose that the mean of w is μ_w and the standard deviation of w is σ_w. Then the center line, the upper control limit, and the lower control limit become:

$$\text{UCL} = \mu_w + L\sigma_w$$

$$\text{CL} = \mu_w$$

$$\text{LCL} = \mu_w - L\sigma_w,$$

Name	Description
counters	A water supply company wants to control the performance of the water counters installed throughout a city. For this purpose, 60 rational samples have been taken, each one composed by 3 measurements, from the same age (10 years) and caliber water counters corresponding to two different brands, and during a period of 5 years. This dataset is based on a study case of A Coruña's water supply company, Empresa Municipal de Aguas de La Coruña (Emalcsa).
employment	A Spaniard-Argentinian hotel company wants to control the level of occupancy (measured in %) in their establishments through the application of a continuous control. For this purpose, 48 subsamples have been taken from six hotels corresponding to two different countries.
oxidation	This database contains information about the resistance against the oxidation of olive oil of the Picual variety. Five measurements of the Onset Oxidation Temperature (OOT, index that measures the resistance against the oxidation) are obtained from 50 batches of Picual olive oil produced in chronological order. It is importantly to note that OOT decreases as the oil is progressively mixed with other olive oil varieties defined by a lower OOT.
plates	A chemical company is developing a patent for a new variant of artificial stone mostly made of quartz (93wt% and polyester resin). This company is launching a pilot plant where it begins to produce plates of this material on an industrial scale. The CTQ variable of this product is the Vickers hardness. In order to measure the hardness level and hardness homogeneity of the product, 50 plates have been measured 5 times in different sections. The characteristic learning curves, through gradual level change, can be observed.
presion	A shipyard of recreational boats is intended to optimize and control the mechanical properties of the yacht hulls made of a composite based on epoxy resin. In this regard, the modulus of elasticity due to tensile efforts is measured after applying two different curing pressures: 0.1 and 10 MPa. Overall, 60 subsamples, composed of three measurements, obtained from 60 vessels, have been taken.

Table 1: Some of the specific datasets included in the **qcr** package

where L is the "distance" of the control limits from the center line, expressed in standard deviation units.

When several random variables characterize the quality of a process/service, applying statistical multivariate quality control techniques becomes necessary. In fact, if we analyze each variable separately, the probability that an observation of a variable will fall within the calculated limits when it is known that the process is actually under control will no longer be 0.9973 for 6σ amplitude. Assuming independence, it will be 0.9973^p, where p is the number of CTQ features, while the probability of type I will actually lead to $\alpha' = 1 - (1 - \alpha)^p$. Therefore, the control limits are different from those drawn, assuming the control of each CTQ variable independently from the others. Moreover, if the variables are dependent, the calculation of α becomes more complex. This subject is particularly important today, as automatic inspection procedures make it customary to measure many parameters of each product over time. The more common multivariate parametric control charts are the Hotelling T^2 (to identify big shifts) and the multivariate CUSUM (MCUSUM) and EWMA (MEWMA) for identifying small shifts.

The functions that compute the quality control statistics for the different univariate control charts (involving continuous, attribute or count data) are shown in Table 2. For the sake of simplicity and taking into account that these types of control charts are implemented in other packages, the use of these functions is not shown in this work. More details are given in the help of **qcr** package (Flores et al., 2021).

Function	Chart name	Statistical quality control charts for Variables
`qcs.xbar`	$\bar{X}$	Sample means of a continuous process variable are plotted to control the process average.
`qcs.R`	R	Sample ranges of a continuous process variable are plotted to control the process variability.
`qcs.S`	S	Sample standard deviations of a continuous variable are plotted to control the process variability.
`qcs.one`	I	Sample values from a I chart data of a continuous process variable to control the level (position) of the process.
		Attributes
`qcs.p`	p	Proportion of nonconforming units is plotted, the number of defective items follow a binomial distribution.
`qcs.np`	np	Number of nonconforming units is plotted, and the chart is constructed based on the average of the process.
`qcs.c`	c	Nonconformities per unit are plotted, number of defects in a large population follow a Poisson distribution.
`qcs.u`	u	Average nonconformities per unit are plotted, this chart does not require a constant number of units.
`qcs.g`	g	Number of non-events between events are plotted, it counts the number of events between rarely-occurring errors or nonconforming incidents.
		Attributes and variables
`qcs.cusum`	CUSUM	Cumulative sums for individual observations or for the averages of rational subgroups are plotted to monitor the process mean.
`qcs.ewma`	EWMA	The exponential weighed average of CTQ variables are plotted to identify small changes in the process (measured as rational samples or individual observations).
		Multivariate control charts
`mqcs.t2`	T^2	Multivariante Hotelling T^2 control chart for individual observations (vectors).
`mqcs.mcusum`	MCUSUM	Multivariate Cumulative Sum control chart for individual observations (vectors).
`mqcs.ewma`	MEWMA	Multivariate EWMA control chart for individual observations (vectors).
		Functional data control charts
`fdqcs.depth`	Phase I	Phase I control chart for functional data: depth control chart and deepest curves envelope.
`fdqcs.rank`	Phase II	Phase II control chart for functional data: rank control chart and deepest curves envelope.
`plot.fdqcs`	FDA plots	Graphical outputs for Phase I and Phase II control charts for functional data.

Table 2: Univariate Shewhart, multivariate Hotelling T^2, univariate and multivariate CUSUM and EWMA and FDA control charts available in the **qcr** package

Nonparametric control charts based on data depth

The control charts presented in this section were proposed by Liu (1995) as an alternative to those described in previous section. The main idea of its control graphs is to reduce each multivariate measure to the univariate index, that is, its relative center-exterior classification induced by a depth of data. This approach is completely nonparametric, and therefore, these control charts are not defined by any parametric assumption regarding the process model. Thus, they are applicable in a wider number of case studies than those counterparts such as T^2, MCUSUM, and MEWMA control charts. In addition, these graphs allow the simultaneous detection of the change of location (shift of the mean) and the increase of the scale (change in variability) in a process.

Liu (1995) proposed and justified three types of control charts, the r, Q, and S charts which can be considered as data-depth-based multivariate generalizations of the univariate X, $\bar{x}$, and CUSUM charts, respectively.

Data depth

In multivariate analysis, the term depth refers to the degree of centrality of a point regarding a data cloud or a probability distribution. Therefore, it is possible to define a rank in the multidimensional Euclidean space through the calculation of observation depth. According to Dyckerhoff (2004) and Cascos et al. (2011), the depth function can be defined as a bounded function $D_p : R^d \longrightarrow R$, with P the distribution set in R^d, that assigns at each point of R^d its degree of centrality with respect to P. Depth functions with which control charts can be performed are the

- Simplicial depth (Liu 1990),
- Mahalanobis depth (Mahalanobis 1936),
- Halfspace or Tukey depth (Tukey 1975),
- Likelihood depth (Fraiman et al. 1997), and
- Random projection depth (Zuo and Serfling 2000).

Statistics derived from data depth

Let G a k-dimensional distribution, and let $Y_1, \ldots, Y_m$ be m random observations from G. The sample $Y_1, \ldots, Y_m$ is generally the reference sample of a CTQ variable in the context of quality control, composed of measurements from products obtained by an under control process. If $X_1, X_2, \ldots$ are the new observations from the manufacturing process, assuming that the different X_i values follow an F distribution if the quality of the studied product has been deteriorated or, in other words, if the process is out of control. Otherwise, they follow a G distribution. Let $D_G(\cdot)$ denote a notion of depth, and assume that G and F are two continuous distributions. Thus, if all the $D_G(Y_i)$ values are sorted in increasing order, and $Y_{[j]}$ denotes the sample value associated with the jth smallest depth value, then $Y_{[1]}, \ldots, Y_{[m]}$ are the order statistics of Y_i's, with $Y_{[m]}$ being the most central point. Therefore, the smaller the order (or the rank) of a point, the farther that point will be from the underlying distribution $G(\cdot)$.

Liu (1995) defines the rank statistic as

$$r_G(y) = P\{D_G(Y) \leq D_G(y) \mid Y \sim G\}$$

whereby $Y \sim G$ indicates that the random variable Y follows the distribution G. When G is unknown, the empirical distribution G_m of the sample $\{Y_1, \ldots, Y_m\}$ can be used instead, and the statistic is defined by

$$r_{G_m}(y) = \frac{\#\left\{D_{G_m}(Y_j) \leq D_{G_m}(y), j = 1, \ldots, m\right\}}{m}$$

In the same way that r_G and r_{G_m}, the Q statistics can be also defined as follows

$$Q(G, F) = P\{D_G(Y) \leq D_G(X) \mid Y \sim G, X \sim F\} = E_F[r_G(X)]$$

$$Q(G, F_n) = \frac{1}{n}\sum_{i=1}^{n} r_G(X_i)$$

$$Q(G_m, F_n) = \frac{1}{n}\sum_{i=1}^{n} r_{G_m}(X_i),$$

whereby $F_n(\cdot)$ denotes the empirical distribution of the sample $\{X_1, \ldots, X_n\}$. The control charts corresponding to these statistics can be developed as described in the following sections.

The r chart

Calculate $\{r_G(X_1), r_G(X_2), \ldots, r_G(X_n)\}$ or $\{r_{G_m}(X_1), r_{G_m}(X_2), \ldots, r_{G_m}(X_n)\}$ if G is unknown but $Y_1, \ldots, Y_m$ are available. As a result, the r chart consists of plotting the rank statistic in

regard to time. The control chart central line is CL = 0.5, whereas the lower limit is LCL =α, with α accounting for the false alarm rate. The process will be out of control if $r_G(\cdot)$ falls under LCL. A small value of the rank statistic $r_{Gm}(X)$ means that only a very small proportion of Y_i values are more outlying than X. Therefore, assuming that $X \sim F$, then a small value of $r_{Gm}(X)$ suggests a possible deviation from G to F. This may be due to a shifting in the location and/or an increase in the scale of the studied CTQ variable. Taking into account that the UCL is not defined for the r chart, the CL line serves as a reference to identify emerging patterns, runs, or trends. If $r_{Gm}(X)$ is greater than 0.5, there is evidence of scale decreasing, and also could take place a negligible location shift. This case should be tackled as an improvement in quality given a gain in the accuracy, and thus the process should not be considered as out of control.

The Q chart

The idea behind the Q chart is similar to the one behind the $\bar{x}$ chart. If $X_1, X_2, \ldots$ are univariate and G is a normal distribution, the $\bar{x}$ chart plots the averages of consecutive subsets of the different X_i. A goal of this type of chart is that it can prevent the identification of a false alarm when the process is actually in control (even when some individual sample points fall out of control limits due to random fluctuations).
The Q chart is the nonparametric alternative to the $\bar{x}$ chart. It is performed by plotting the averages of consecutive subsets of size n corresponding to the rank statistic ($r_G(X_i)$ or $r_{Gm}(X_i)$), given by $Q\left(G, F_n^j\right)$ or $Q\left(G_m, F_n^j\right)$, whereas F_n^j is the empirical distribution of the X_i's in the jth subset, $j = 1, 2, \ldots$. Accordingly, if only $\{Y_1, Y_2, \ldots, Y_m\}$ are available, the Q chart plots the sequence $\left\{Q\left(G_m, F_n^j\right), Q\left(G_m, F_n^j\right), \ldots\right\}$.
Depending on the value of n, the corresponding control limits are as follows:

- If $n \geq 5$, $CL = 0.5$ and
  - $LCL = 0.5 - Z_\alpha \left(12n\right)^{\frac{1}{2}}$ for $Q\left(G, F_n^j\right)$.
  - $LCL = 0.5 - Z_\alpha \sqrt{\frac{1}{12}\left(\frac{1}{m} + \frac{1}{n}\right)}$ for $Q\left(G_m, F_n^j\right)$.
- If $n < 5$, $CL = 0.5$ and $LCL = \frac{(n!\alpha)^{\frac{1}{n}}}{n}$.

The S control chart

The S control chart is based on the CUSUM univariate control chart, which is basically the plot of $\sum_{i=1}^{n}(X - \mu)$, which reflects the pattern of the total deviation from the expected value. As mentioned above, it is more effective than the X chart or the $\bar{x}$ chart in detecting small process changes. The nonparametric CUSUM chart based on data depth suggests plotting $S_n(G)$ and $S_n(G_m)$, defined by

$$S_n\left(G\right) = \sum_{i=1}^{n}\left(r_G\left(X_i\right) - \frac{1}{2}\right)$$

with control limits $CL = 0$ and $LCL = -Z_\alpha\left(\frac{n}{12}\right)^{\frac{1}{2}}$ and

$$S_n\left(G_m\right) = \sum_{i=1}^{n}\left(r_{G_m}\left(X_i\right) - \frac{1}{2}\right).$$

If only $Y_1, \ldots Y_m$ are available, the control limits are $CL = 0$ and $LCL = -Z_\alpha\sqrt{n^2\frac{\left(\frac{1}{m}+\frac{1}{n}\right)}{12}}$. The LCL control limits in both cases constitute a curve instead of a straight line; if n is large, the control chart S should be standardized as follows:

$$S_n^*\left(G\right) = \frac{S_n^*\left(G\right)}{\sqrt{\frac{n}{12}}}$$

$$S_n^*\left(G_m\right) = \frac{S_n\left(G_m\right)}{\sqrt{n^2\frac{\left(\frac{1}{m}+\frac{1}{n}\right)}{12}}}$$

Therefore, this S^* chart is defined by $CL = 0$ and $LCL = -Z_\alpha$.

Examples of r, Q and S control charts applied using synthetic data

A bivariate data set is used to illustrate how the previously discussed control charts arise. In fact, a synthetic dataset composed of 540 observations of a bidimensional standard Gaussian variable has been simulated, in addition to 40 individuals corresponding to another bidimensional Gaussian variable with mean and standard deviation equal to 2.

```
R> mu <- c(0, 0)
R> Sigma <- matrix(c(1, 0, 0, 1), nrow = 2)
R> Y <- rmvnorm(540, mean = mu, sigma = Sigma)
R> u <- c(2, 2)
R> S <- matrix(c(4, 0, 0, 4), nrow = 2)
R> x <- rmvnorm(40, mean = u, sigma = S)
```

Prior to the application of nonparametric control charts, the dataset has to be converted into a `npqcsd` object. The synthetic dataset is arranged as two matrices, `G` composed of the 500 first rows (multivariate observations) of `Y`, and `x` with the remaining ones and including those belonging to the second bidimensional variable

```
R> x <- rbind(Y[501:540, ], x)
R> G <- Y[1:500, ]
R> data.npqcd <- npqcd(x, G)
```

In the same way, the `npqcd` function creates a data object for non parametric quality control, the `npqcs.r()`, `npqcs.Q()`, and `npqcs.S()` functions computes all the statistics required to obtain the r, Q, and S control charts, respectively. The argument `method = c("Tukey","Liu","Mahalanobis","RP","LD")` specifies the data depth function, and `alpha` is the signification level that defines the LCL. See Flores et al. (2021) to obtain additional information about these functions and their arguments.

r chart

The r control chart can be obtained by applying the `npqcs.r()` function to the `npqcd` object and plotting the result.

```
R> res.npqcs <- npqcs.r(data.npqcd, method = "Tukey", alpha = 0.025)
R> plot(res.npqcs, title = " r Control Chart")
```

The resulting chart is shown in Figure 4, where it can be observed that the process is out of control from the 42nd observation, as expected, taking into account that most of the $r_{Gm}(X_i)$ values are falling below the LCL.

Q chart

In this case, the dataset is assumed to be composed of rational samples of size 4. Thus, the Q nonparametric alternative of $\bar{x}$ chart is proposed and applied to control the bidimensional process:

```
R> n <- 4    # samples
R> m <- 20   # measurements
R> k <- 2    # number of variables
R> x.a <- array( , dim = c(n, k, m))
R> for (i in 1:m) {
+      x.a[, , i] <- x[(1 + (i - 1) * n):(i * n), ]
+  }
R> data.npqcd <- npqcd(x.a, G)
R> res.npqcs <- npqcs.Q(data.npqcd, method = "Tukey", alpha = 0.025)
R> plot(res.npqcs, title = "Q Control Chart")
```

Figure 5 clearly shows that the process is out of control in the second half, from the 20th rational sample. We can also see that the high random fluctuations of the r chart are attenuated in the Q chart due to the averaging effect.

S chart

Finally, the nonparametric counterpart of CUSUM control chart is performed from the multivariate individual observations.

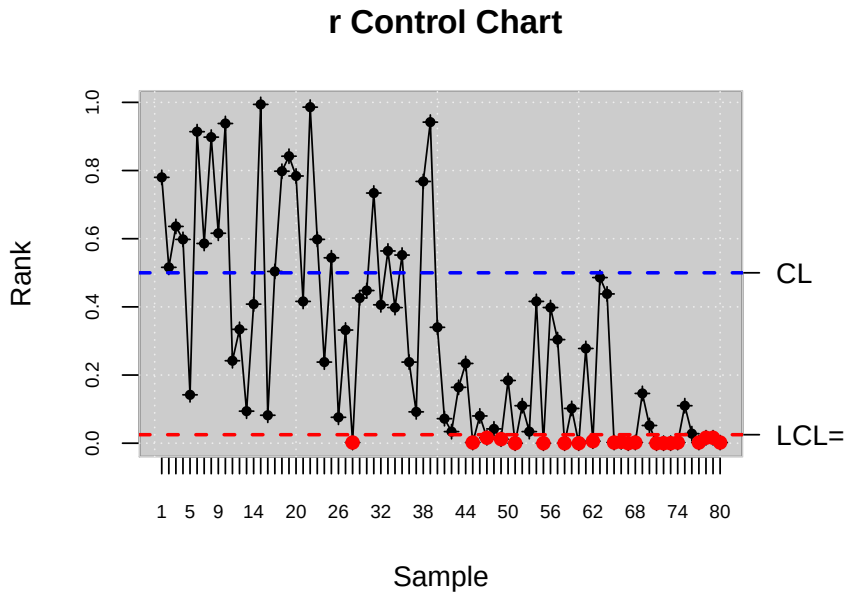


Figure 4: r control chart.

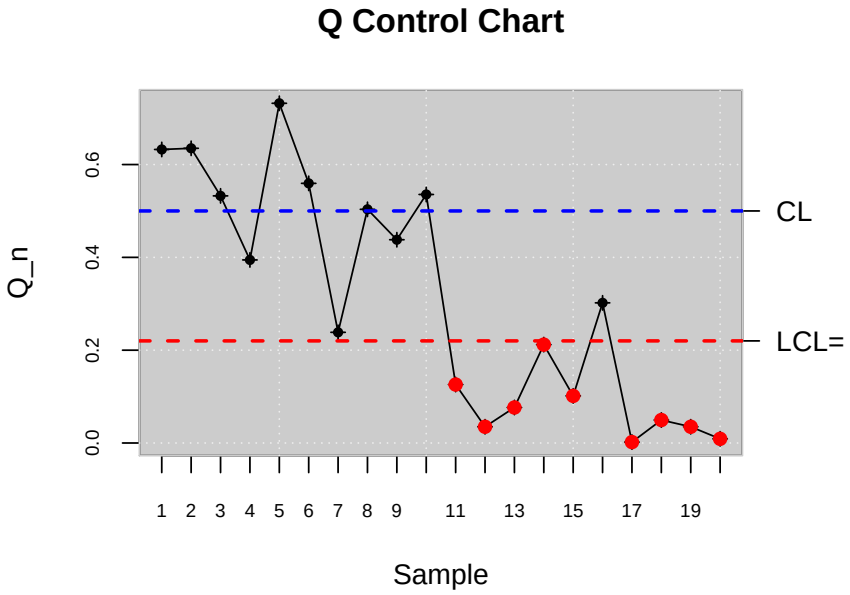


Figure 5: Q control chart.

```
R> data.npqcd <- npqcd(x, G)
R> res.npqcs <- npqcs.S(data.npqcd, method = "Tukey", alpha = 0.05)
R> plot(res.npqcs, title = "S Control Chart")
```

Figure 6 shows that the process is out of control from the 48th observation. Note that the S graph performs better in identifying small changes in a process. In this case, the performance of the Q chart is better than the corresponding to the S chart.

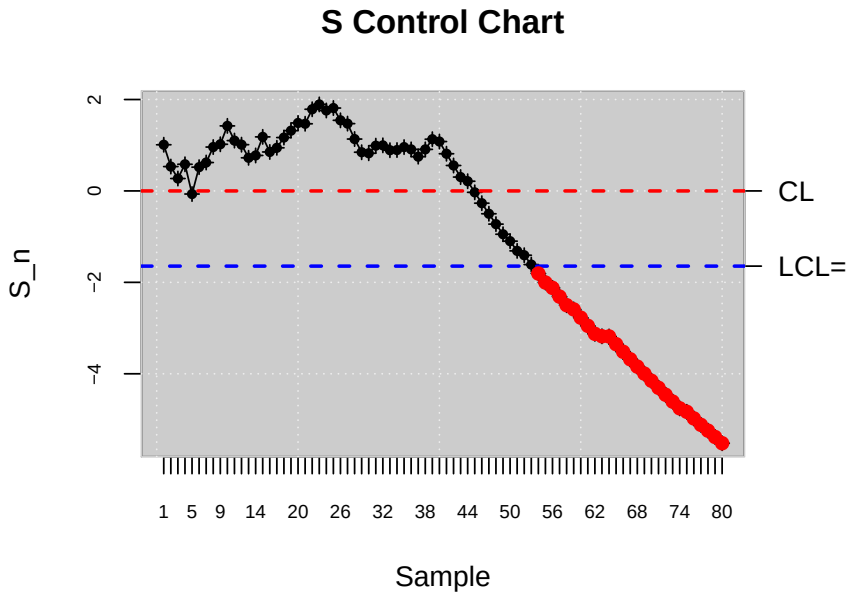


Figure 6: S control chart.

Control charts for functional data based on data depth

In the paradigm of Industry 4.0, processes and services are many times described by continuously monitored data of hourly, daily, monthly curves or smooth functions. When the processes are defined by functional data, the authors encourage to apply control charts based on Functional Data Analysis (FDA) in order to implement control and improvement tasks. In this section, the use of control charts presented in Flores et al. (Flores et al., 2020). Summarizing, this methodology consists of the proposal of new Phase I and Phase II control charts to be applied in those case study in which the datum unit is a curve. Phase I control chart is based on the computation of functional data depth (specifically Fraiman and Muniz (Fraiman and Muniz, 2001), Mode (Cuevas et al., 2007), and random projections (Cuevas et al., 2007) data depth) from which a data depth control chart is developed. Once the in-control calibration sample is obtained, the Phase II control chart based on functional data depth and rank nonparametric control chart can be applied. In addition to the Phase I functional data depth and Phase II rank control charts, plots of functional envelopes from the original curves are provided in order to help to identify the possible assignable causes of out of control states.

Estimating a Phase I control chart for functional data (calibration)

A dataset is simulated in order to illustrate the use of FDA control charts for Phase I and II. A functional mean, mu_0, and a functional standard deviation, `sigma`, are defined as shown in (Flores et al., 2020). An $n_0 = 100$ hundred curves composed of $m = 30$ points are simulated. They account for the calibration or retrospective sample.

```
R> library(fda.usc)
R> m <- 30
R> tt<-seq(0,1,len=m)
# H0
R> mu_0<-30 * tt * (1 - tt)^(3/2)
R> n0 <- 100
R> mdata<-matrix(NA,ncol=m,nrow=n0)
R> sigma <- exp(-3*as.matrix(dist(tt))/0.9)
R> for (i in 1:n0) mdata[i,]<- mu_0+0.5*mvrnorm(mu = mu_0,Sigma = sigma )
```

Prior to the application of control charts, the dataset is converted in a specific format by the `fdqcd` function. A plot function is also programmed to properly show the original functional data, `plot.fdqcd`. Figure 7 shows the original functional data consisting of curves.

```
R> fdchart <- fdqcd(mdata)
R> plot(fdchart,type="l",col="gray",main="Functional data")
```

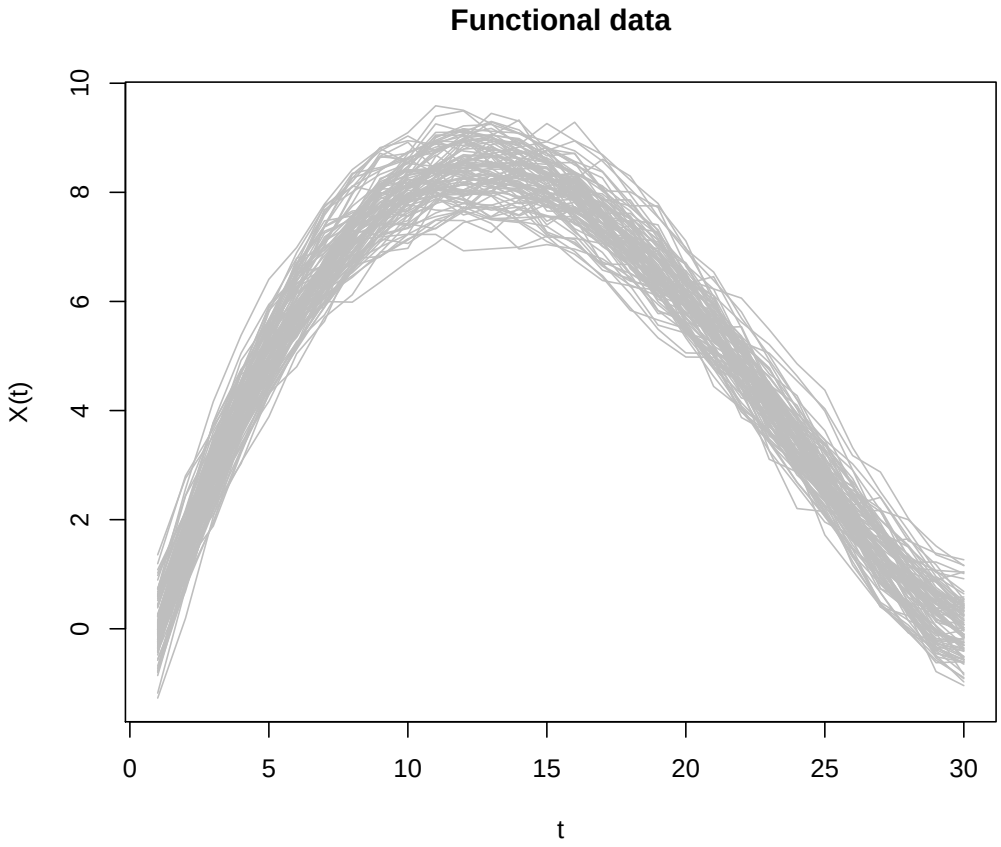


Figure 7: Original curves that account for the calibration sample.

The following step is to identify those curves that account for the in-control process. This task is done by the application of a Phase I control chart for functional data. This method is implemented in the **qcr** package by the `fdqcs.depth` function. Specifically, the arguments and default values for these functions are

```
R> fdqcs.depth.default <- function(x, data.name=NULL,func.depth = depth.mode,nb=200,
+                                  type = c("trim","pond"),ns =  0.01,
+                                  plot = TRUE, trim = 0.025, smo =0.05,
+                                  draw.control = NULL,...)
```

where `func.depth` is the type of depth measure, by default `depth.mode`, `nb` the number of bootstrap resamples, `type` accounts for the method used to trim the data, `trim` or `pond` (Flores et al., 2020), `ns` is the quantile to determine the cutoff from the bootstrap procedure (Flores et al., 2020), `plot` a logical value indicating that it should be plotted, `trim` the percentage of the trimming, `smo` the smoothing parameter for the bootstrap resampling (Flores et al., 2020), whereas `draw.control` specifies the `col`, `lty`, and `lwd` for the fdataobj, statistic, IN and OUT objects. When the `fdqcs.depth` function is applied to the curves of the calibration sample, the `fddep` object of `fdqcs.depth` class is obtained. It is composed of the original data, the depth corresponding to each curve, the lower control limit of the depth chart, the index of those curves out of control, the curves that account for the limits of the envelope composed by the deepest cures, and the deepest curve or functional median.

```
R> fddep <- fdqcs.depth(fdchart)
R> summary(fddep)

      Length Class  Mode
fdata 100    fdata  list
```

```
Depth 100    -none- numeric
LCL     1    -none- numeric
out     1    -none- numeric
fmin    1    fdata  list
fmax    1    fdata  list
fmed    1    fdata  list
ns      1    -none- numeric

R> class(fddep)

[1] "fdqcs.depth"

R> plot(fddep,title.fdata = "FDA chart",title.depth = "Depth chart")
R> out <- fddep$out; out

[1] 29
```

Figure 8 shows the control chart for the depth of the curves (right panel). The LCL is estimated by a smoothed bootstrap procedure (Flores et al., 2020). In order to provide a tool to identify the assignable cause of each out-of-control curve, the original curves with the envelope with the 99% of the deepest curves are also shown (left panel). The analysis of the shape and magnitude of the curves in and out of bounds can help to associate each curve out of control to an assignable cause, allowing for processes control, maintenance, and improvement.

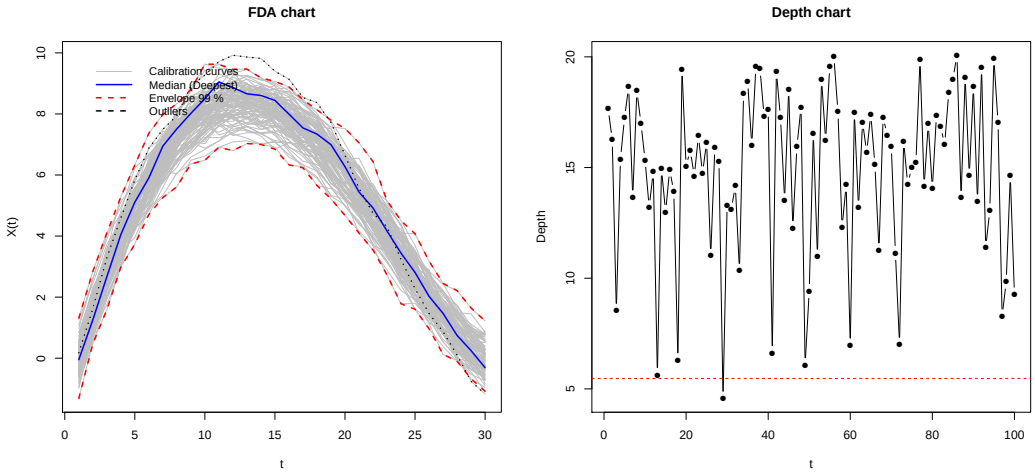


Figure 8: Left panel: Original curves with the envelope composed of 90% of the deepest curves. Right panel: Control chart for the depths of the curves (the LCL has been estimated by bootstrap procedures at a signification level of 10%.

Phase I ends when a calibration sample without curves out of control is obtained. The iterative procedure to obtain an in control calibration sample is shown in the following lines.

```
R> alpha <- 0.1
R> trim <- 0.1
R> while (length(out)>0) {
R>   mdata <- fddep$fdata$data[-out,]
R>   fddep <- fdqcs.depth(mdata,ns = alpha, trim=trim, plot=FALSE)
R>   out <- fddep$out
R> }
R> plot(fddep,title.fdata = "Envelope with the 90\% deepest curves",
+  title.depth = "Depth control chart")
```

Figure 9 is obtained from the application of `plot` function to `fddep` object. It shows that all the curves of the calibration sample are in control, and thus, the natural variability of the process is estimated.

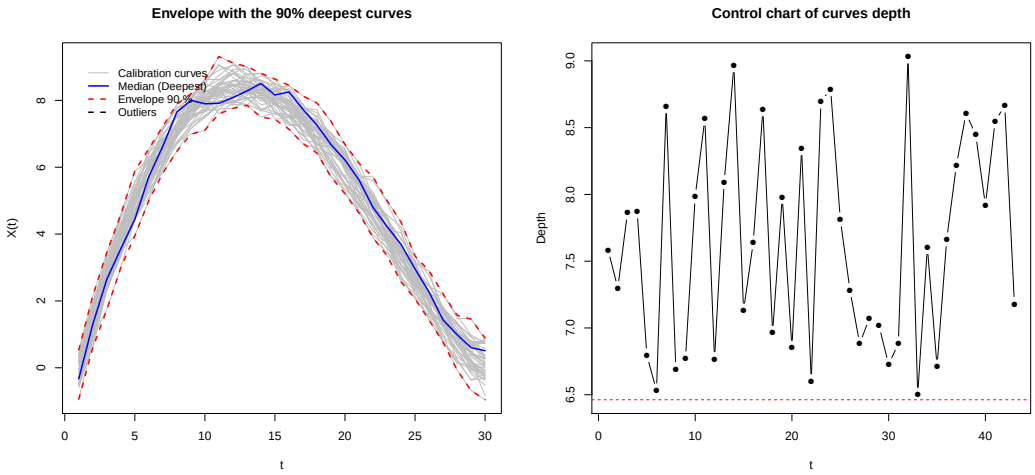


Figure 9: Results corresponding to the second iteration to obtain the in control calibration sample. Left panel: Original curves with the envelope composed of the 90% of the deepest curves. Right panel: Control chart for the depths of the curves (the LCL has been estimated by bootstrap procedures at a signification level of 10%.

Estimating a Phase II control chart for functional data (monitoring)

The next step is to perform the Phase II of process control. The monitoring phase is performed by the application of Phase II control charts for functional data based on multivariate nonparametric control charts. Firstly, a monitoring sample composed of 50 curves is simulated by the following code.

```
R> mu_a<- 30 * tt^(3/2) * (1 - tt)
R> n_a <- 50
R> mdata_a<-matrix(NA,ncol=m,nrow=n_a)
R> for (i in 1:n_a) mdata_a[i,]<- mu_a+0.5*mvrnorm(mu = mu_a,Sigma = sigma )
```

The curves of the monitoring sample are defined with `fdqcd` format and a control chart for Phase II is developed by applying the `fdqcs.rank` function. It is composed by the following arguments, `fdqcs.rank(x,y = x,func.depth = depth.FM,alpha = 0.01,plot = TRUE,trim = 0.1,draw.control = NULL,...)`.

```
R> fdchart_a <- fdqcd(mdata_a,"Monitoring curves")
R> phase2.chart <- fdqcs.rank(fdchart,fdchart_a)
R> plot(phase2.chart)
R> summary(phase2.chart)
```

Figure 10 accounts for the FDA chart with the calibration sample and its envelope composed by the deepest curves. Moreover, the monitoring sample is also included and compared with the calibration sample by using the FDA chart. In addition, the Phase II rank control chart for functional data is shown including both calibration and monitoring samples or only the ranks corresponding to the monitoring sample. The second population that corresponds with the monitoring sample is identified by the control chart from the first monitored curve (panels below in Figure 10).

Process capability analysis

The analysis of the capability of a process in the case of statistical quality control is done through the calculation of the so-called capability. These indices measure whether a process is capable or not of meeting the corresponding technical specifications set by the customer, or the manufacturer, by comparing those with the natural variability of the CTQ variable that characterizes the process. The interpretation of these indices is associated with the result of this relation. Capability indices are generally calculated as the ratio between the length of the specification interval and the natural variability of the process in terms of σ. Large values of these indices mean that the corresponding process is capable of producing articles that meet the requirements of the client and manufacturers.

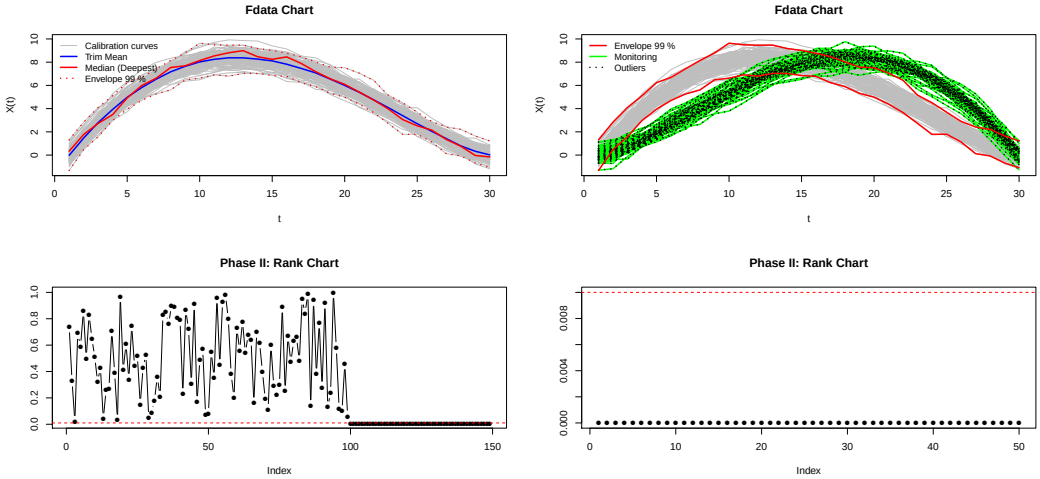


Figure 10: First row: the left and right panels show the FDA charts for the calibrating and monitoring samples with a signification level of 0.01. Second row: The left and right panels show the Phase II rank control chart for functional data, including calibration and monitoring sample (left panel) and only monitoring sample (right panel).

In other words, the larger the value of the capability index, the smaller the number of products outside the specification limits.

In this section, we describe the capability indices for processes whose underlying distribution is normal and not normal (exponential, Weibull, etc.). However, it is important to note that the development of programming tools for nonparametric capability analysis is one of the main goals and contributions of the **qcr** package. In addition to the estimation of capability indices, a graphical output is provided. Based on the proposal of **qualityTools** package, the **qcr** graphical output for capability analysis includes a normality test for the CTQ variable, a Q-Q plot, a histogram with the theoretical Gaussian distribution density, parametric and nonparametric estimates of capability indices and a contour capability control chart. In the following lines, parametric and nonparametric capability analysis utilities are described using different examples of applications.

Assuming a normal distribution

The most widely used capability indexes in the industry analyze the process capability under the assumptions of a stabilized process (in control) and a Gaussian distributed CTQ variable. Table 3 shows the main parametric (assuming Gaussian distribution) indices, namely C_p, C_{pk}, C_{pm}, and C_{pmk}.

Vännman (1995) proposed a general formulation of these indices by an expression that depends on the non-negative parameters u and v:

$$C_p(u,v) = \frac{d - u|\mu - m|}{3\sqrt{\sigma^2 + v(\mu - T)^2}},$$

whereby $d = (USL - LSL)/2$, $m = (LSL + USL)/2$, USL is the upper specification limit, the LSL is the lower specification limit, σ is the theoretical standard deviation, μ accounts for the theoretical mean of the CTQ variable, and T is the specification target (by default the mean between the LSL and USL). The indices shown in Table 3 are obtained from this expression just considering values of 0 or 1 for u and v: $C_p(0,0) = C_p$, $C_p(1,0) = C_{pk}$, $C_p(0,1) = C_{pm}$, $C_p(1,1) = C_{pmk}$.

The piston rings data set is used to illustrate the calculation of the capability indices using the `qcs.cp()` function based on the expressions previously described in Table 3. From the statistics obtained from the $\bar{x}$ control chart of pistonrings dataset, the γ and β values are estimated, and the corresponding capability index is computed.

```
R> data("pistonrings")
R> xbar <- qcs.xbar(pistonrings[1:125, ], plot = FALSE)
R> limits <- c(lsl = 73.99, usl = 74.01)
```

Potential capability	$\hat{C}_p = \frac{USL-LSL}{6\hat{\sigma}}$
Actual capability with respect to the specification limits	$\hat{C}_{p,lower} = \frac{\hat{\mu}-LSL}{3\hat{\sigma}}$ $\hat{C}_{p,upper} = \frac{USL-\hat{\mu}}{3\hat{\sigma}}$ $\hat{C}_{pk} = \min\left[\frac{USL-\hat{\mu}}{3\hat{\sigma}}, \frac{\hat{\mu}-LSL}{3\hat{\sigma}}\right]$
Shifting of the mean with respect to the target	$\hat{C}_{pm} = \frac{\hat{C}_p}{\sqrt{1+\left(\frac{\hat{\mu}-T}{\hat{\sigma}}\right)^2}}$
C_{pk} correction for detecting deviations with respect to the target	$\hat{C}_{pkm} = \frac{\hat{C}_{pk}}{\sqrt{1+\left(\frac{\hat{\mu}-T}{\hat{\sigma}}\right)^2}}$

Table 3: PCR from first to fourth generation, USL is the upper specification limit, LSL is the lower specification limit, μ is the real mean, $\hat{\mu}$ is the estimated mean, and $\hat{\sigma}$ is the estimated standard deviation.

```
R> # qcs.cp(object = xbar, parameters = c(0, 0), limits = limits,
R> #        contour = FALSE)
R> # qcs.cp(object = xbar, parameters = c(1, 0), limits = limits,
R> #        contour = FALSE)
R> # qcs.cp(object = xbar, parameters = c(0, 1), limits = limits,
R> #        contour = FALSE)
R> qcs.cp(object = xbar, parameters = c(1, 1), limits = limits,
+         contour = FALSE)

     Cpmk delta.usl gamma.usl
   0.2984    0.1176    0.9785
```

Consequently, the obtained results are $C_p = 0.3407$, $C_{pk} = 0.3006$, $C_{pm} = 0.3382$, and $C_{pmk} = 0.2984$, respectively. The argument `parameters` account for u and v values, while `object` is the type of control chart from which the σ is estimated, `limits` are the specification control limits, and `contour` is the parameter that indicates when the process capability contour chart is plotted.

Process capability plot

In Vännman (2001) and Deleryd and Vännman (1999), a graphical method (based on common capability indices) to analyze the capability of a process is proposed. The goal of using this type of plot (if compared with respect to only capability indices calculation) is to provide immediate information of the location and spread of the CTQ feature and about the capability to meet the specifications of the corresponding process. When using this chart, a process will be *capable* if the process capability index is higher than a certain value k, with $k > 1$. The most used values for k are $k = 1$, $k = 4/3$, or $k = 5/3$, even 2 at a Six Sigma level, taking into account the usual index limits for which a process could be assumed capable. It will also be assumed that the target value matches the center of the specification interval, that is, $T = \frac{(USL+LSL)}{2} = m$. Then, one of the indices defined by the $C_p(u,v)$ family is used, e.g., C_{pk} or C_{pm}, and the process will be defined as capable if $C_p(u,v) > k$, given the values of u, v, and k. Also note that if $\mu = T$, all the $C_p(u,v)$ indices are defined by the same expression as the C_p. Moreover, different setting for u, v, and k impose different constraints on the process parameters (μ, σ). This can be easily seen through a process capability plot. This graph is a contour plot of $C_p(u,v) = k$ as a function of μ and σ, but it can also be defined as a function of δ and γ, with $\delta = \frac{\mu-T}{d}$ and $\gamma = \frac{\sigma}{d}$. The contour line is obtained by rewriting the index $C_p(u,v)$ as a function of δ and γ as follows $C_p(u,v) = \frac{1-u|\delta|}{3\sqrt{\gamma^2+v(\delta)^2}}$. Therefore, the $C_p(u,v) = k$ equation is solved, plotting γ depending on the values of δ. The resulting expressions are:

$$\gamma = \sqrt{\frac{(1-u|\delta|)}{9k^2} - v\delta^2},\ |\delta| \leq \frac{1}{u+3k\sqrt{v}},\ (u,v) \neq (0,0)$$

When $u = v = 0$, that is, when we consider the index $C_p = k$, we have $\gamma = \frac{1}{3k}$ and $|\delta| \leq 1$. It is important to highlight that the γ-axis accounts for the process spread, whereas the δ-axis accounts for the process location. The values of the parameters μ and σ which provide values (δ, γ) within the region bounded by the contour line $C_p(u,v) = k$ and the δ-axis will provide a larger $C_p(u,v)$

value than k, leading a capable process. Furthermore, values of μ and σ which provide values (δ, γ) outside this region will provide a value $C_p\,(u, v)$ smaller than k, i.e., a non-capable process. In the case of the process not being capable, this type of plot is useful to understand if the corrective actions have to be performed to decrease the process spread, or the process location (deviation with respect to target), or even when both changes are needed to improve the process capability. This can be observed by observing the distance with respect to the x and y-axis. Below are some examples of capability plot application which can be generated through the application of the `qcs.cp` function with `contour=TRUE` and `k=1` (default values):

```
R> oldpar <- par(mfrow = c(2, 2))
R> qcs.cp(object = xbar, parameters = c(0, 0), limits = limits,
+         ylim = c(0, 1))
R> qcs.cp(object = xbar, parameters = c(1, 0), limits = limits,
+         ylim = c(0, 1))
R> qcs.cp(object = xbar, parameters = c(0, 1), limits = limits,
+         ylim = c(0, 1))
R> qcs.cp(object = xbar, parameters = c(1, 1), limits = limits,
+         ylim = c(0, 1))
R> par(oldpar)
```

The result is shown in Figure 11. In all the cases, the points in red are out of the area defined by the line in blue and the δ axis. Thus, the corresponding process is not capable, no matter the capability index that is used. In any case, note that the C_p index is useless in identifying non-capable processes due to location shifts with respect to the target. In the same way, the C_{pk} index assumes as capable processes that are far from the target as long as they were close to the specification limits (as shown in Figure 11). Thus, the use of the C_{pm} and C_{pmk} are recommended due to they take into account both shifts from the target and the spread. In the present case, the process is not capable due to the spread rather than the target shift. Therefore, the process changes could be due to decreases in the variability process.

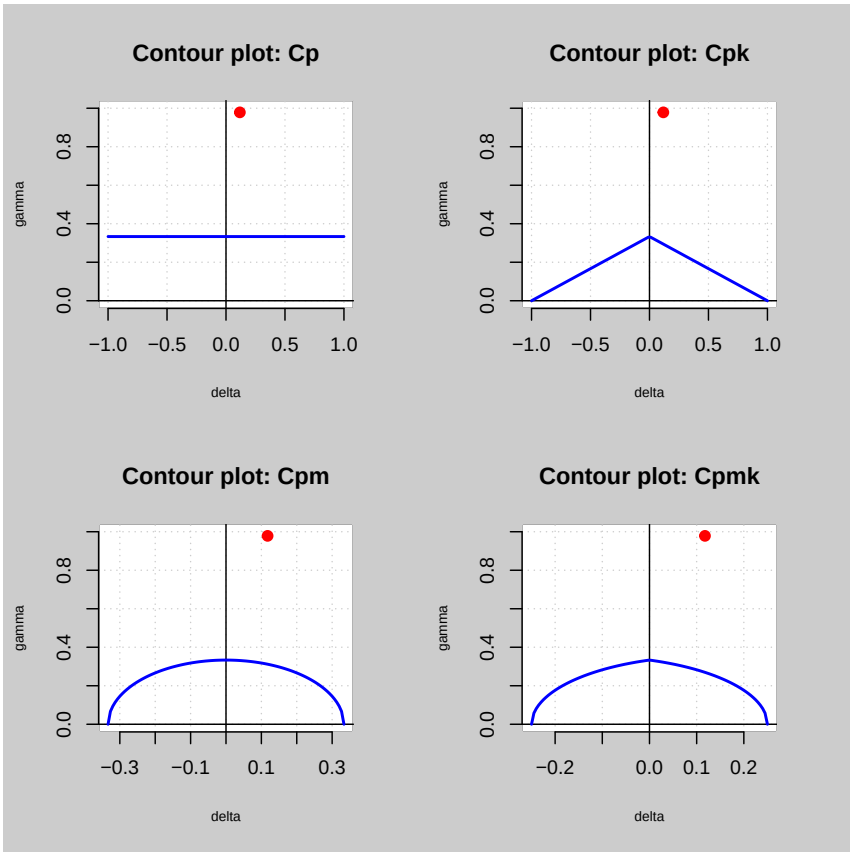


Figure 11: Process capability plots using the C_p, C_{pk}, C_{pm}, and C_{pmk} indexes.

Estimated process capability plot

In practice, the process parameters are unknown and we need to estimate them. We can perform a decision rule based on the sample statistics that provide a sample estimate of the capability index and, finally the so called estimated process capability plot, also called $\gamma^* - \delta^*$ plot (Deleryd and Vännman, 1999). It allows us to decide whether a process is capable or not assuming that μ and σ parameters are unknown and estimated by $\hat{\mu} = \bar{X}$ and $\hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n} X_i^2 - \bar{X}^2$. They are the maximum likelihood estimators when the CTQ variable of the process is normally distributed, and $X_1, X_2, \ldots, X_n$ is a random sample of a normal distribution with μ mean and σ^2 variance.
The **qcr** package only provides the $\gamma^* - \delta^*$ plot corresponding to the C_{pm} index taking into account that the other capability indices do not consider shifts from the target value in their calculations. For the general case, see the work of Vännman (2001). In order to obtain an appropriate decision rule for the case of C_{pm} index, we test the hypotheses $H_0 : C_{pm} \leq k_0$ versus $H_1 : C_{pm} > k_0$, using

$$\hat{C}_{pm} = \frac{d}{3\sqrt{\hat{\sigma}^2 + (\hat{\mu} - T)^2}}$$

as test statistic. The null hypothesis will be rejected if $\hat{C}_{pm} > c_\alpha$, where the constant c_α is determined by previously defining a signifition test level α . Vännman (2001) showed that the null hypothesis $H_0 : C_{pm} \leq k_0$ can be reduced to $H_0 : C_{pm} = k_0$. Thus, for given values of α and n, the process will be considered capable if $\hat{C}_{pm} > c_\alpha$, with $c_\alpha > k_0$. Hubele and Vannman (2004) proved that, when the C_{pm} index is used, the critical value for a given α is obtained as

$$c_\alpha = k_0\sqrt{\frac{n}{\chi^2_{\alpha,n}}},$$

where $\chi^2_{\alpha,n}$ is the quantile α of a χ^2 distribution with n degrees of freedom. The **qcr** package includes the `qcs.hat.cpm()` function to obtain both the theoretical capability plot and the estimated capability plot from sample statistics. Among other options, the user can indicate the control chart from which the estimates $\hat{\mu}$ and $\hat{\sigma}$ are obtained (alternatively, $\hat{\mu}$ and $\hat{\sigma}$ can be introduced through `mu` and `std.dev`), and the specification limits using `limits`. Furthermore, the signification level and the capability limit can be modified, as they are set to $\alpha = 0.05$ and $k_0 = 1$ by default. The following code illustrates its application to `pistonrings` data.

```
R> xbar <- qcs.xbar(pistonrings[1:125, ], plot = FALSE)
R> limits <- c(lsl = 73.99, usl = 74.01)
R> # qcs.hat.cpm(object = xbar, limits = limits, ylim = c(0,1))
R> mu <- xbar$center
R> std.dev <- xbar$std.dev
R> qcs.hat.cpm(limits = limits, mu = mu, std.dev = std.dev, ylim = c(0,1))
```

The result is shown in Figure 12. The contour line corresponding to the capability region obtained from the capability index sample is always more restrictive than the corresponding theoretical one.

Nonparametric capability analysis

Traditional assumptions about data such as normality or independence are frequently violated in many real situations. Thus, in scenarios in which assumptions of normality are not verified, the indices defined in the previous sections are not valid. Pearn and Chen (1997) and Tong and Chen (1998) proposed generalizations of $C_p\,(u, v)$ for the case of arbitrary distributions of data

$$C_{Np}\,(u, v) = \frac{d - u|M - m|}{3\sqrt{\left(\frac{F_{99.865} - F_{0.135}}{6}\right)^2 + v\,(M - T)^2}},$$

where F_α is the percentile $\alpha\%$ of the corresponding distribution and M the median of the process. However, the distribution of the underlying process is always unknown. Chang and Lu (1994) calculated estimates for $F_{99.865}$, $F_{0.135}$ and M based on the sample percentiles.

Pearn and Chen (1997) proposed the following estimator

$$\hat{C}_{Np}\,(u, v) = \frac{d - u|\hat{M} - m|}{3\sqrt{\left(\frac{U_p - L_p}{6}\right)^2 + v\,\left(\hat{M} - T\right)^2}}$$

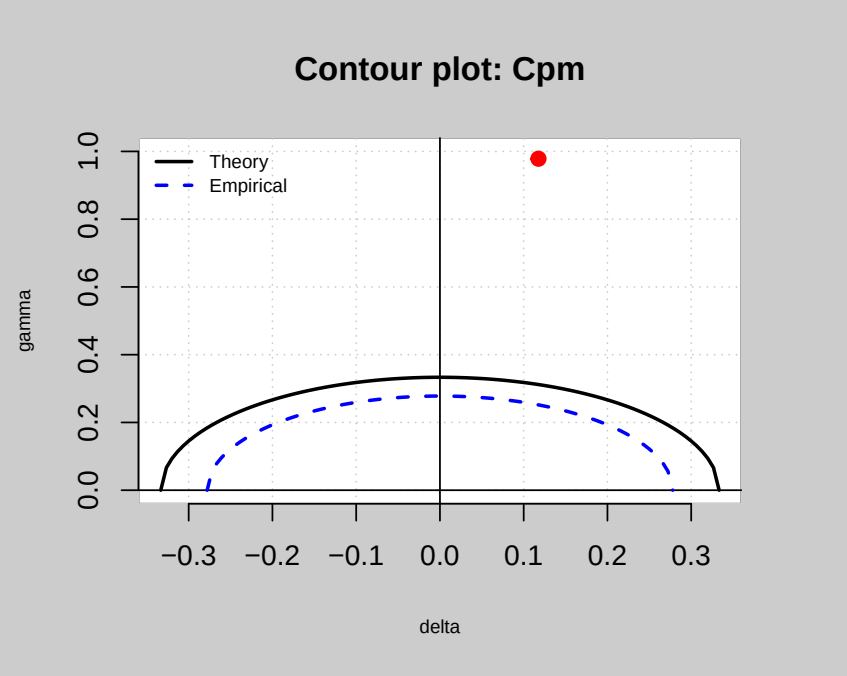


Figure 12: Comparison between theorical and estimated process capability plots.

where U_p is an estimator for $F_{99.865}$, L_p is an estimator for $F_{0.135}$, and $\hat{M}$ is an estimator for M, obtained from the tables developed by Gruska et al. (1989).

The `qcs.cpn()` function of **qcr** calculates C_{Np}, C_{Npk}, C_{Npm}, and C_{Npmk} using the formulation described by Tong and Chen (1998). The code that illustrates its use is shown below. To obtain the nonparametric capability indices it is necessary to indicate the u and v parameters.

```
R> xbar <- qcs.xbar(pistonrings[1:125, ], plot = FALSE)
R> limits <- c(lsl = 73.99, usl = 74.01)
R> # x <- xbar$statistics[[1]]
R> # median <- median(x)
R> # q = quantile(x, probs = c(0.00135, 0.99865)) # c(lq, uq)
R> # qcs.cpn(parameters = c(0, 0), limits = limits, median = median, q = q)
R> # qcs.cpn(object = xbar, parameters = c(0, 0), limits = limits)
R> # qcs.cpn(object = xbar, parameters = c(1, 0), limits = limits)
R> # qcs.cpn(object = xbar, parameters = c(0, 1), limits = limits)
R> qcs.cpn(object = xbar, parameters = c(1, 1), limits = limits)

 CNpmk
0.9015
```

Thus, the values obtained are $C_{Np} = 1.0082$, $C_{Npk} = 0.9275$, $C_{Npm} = 0.9799$ and $C_{Npmk} = 0.9015$. If a capability limit of $k = 1$ or $k = 1.33$ is assumed, we can infer that the process is not actually capable to meet the customers or manager's requirements.

Tools for a comprehensive processs capability analysis

Function `qcs.ca()` provides a comprehensive information of the capability of a process, summarized through a graphical output. This function calculates the process capability indices C_p, C_{pk}, C_{pL}, C_{pU}, C_{pm}, C_{pmk} from a `qcs` object, assuming a Gaussian distribution. Moreover, it computes confidence limits for C_p using the method described by Chou et al. (1990). Approximate confidence limits for C_{pl}, C_{pu}, and C_{pk} are also estimated using the method described in Bissell (1990), while the confidence limits for C_{pm} are based on the approximated method of Boyles (1991) that assumes the target is the mean of the specification limits. Moreover, the C_{Np}, C_{Npk}, C_{Npm}, and C_{Npmk} nonparametric capability indices are also obtained. There is also a specific box within the summary plot that shows the proportion of observations and expected observations under the Gaussian assumption out of the specification limits (nonconforming observations). Further, a histogram of the

data sample is provided, in addition to the corresponding Gaussian density curves obtained from the sample estimates (one per standard deviation estimate procedure). They are displayed along with the specification limits, a quantile-quantile plot for the specified distribution, and a process capability plot obtained from the C_{pm} index (both using theoretical and sample alternatives). In order to describe the `qcs.ca()` performance, the following code corresponds to the analysis of the first 125 observations of the `pistonrings` dataset (the corresponding output is shown in Figure 13).

```
R> qcs.ca(xbar, limits = c(lsl = 73.99, usl = 74.01))

Process Capability Analysis

Call:
qcs.ca(object = xbar, limits = c(lsl = 73.99, usl = 74.01))

Number of obs = 125          Target = 74
       Center =  74               LSL =  73.99
       StdDev =  0.009785         USL =  74.01

Paremetric Capability indices:

       Value    0.1%   99.9%
Cp    0.3407  0.2771  0.4065
Cp_l  0.3807  0.2739  0.4875
Cp_u  0.3006  0.2021  0.3991
Cp_k  0.3006  0.1944  0.4068
Cpm   0.3382  0.2749  0.4038


Non parametric Capability indices:

        Value
CNp    1.0082
CNpK   0.9275
CNpm   0.9799
CNpmk  0.9015


PPM:

         Exp<LSL 1.267e+07       Obs<LSL 0
         Exp>USL 1.836e+07       Obs>USL 8e+05
       Exp Total 3.103e+07     Obs Total 8e+05

Test:


        Anderson Darling Test for normal distribution

data:  xbar
A = 0.1399, mean = 74.001, sd = 0.005, p-value = 0.9694
alternative hypothesis: true distribution is not equal to normal
```

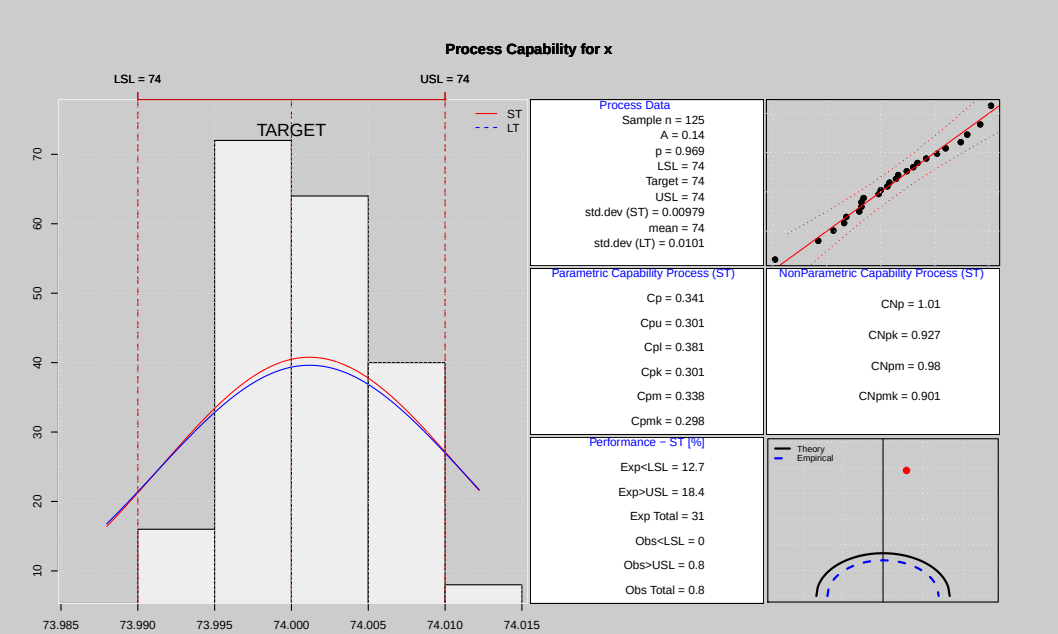


Figure 13: A complete analysis of the process capability.

Conclusions

The qcr package has been developed to provide users with a comprehensive set of functions that manage statistical process control, ranging from univariate parametric analysis to multivariate and FDA nonparametric statistics. This package includes the main types of control charts and capability indices. It combines the main features of reputed SQC packages in R such as qcc and **qualityTools** with the proposal of a new graphical appearance and the implementation of new SQC tools with increasing importance in Industry 4.0 such as multivariate and nonparametric analysis.
In addition to some utilities provided by reference R packages such as qcc, SixSigma, and **qualityTools**, qcr implements very important statistical techniques of Control and Analysis tasks of the Six Sigma procedure that are not included in other libraries. In the case of multivariate control charts, these tools are the MEWMA and MCUSUM multivariate control charts, on the one hand, and the r, Q and S nonparametric control charts based on data depth, on the other hand. In addition, Phase I and Phase II control charts for functional data (monthly, daily, hourly curves) based on functional data depth, bootstrap procedures, and nonparametric rank charts have also been implemented in the qcr package. These control charts for functional data provide tools to control and improve processes when their CTQ variables are obtained as hourly, monthly, daily, yearly smooth curves.
It is also very important to note that qcr provides functions to perform nonparametric capability analysis. In addition, the new implementation of the process capability plots for the main parametric capability indices allows us to analyze if improvements in process spread or/and process location are needed to obtain a capable process. The comparison between suppliers, machines, etc., is enabled through capability plots.
All these utilities intend to make qcr a useful tool for users of a wide variety of industries, providing a competitive alternative to commercial software.

Acknowledgments

The work of Salvador Naya, Javier Tarrío-Saavedra, Miguel Flores and Rubén Fernández-Casal has been supported by MINECO grant MTM2017-82724-R, and by the Xunta de Galicia (Grupos de Referencia Competitiva ED431C-2020-14 and Centro de Investigación del Sistema universitario de Galicia ED431G 2019/01), all of them through the ERDF. The research of Miguel Flores has been partially supported by Grant PII-DM-002-2016 of Escuela Politécnica Nacional of Ecuador. In addition, the research of Javier Tarrío-Saavedra has been also founded by the eCOAR project (PC18/03) of CITIC. The authors thank Mateo Larco and Bibiana Santiso for their valuable help with the English edition.

Bibliography

F. Barros. *IQCC: Improved Quality Control Charts*, 2017. URL https://CRAN.R-project.org/package=IQCC. R package version 0.7. [p196]

A. Bissell. How reliable is your capability index? *Applied Statistics*, pages 331–340, 1990. [p212]

R. A. Boyles. The Taguchi capability index. *Journal of Quality Technology*, 23:17–26, 1991. [p196, 212]

D. W. Brown and G. B. Wetherill. *Statistical process control*. Chapman and Hall, 1990. [p197]

E. L. Cano, J. M. Moguerza, and A. Redchuk. *Six Sigma with R*, volume 36 of *Use R!* Springer-Verlag, New York, 2012. [p196]

E. L. Cano, J. M. Moguerza, and M. P. Corcoba. *Quality Control with R*. Use R! Springer-Verlag, New York, 2015. [p196]

I. Cascos, A. López, and J. Romo. Data depth in multivariate statistics. *Boletín de Estadística e Investigación Operativa*, 27(3):151–174, 2011. [p200]

C. W. Champ and W. H. Woodall. Exact results for Shewhart control charts with supplementary runs rules. *Technometrics*, 29(4):393–399, 1987. [p194]

P.-L. Chang and K.-H. Lu. PCI calculations for any shape of distribution with percentile. *Quality World Technical Supplement*, pages 110–114, 1994. [p211]

Y.-M. Chou, D. Owen, and S. Borrego A. Lower confidence limits on process capability indices. *Journal of Quality Technology*, 22(3):223–229, 1990. [p212]

A. Cuevas, M. Febrero, and R. Fraiman. Robust estimation and classification for functional data via projection-based depth notions. *Computational Statistics*, 22(3):481–496, 2007. [p204]

M. Deleryd and K. Vännman. Process capability plots – a quality improvement tool. *Quality and Reliability Engineering International*, 15(3):213–227, 1999. [p209, 211]

R. Dyckerhoff. Data depths satisfying the projection property. *Allgemeines Statistisches Archiv*, 88 (2):163–190, 2004. [p200]

M. Flores, S. Naya, R. Fernández-Casal, S. Zaragoza, P. Raña, and J. Tarrío-Saavedra. Constructing a control chart using functional data. *Mathematics*, 8(1):58, 2020. [p204, 205, 206]

M. Flores, R. Fernández-Casal, S. Naya, and J. Tarrío-Saavedra. *qcr: Quality Control Review*, 2021. URL https://CRAN.R-project.org/package=qcr. R package version 1.3. [p198, 202]

R. Fraiman and G. Muniz. Trimmed means for functional data. *Test*, 10(2):419–440, 2001. [p204]

R. Fraiman, R. Y. Liu, and J. Meloche. Multivariate density estimation by probing depth. *Lecture Notes-Monograph Series*, pages 415–430, 1997. [p200]

A. Gandy and J. T. Kvaloy. Guaranteed conditional performance of control charts via bootstrap methods. *Scandinavian Journal of Statistics*, 40:647–668, 2013. doi: 10.1002/sjos.12006. [p196]

G. Gruska, K. Mirkhani, and L. Lamberson. Non normal data analysis. *St. Clair Shores, MI: Applied Computer Solutions, Inc*, 1989. [p212]

N. F. Hubele and K. Vannman. The effect of pooled and un-pooled variance estimators on c_{pm} sub when using subsamples. *Journal of quality technology*, 36(2):207, 2004. [p211]

S. Knoth. *spc: Statistical Process Control – Calculation of ARL and Other Control Chart Performance Measures*, 2021. URL https://CRAN.R-project.org/package=spc. R package version 0.6.5. [p196]

R. Liu. On a notion of data depth based on random simplices. *The Annals of Statistics*, 18(1): 405–414, 1990. [p200]

R. Y. Liu. Control charts for multivariate processes. *Journal of the American Statistical Association*, 90(432):1380–1387, 1995. [p195, 199, 200]

R. Y. Liu and J. Tang. Control charts for dependent and independent measurements based on bootstrap methods. *Journal of the American Statistical Association*, 91(436):1694–1700, 1996. [p197]

P. C. Mahalanobis. On the generalised distance in statistics. *Proceedings of the National Institute of Sciences of India, 1936*, pages 49–55, 1936. [p200]

D. C. Montgomery. *Introduction to statistical quality control*. John Wiley & Sons (New York), 2009. [p194]

P. S. Pande, R. P. Neuman, and R. R. Cavanagh. *The six sigma way: How GE, Motorola, and other top companies are honing their performance*. McGraw-Hill (New York), 2000. [p194]

W. Pearn and K. Chen. A practical implementation of the process capability index c_{pk}. *Quality Engineering*, 9(4):721–737, 1997. [p211]

A. M. Polansky. Process capability indices, nonparametric. *Encyclopedia of Statistics in Quality and Reliability*, 2007. [p196]

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2021. URL https://www.R-project.org/. [p196]

T. Roth. *qualityTools: Statistics in Quality Science.*, 2016. URL http://www.r-qualitytools.org. R package version 1.55 http://www.r-qualitytools.org. [p196]

E. Santos-Fernandez. *Multivariate Statistical Quality Control Using R*, volume 14. Springer-Verlag, 2013. ISBN 9781461454533. URL http://www.springer.com/statistics/computational+statistics/book/978-1-4614-5452-6. [p196, 197]

E. Santos-Fernández and M. Scagliarini. MPCI: An R package for computing multivariate process capability indices. *Journal of Statistical Software*, 47(7):1–15, 2012. URL http://www.jstatsoft.org/v47/i07/. [p196]

L. Scrucca. qcc: An R package for quality control charting and statistical process control. *R News*, 4/1:11–17, 2004. URL https://cran.r-project.org/doc/Rnews/. [p196, 197]

L.-I. Tong and J.-P. Chen. Lower confidence limits of process capability indices for non-normal process distributions. *International Journal of Quality & Reliability Management*, 15(8/9):907–919, 1998. [p211, 212]

J. W. Tukey. Mathematics and the picturing of data. In *Proceedings of the international congress of mathematicians*, volume 2, pages 523–531, 1975. [p200]

K. Vännman. A unified approach to capability indices. *Statistica Sinica*, pages 805–820, 1995. [p208]

K. Vännman. A graphical method to control process capability. In *Frontiers in Statistical Quality Control 6*, pages 290–311. Springer-Verlag, 2001. [p209, 211]

W. Zhu and C. Park. edcc: An R package for the economic design of the control chart. *Journal of Statistical Software*, 52(9):1–24, 2013. URL http://www.jstatsoft.org/v52/i09/. [p196]

Y. Zuo and R. Serfling. General notions of statistical depth function. *Annals of statistics*, pages 461–482, 2000. [p200]

Miguel Flores
MODES, SIGTI, ADIAAC, Departamento de Matemáticas, Escuela Politécnica Nacional
170517 Quito
Ecuador
0000-0002-7742-1247
miguel.flores@epn.edu.ec

Rubén Fernández-Casal
MODES, CITIC, Universidade da Coruña
Facultade de Informática, Campus de Elviña, A Coruña
Spain
0000-0002-5785-3739
ruben.fcasal@udc.es

Salvador Naya
MODES, CITIC, ITMATI, Universidade da Coruña
Escola Politécnica Superior, Mendizábal s/n, Ferrol
Spain
0000-0003-4931-9859
salva@udc.es

Javier Tarrío-Saavedra
MODES, CITIC, Universidade da Coruña
Escola Politécnica Superior, Mendizábal s/n, Ferrol
Spain
0000-0003-4931-9859
javier.tarrio@udc.es

pdynmc: A Package for Estimating Linear Dynamic Panel Data Models Based on Nonlinear Moment Conditions

by Markus Fritsch, Andrew Adrian Yu Pua and Joachim Schnurbus

Abstract This paper introduces **pdynmc**, an R package that provides users sufficient flexibility and precise control over the estimation and inference in linear dynamic panel data models. The package primarily allows for the inclusion of nonlinear moment conditions and the use of iterated GMM; additionally, visualizations for data structure and estimation results are provided. The current implementation reflects recent developments in literature, uses sensible argument defaults, and aligns commercial and noncommercial estimation commands. Since the understanding of the model assumptions is vital for setting up plausible estimation routines, we provide a broad introduction of linear dynamic panel data models directed towards practitioners before concisely describing the functionality available in **pdynmc** regarding instrument type, covariate type, estimation methodology, and general configuration. We then demonstrate the functionality by revisiting the popular firm-level dataset of Arellano and Bond (1991).

Introduction

This paper introduces the contributed package **pdynmc** (Fritsch et al., 2020) – a unified framework for estimating linear dynamic panel data models based on linear and nonlinear moment conditions (Ahn and Schmidt, 1995). Our implementation of the commands in **pdynmc** allows the user to exert precise control over the available functionality, reflects recent developments in literature, uses sensible argument defaults, aligns commercial and noncommercial estimation commands, and provides visualizations of data structure and estimation results. Additionally, this paper provides a concise introduction into linear dynamic panel data models directed towards the practitioner, describes the functionality available in **pdynmc**, and walks the reader through estimation of linear dynamic panel data models by replicating the analysis in Arellano and Bond (1991).

Practitioners have a variety of recent packages that enable linear dynamic panel data modeling meant for a fixed number of time periods. In particular, contributed R packages such as **OrthoPanels** (Pickup et al., 2017), **plm** (Croissant and Millo, 2019), and **panelvar** (Sigmund and Ferstl, 2019) have considerably enlarged the set of noncommercial routines available. All of these packages implement some default routines for estimating common parameters in linear dynamic panel data models. **OrthoPanels** implements a likelihood-based orthogonal reparameterization procedure for first-order autoregressive linear panel data models with strictly exogenous covariates. **plm** implements one-step and two-step GMM-based procedures for pth-order autoregressive linear panel data models. **panelvar** implements iterated (or "m-step", compare Sigmund and Ferstl, 2019) GMM procedures for pth-order vector autoregressive linear panel data models. For the latter two packages, linear moment conditions are used to identify common parameters.

Additional functionality of our contributed package **pdynmc** includes nonlinear moment conditions which are generally not available across standard GMM estimation routines. To the best of our knowledge, there is currently only the xtdpdgmm-implementation provided by (Kripfganz, 2019) for the commercial statistical software Stata (StataCorp, 2015). The current implementation in Stata restricts accessibility to the routine as it requires a recent Stata version (version 13 or higher). Another key estimation option provided by **pdynmc** is iterated GMM. Hansen and Lee (2021) recently outlined the merits of the technique and developed the theory under potential misspecification of moment conditions. The availability of iterated GMM for dynamic panels may help to apply the results found in, for example, Hwang and Sun (2018). Visualizations of the estimation results of iterated GMM are also included.

The structure of the paper is as follows. Section Framework and methodology briefly sketches the linear dynamic panel data model, states underlying assumptions frequently used in literature, and describes moment conditions arising from different sets of model assumptions. Section R implementation covers details on the arguments of the model fitting function `pdynmc` and connects the description with the estimation methodology. Section Empirical example illustrates the estimation of linear dynamic panel data models with **pdynmc** for the dataset of Arellano and Bond (1991), while Section Conclusion concludes.

Framework and methodology

Linear dynamic panel data models account for dynamics and unobserved individual-specific heterogeneity. Due to the presence of lagged dependent variables, applying ordinary least squares including individual-specific dummy variables is inconsistent (see, e.g., Hsiao, 2014). A suitable alternative for obtaining parameter estimates of linear dynamic panel data models is deriving moment conditions (or population orthogonality conditions) from the model assumptions. The moment conditions may be linear (Anderson and Hsiao, 1982; Holtz-Eakin et al., 1988; Arellano and Bover, 1995) or nonlinear (Ahn and Schmidt, 1995) in parameters and determine the natural instruments available for estimation. Usually, the number of moment conditions exceeds the number of parameters, and the moment conditions need to be aggregated appropriately. This can be achieved by the generalized method of moments (GMM), where weighted linear combinations of moment conditions are employed to obtain parameter estimates.

Theoretical results and evidence from Monte Carlo simulations in the literature suggest that incorporating nonlinear (quadratic) moment conditions proposed by Ahn and Schmidt (1995) is valuable for identification. For example, when the lag parameter exhibits high persistence, linear moment conditions fail to identify the model parameters, while quadratic moment conditions can still provide identification (Bun and Kleibergen, 2021; Bun and Sarafidis, 2015; Gørgens et al., 2019; Pua et al., 2019a,b). Note that the quadratic moment conditions are immediate by-products of imposing standard assumptions, which are the basis of the Arellano and Bond (1991) estimator – the most popular default routine in dynamic panel data estimation.

Since the moment conditions employed in GMM estimation of linear dynamic panel data models are derived from model assumptions, a basic understanding of these assumptions is vital for setting up a plausible estimation routine. We briefly review the assumptions implied when using particular moment conditions in the estimation below and add to the exposition in the **plm** vignette (Croissant and Millo, 2019), where the function `pgmm` is used to estimate linear dynamic panel data models. For further reading on the methodology, we suggest Fritsch (2019).

Linear dynamic panel data model

For a given dataset with cross-section dimension n and time series dimension T, consider a linear dynamic panel data model of the form:

$$y_{i,t} = \alpha y_{i,t-1} + \beta x_{i,t} + u_{i,t}, \quad i = 1, \ldots, n;\ t = 2, \ldots, T, \tag{1}$$

$$u_{i,t} = \eta_i + \varepsilon_{i,t}. \tag{2}$$

Variables $y_{i,t}$ and $y_{i,t-1}$ denote the dependent variable and its lag, α is the lag parameter, and $x_{i,t}$ is a single covariate with corresponding slope coefficient β. The second equation separates the composite error term $u_{i,t}$ into an unobserved individual-specific effect η_i and an idiosyncratic remainder component $\varepsilon_{i,t}$.

Combining Equations (1) and (2) yields the single equation form:

$$y_{i,t} = \alpha y_{i,t-1} + \beta x_{i,t} + \eta_i + \varepsilon_{i,t}. \tag{3}$$

We only include one lag of the dependent variable, one covariate, and omit unobserved time-specific effects in this section for simplicity of exposition and notational convenience. Extending the representation is straightforward, and **pdynmc** can also accommodate AR(p) models and time effects. The initial time period is denoted by $t = 1$.

The unobserved individual-specific effects η_i may be eliminated from Equation (3) by taking first differences:

$$\Delta y_{i,t} = \alpha \Delta y_{i,t-1} + \beta \Delta x_{i,t} + \Delta \varepsilon_{i,t}. \tag{4}$$

Since the first difference of the lagged dependent variable $\Delta y_{i,t-1} = y_{i,t-1} - y_{i,t-2}$ and the first difference of the idiosyncratic remainder component $\Delta \varepsilon_{i,t} = \varepsilon_{i,t} - \varepsilon_{i,t-1}$ are not orthogonal, ordinary least squares estimation of Equation (4) is inconsistent. Linear dynamic panel data models are usually estimated by GMM, where corresponding moment conditions are derived from model assumptions.

Standard assumptions and associated moment conditions

Researchers have focused on the following standard assumptions, henceforth StA, (see Ahn and Schmidt, 1995):

$$\begin{aligned}
&\text{The data are independently distributed across } i, \\
&E(\eta_i) = 0, \quad i = 1, ..., n, \\
&E(\varepsilon_{i,t}) = 0, \quad i = 1, ..., n, \ t = 2, ..., T, \\
&E(\varepsilon_{i,t} \cdot \eta_i) = 0, \quad i = 1, ..., n, \ t = 2, ..., T, \\
&E(\varepsilon_{i,t} \cdot \varepsilon_{i,s}) = 0, \quad i = 1, ..., n, \ t \neq s, \\
&E(y_{i,1} \cdot \varepsilon_{i,t}) = 0, \quad i = 1, ..., n, \ t = 2, ..., T, \\
&n \to \infty, \text{ while } T \text{ is fixed, such that } \frac{T}{n} \to 0.
\end{aligned} \tag{5}$$

We assume that StA hold for the rest of this paper.

Under StA, Holtz-Eakin et al. (1988) (henceforth **HNR**) propose the moment conditions:

$$E(y_{i,s} \cdot \Delta u_{i,t}) = 0, \qquad t = 3, \ldots, T; \ s = 1, \ldots, t-2. \tag{6}$$

Equation (6) provides $0.5(T-1)(T-2)$ moment conditions. Equivalent moment conditions can be derived from the covariate $x_{i,t}$ – depending on its correlation with the idiosyncratic remainder component $\varepsilon_{i,t}$. Endogenous, predetermined, and (strictly) exogenous covariates provide the following linear moment conditions, respectively:

$$\begin{aligned}
E(x_{i,s} \cdot \Delta u_{i,t}) = 0, \qquad & t = 3, \ldots, T, && \text{where} \\
& s = 1, \ldots, t-2, && \text{for} \quad \text{endogenous } x, \\
& s = 1, \ldots, t-1, && \text{for} \quad \text{predetermined } x, \\
& s = 1, \ldots, T, && \text{for} \quad \text{strictly exogenous } x.
\end{aligned} \tag{7}$$

After solving Equation (1) for $u_{i,t}$ and inserting this as $\Delta u_{i,t} = u_{i,t} - u_{i,t-1}$, it is apparent that the HNR moment conditions are linear in parameters (α and β). In literature, the HNR moment conditions also appear as "moment conditions with instruments in levels" (w.r.t. $y_{i,s}, x_{i,s}$) and "moment conditions from equations in differences" (w.r.t. $\Delta u_{i,t}$).

Ahn and Schmidt (1995) (henceforth **AS**) have shown that under StA the following $T-3$ additional moment conditions hold:

$$E(u_{i,T} \cdot \Delta u_{i,t-1}) = 0, \qquad t = 4, \ldots, T. \tag{8}$$

These moment conditions are nonlinear in parameters (quadratic in α and β).

Extended assumptions and associated moment conditions

Another set of moment conditions, beyond those implied by StA, that is popular in theoretical and applied research is derived from the assumption:

$$E(\Delta y_{i,t} \cdot \eta_i) = 0, \quad i = 1, \ldots, n. \tag{9}$$

This expression requires that the dependent variable and the unobserved individual-specific effects are constantly correlated over time for each individual and has thus been called "constant correlated effects" (Bun and Sarafidis, 2015). This assumption is also called "effect stationarity" (Kiviet, 2007) or "mean stationarity" (Arellano, 2003).

From this assumption, Arellano and Bover (1995) derive $T-2$ additional moment conditions (henceforth **ABov**):

$$E(\Delta y_{i,t-1} \cdot u_{i,t}) = 0, \quad t = 3, \ldots, T. \tag{10}$$

By rewriting these moment conditions, it can be shown that the ABov moment conditions encompass the nonlinear AS moment conditions (for a derivation see Fritsch, 2019).

Depending on the assumptions about $x_{i,t}$, additional ABov moment conditions can be derived:

$$E(\Delta x_{i,v} \cdot u_{i,t}) = 0, \qquad \text{where}$$
$$v = t-1;\ t = 3,\dots T, \quad \text{for} \quad x \text{ endogenous},$$
$$v = t;\ t = 2,\dots,T, \quad \text{for} \quad x \text{ strictly exogenous} \quad \text{or} \quad x \text{ predetermined}.$$

Deviations from the assumption are required to be unsystematic over both the cross-section and the time series dimension (see Section 6.5 in Arellano, 2003, which also provides an empirically relevant example). Employing the constant correlated effects assumption implicitly constrains the relationship between $\Delta x_{i,t}$ and η_i (see Blundell et al., 2001). If the statistician is not willing to impose this restriction, nonlinear AS moment conditions can be used instead.

R implementation

Similar to function `pgmm` in the package **plm**, **pdynmc** provides one-step and two-step closed-form GMM estimators and standard specification testing such as overidentifying restrictions tests, serial correlation tests, and Wald tests. These features are shared by other packages implemented in Gauss, Ox (Doornik et al., 2012), R, and Stata. We provide options to match results from other statistical software estimation routines. In contrast to **OrthoPanels** and **plm**, **pdynmc** does not include a formula interface to allow the user to exert full control over all functionality.

GMM estimation, inference, and testing

We provide one-step, two-step, and iterated estimation for the coefficients. The weighting matrix of the moment conditions plays a prominent role in estimation (Arellano and Bond, 1991; Blundell et al., 2001; Kripfganz, 2019). An optimal weighting matrix is proportional to the inverse of the covariance matrix of the moment conditions (see, e.g., Arellano, 2003). The default weighting matrix used in **pdynmc** is based on the proposal of Arellano and Bond (1991). For details on available alternatives, see the documentation of **pdynmc** and the corresponding package vignette (Fritsch et al., 2020).

Details on the computation of asymptotic one- and two-step standard errors can be found in Doornik et al. (2012). As asymptotic two-step GMM standard errors for the estimated coefficients exhibit a downward bias in small samples, they can be substantially lower than one-step GMM standard errors (see, e.g., Arellano and Bond, 1991). Windmeijer (2005) relates the bias to the dependence of the two-step weighting matrix on one-step parameter estimates and proposes an analytic correction of two-step standard errors. Robust and non-robust versions of the standard errors are available.

Coefficient estimates and standard errors from one- and two-step GMM estimation can be misleading. An example of high practical relevance is when the estimated model is a reasonable approximation instead of the true functional relationship (Hansen and Lee, 2021; Hwang et al., 2021): This may render some of the moment conditions invalid. Hansen and Lee (2021) highlight the arbitrariness of the initial weighting matrix and note that iterated GMM provides a remedy. Across iterations, the weighting matrix is updated based on the residuals of the previous estimation step (for more details, see Hansen and Lee, 2021, p. 4–6). Iterated GMM is used as a default in **pdynmc**.

We implement the following tests:

- The serial correlation test of Arellano (2003).
- The overidentifying restrictions test of Hansen (1982), called "J-test".
- A Wald test of joint significance of (i) coefficients of lagged-dependent variable(s) and covariates; (ii) time dummy coefficients; (iii) both, (i) and (ii).

When nonlinear moment conditions are used in GMM estimation, nonlinear optimization techniques are required to obtain coefficient estimates. By default, GMM estimation by **pdynmc** is based on numerical optimization. For the optimization procedure, we rely on R-package **optimx** (Nash and Varadhan, 2011; Nash, 2014). All optimization routines implemented in **optimx** are available in **pdynmc**. Based on our experience, we recommend using the Variable Metric method (Fletcher, 1970; Nash, 1990, 2020) in the estimation of linear dynamic panel data models. The technique is labeled `BFGS` in **optimx** and serves as the default procedure in **pdynmc**.

Function arguments explained

In package **pdynmc**, the eponymous function is intended for model fitting. The most important function arguments for applied work are summarized in Table 1.

Argument	Description
`dat`	Dataset with rows (i.e., observations) and columns (i.e., variables).
`varname.i`	Cross-section identifier (column name).
`varname.t`	Time series identifier (column name).
`use.mc.diff`	Use moment conditions from Equation (6) (i.e., equations in differences, instruments in levels).
`use.mc.lev`	Use moment conditions from Equation (10) (i.e., equations in levels, instruments in differences).
`use.mc.nonlin`	Use nonlinear (quadratic) moment conditions from Equation (8).
`use.mc.nonlinAS`	If turned to `FALSE`, the nonlinear moment conditions are used in a modified version (that is also valid under StA), where T in Equation (8) is replaced by t.
`include.y`	Derive instruments from lags of dependent variable.
`varname.y`	Name of dependent variable in dataset.
`lagTerms.y`	Number of lags of dependent variable.
`maxLags.y`	Maximum number of lags of dependent variable from which to derive instruments.
`include.x`	Derive instruments from covariates.
`varname.reg.end`	Name(s) of covariate(s) to be treated as endogenous (replace suffix `end` by `pre` for predetermined; by `ex` for exogenous covariates).
`lagTerms.reg.end`	Number of lags of endogenous covariate(s) (also for `pre` or `ex`).
`maxLags.reg.end`	Maximum number of lags of endogenous covariate(s) used to derive instruments (also for `pre` or `ex`).
`fur.con`	Include further control variables (i.e., covariates that are not used for deriving instruments).
`fur.con.diff`	Logical variable indicating whether to include further control variables in equations in differences.
`fur.con.lev`	Logical variable indicating whether to include further control variables in equations in levels.
`varname.reg.fur`	Name(s) of covariate(s) in dataset to be treated as further controls.
`lagTerms.reg.fur`	Number of lags of further controls.
`include.dum`	Include time dummies. Note: Can be constructed from multiple variables.
`dum.diff`	Include time dummies in equations in first differences.
`dum.lev`	Include time dummies in equations in levels.
`varname.dum`	Variable name(s) for creating time dummies (can be different from `varname.t`).
`w.mat`	Type of initial weighting matrix to be used, `iid.err` (as proposed by Arellano and Bond (1991)), `identity`, or `zero.cov`.
`std.err`	Type of standard errors to be used, either bias-corrected (`corrected`) according to Windmeijer (2005) or not (`unadjusted`).
`estimation`	Type of estimation, `onestep`, `twostep`, or `iterative`.

Table 1: Most-used arguments of function `pdynmc`.

Besides the arguments described in Table 1, various further configuration options exist for function `pdynmc`. The function allows for the inclusion of further covariates which are only used as instruments (i.e., covariates from which moment conditions are derived, but for which no parameters are estimated; compare arguments `include.x.instr` and `varname.reg.instr`) as well as the opposite, covariates which are instrumented (i.e., covariates for which parameters are estimated, but from which no moment conditions are derived; compare arguments `include.x.toInstr` and `varname.reg.toInstr`).

Further, thresholds for collinearity checks can be adjusted via `col_tol`. The total number of instruments above which a generalized inverse is used to invert the weighting matrix can be specified by `inst.thresh`.

When only linear moment conditions are used, a closed-form solution exists for the estimator, and nonlinear optimization can be turned off (by `opt.meth = "none"`). Package **optimx** is employed for nonlinear optimization and argument `hessian` controls whether the Hessian matrix is approximated in estimation. All other control arguments for **optimx** can be provided in a list via `optCtrl`. Starting values for initializing nonlinear optimization are drawn from the uniform distribution in the interval [-1, 1] via `start.val.lo = -1` and `start.val.hi = 1`; a seed (`seed.input = 42`) ensures reproducibility (limits and seed can be adjusted). Alternatively, specific starting values can be provided via argument `start.val` (note that `custom.start.val` has to be set to `TRUE`).

For iterated estimation, termination criteria can be set via `max.iter` (maximum number of iterations) and `iter.tol` (tolerance for determining convergence).

The Stata packages xtabond2 (Roodman, 2018) and xtdpdgmm have somewhat different usage and weighting of moment conditions. If HNR and ABov moment conditions are available for estimation, some of the ABov moment conditions are redundant (see Fritsch, 2019, for a derivation). While the Stata routines fully expand the linear ABov moment conditions when setting up the instrument matrix (including the redundant moment conditions), **pdynmc** omits the redundant moment conditions. The **pdynmc** arguments `inst.stata` and `w.mat.stata` are included to allow for conformity to Stata and to reproduce estimation results.

Empirical example

The functionality of **pdynmc** is illustrated by replicating Arellano and Bond (1991) in a wide sense as we incorporate linear ABov and nonlinear AS moment conditions into the analysis; we also draw comparisons between **pdynmc**, the `pgmm` (Croissant et al., 2020) function in R-package **plm**, and Stata implementations xtabond2 and xtdpdgmm (Kripfganz, 2019).

Arellano and Bond (1991) employ an unbalanced panel of $n = 140$ firms located in the UK. The dataset spans $T = 9$ time periods and is available from R package **plm**. Arellano and Bond (1991) investigate employment equations and consider the dynamic specification:

$$\begin{aligned} n_{i,t} =& \alpha_1 n_{i,t-1} + \alpha_2 n_{i,t-2} + \\ & \beta_1 w_{i,t} + \beta_2 w_{i,t-1} + \beta_3 k_{i,t} + \beta_4 k_{i,t-1} + \beta_5 k_{i,t-2} + \beta_6 ys_{i,t} + \beta_7 ys_{i,t-1} + \beta_8 ys_{i,t-2} + \\ & \gamma_3 d_3 + \cdots + \gamma_T d_T + \eta_i + \varepsilon_{i,t}, \qquad i = 1, ..., n;\ t = 3, ..., T, \end{aligned} \tag{11}$$

where i denotes the firm, and t is the time series dimension. The natural logarithm of employment (n) is explained by its first two lags and the further covariates natural logarithm of wage (w), the natural logarithm of capital (k), the natural logarithm of output (ys), and their lags of order up to one (for w) or two (for k and ys). Variables $d_3, \ldots, d_T$ are time dummies with corresponding coefficients $\gamma_3, \ldots, \gamma_T$; unobserved individual-specific effects are represented by η, and ε is an idiosyncratic remainder component.

We load the dataset and compute logarithms of the four mentioned variables via:

```
data(EmplUK, package = "plm")
dat             <- EmplUK
dat[,c(4:7)]    <- log(dat[,c(4:7)])
names(dat)[4:7] <- c("n", "w", "k", "ys")
```

As GMM estimation with linear and/or nonlinear moment conditions in **pdynmc** allows for arbitrary unbalancedness, we included the functions `data.info` and `strucUPD.plot` to provide an overview of the panel data structure. Both functions require the column name of cross-section (`i.name`) and time series identifier (`t.name`). Using `data.info(dat,i.name = "firm",t.name = "year")` yields:

```
Unbalanced panel dataset with 1031 rows and the following time period frequencies:
1976 1977 1978 1979 1980 1981 1982 1983 1984
  80  138  140  140  140  140  140   78   35
```

The command `strucUPD.plot(dat,i.name = "firm",t.name = "year")` gives the visual representation of the panel data structure shown in Figure 1. Figure 1 indicates the time periods (compare

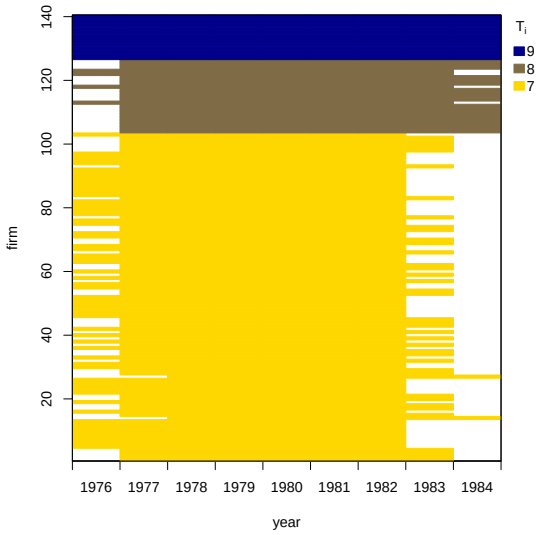


Figure 1: Unbalanced panel structure plot.

abscissa) available for each cross-sectional unit (ordinate). Blank areas represent missing observations. The coloring scheme shows the number of time series units that are available for the corresponding cross-sectional units. We see that the given dataset has already been ordered by the number of time periods available.

The goal of the empirical analysis is to estimate the lag parameters α_1 and α_2 and the coefficients β_j of the $j = 1, \ldots, 8$ further covariates while controlling for (unobserved) time effects and accounting for unobserved individual-specific heterogeneity. In the following, we first apply **pdynmc** to replicate the original results of Arellano and Bond (1991) that are based on HNR moment conditions only and introduce the implemented tests. Then, we provide results for adding ABov moment conditions to the analysis. Finally, we discuss results for HNR moment conditions extended by AS moment conditions and apply iterated GMM. All results on estimated coefficients and robust standard errors are summarized in Table 2. Details on employed moment conditions are provided in the table footnotes.

GMM estimation with HNR moment conditions

When reproducing the results in Table 4 on p. 290 of Arellano and Bond (1991) with **pdynmc**, the model structure underlying Equation (11) can be specified and estimated by:

```
m1 <- pdynmc(
  dat = dat, varname.i = "firm", varname.t = "year",
  use.mc.diff = TRUE, use.mc.lev = FALSE, use.mc.nonlin = FALSE,
  include.y = TRUE, varname.y = "n", lagTerms.y = 2,
  fur.con = TRUE, fur.con.diff = TRUE, fur.con.lev = FALSE,
  varname.reg.fur = c("w", "k", "ys"), lagTerms.reg.fur = c(1,2,2),
  include.dum = TRUE, dum.diff = TRUE, dum.lev = FALSE, varname.dum = "year",
  w.mat = "iid.err", std.err = "corrected",
  estimation = "onestep", opt.meth = "none"
)
```

The standard output is accessed via `summary(m1)` and can be found in panel (a) of Table 2. The estimated coefficients reproduce the estimates in Table 4, column (a1) on p. 290 of Arellano and Bond (1991) when one specifies all arguments as stated in this section. Changing the argument `estimation` to `twostep` yields two-step GMM coefficient estimates (the `pdynmc`-output object is assigned to `m2`) from Table 4, column (a2) on p. 290 of Arellano and Bond (1991). These results may be found in panel (b) of Table 2. Note that the standard errors presented in column (b) of Table 2 are based on the Windmeijer-correction and deviate from the conventional standard errors reported in Arellano and Bond (1991). Standard errors from the original analysis can be reproduced by setting `std.err = "unadjusted"`.

	(a) 1-Step Estimate HNR only (SE Rob.)	(b) 2-S. Estimate HNR only (SE Rob.)	(c) 2-S. Estimate HNR & ABov (SE Rob.)	(d) 2-S. Estimate HNR & AS (SE Rob.)	(e) Iterated Est. HNR & AS (SE Rob.)
L1.n	0.686***	0.629**	1.103***	1.112***	1.197***
	(0.145)	(0.193)	(0.050)	(0.066)	(0.069)
L2.n	-0.085	-0.065	-0.104*	-0.071	-0.126
	(0.056)	(0.045)	(0.047)	(0.069)	(0.068)
w	-0.608***	-0.526***	-0.448**	-0.417**	-0.219
	(0.178)	(0.155)	(0.149)	(0.153)	(0.127)
L1.w	0.393*	0.311	0.423**	0.413**	0.258
	(0.168)	(0.203)	(0.156)	(0.160)	(0.138)
k	0.357***	0.278***	0.290***	0.309***	0.255***
	(0.059)	(0.073)	(0.050)	(0.053)	(0.056)
L1.k	-0.058	0.014	-0.153*	-0.189**	-0.155*
	(0.073)	(0.092)	(0.067)	(0.068)	(0.077)
L2.k	-0.020	-0.040	-0.137***	-0.154**	-0.156**
	(0.033)	(0.043)	(0.041)	(0.050)	(0.055)
ys	0.609***	0.592***	0.548**	0.582**	0.530**
	(0.173)	(0.173)	(0.194)	(0.178)	(0.183)
L1.ys	-0.711**	-0.566*	-0.666**	-0.624**	-0.379
	(0.232)	(0.261)	(0.221)	(0.216)	(0.223)
L2.ys	0.106	0.101	0.127	0.023	-0.208
	(0.141)	(0.161)	(0.156)	(0.151)	(0.152)
1979	0.010	0.011	0.024*	0.027*	0.031**
	(0.010)	(0.012)	(0.011)	(0.011)	(0.010)
1980	0.022	0.023	0.041*	0.047**	0.053**
	(0.018)	(0.020)	(0.020)	(0.018)	(0.018)
1981	-0.012	-0.021	0.002	0.018	0.026
	(0.030)	(0.033)	(0.034)	(0.030)	(0.030)
1982	-0.027	-0.031	0.018	0.022	0.034
	(0.029)	(0.034)	(0.023)	(0.021)	(0.023)
1983	-0.021	-0.018	0.043*	0.037*	0.041*
	(0.030)	(0.037)	(0.018)	(0.019)	(0.021)
1984	-0.008	-0.023	0.029	0.015	0.021
	(0.031)	(0.037)	(0.022)	(0.022)	(0.024)

(a, b) Equations in differences: $L\,(2/8)\,.n, D.w, L.D.w, D.k, L.D.k, L2.D.k, D.ys, L.D.ys, L2.D.ys, D.1979 - D.1984$

(c) Equations in differences: $L\,(2/8)\,.n, D.w, L.D.w, L2.D.w, D.k, L.D.k, L2.D.k, D.ys, L.D.ys, L2.D.ys, D.1979 - D.1984$

Equations in levels: $L\,(1/7)\,.D.n, w, L.w, L2.w, k, L.k, L2.k, ys, L.ys, L2.ys$

(d, e) Equations in differences: $L\,(2/8)\,.n, u, D.w, L.D.w, L2.D.w, D.k, L.D.k, L2.D.k, D.ys, L.D.ys, L2.D.ys, D.1979 - D.1984$

Equations in levels: $w, L.w, L2.w, k, L.k, L2.k, ys, L.ys, L2.ys$

* $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$ (refers to t-test of the null that the coefficient is equal to zero)

Table 2: Estimates in the spirit of Table 4 in Arellano and Bond (1991). Use of $L(a_1/a_2)$ indicates lag transformation by a minimum of a_1 and a maximum of a_2 time periods; D indicates first differences. Table footnotes indicate available instruments and corresponding equations.

The command `mtest.fct(m2,t.order = 2)` is used to perform the test of Arellano and Bond (1991) for second order serial correlation and yields:

```
        Arellano and Bond (1991) serial correlation test of degree 2

data:  2step GMM Estimation; H0: no serial correlation of order 2 in the error terms
normal = -0.37133, p-value = 0.7104
```

The test does not reject the null hypothesis at any plausible significance level and provides no indication that the model specification might be inadequate.

Computing the Hansen *J*-test of overidentifying restrictions by `jtest.fct(m2)` yields:

```
        J-Test of Hansen

data:  2step GMM Estimation; H0: overidentifying restrictions valid
chisq = 31.381, df = 25, p-value = 0.1767
```

As the test does not reject the null hypothesis, there are no indications that the validity of the instruments (i.e., the model assumptions) employed in estimation may be in doubt.

For the Wald test of the null hypothesis that the population parameters of all coefficients included in the model are jointly zero, which is tested by `wald.fct(m2,param = "all")`, we obtain:

```
        Wald test

data:  2step GMM Estimation; H0: all parameters are jointly zero
chisq = 1100, df = 16, p-value < 2.2e-16
```

The test rejects the null hypothesis. Hence, all tests shown here provide no indications that the model in column (b) of Table 2 is misspecified.

Comparing the results to xtabond2 shows that degrees of freedom and *p*-values differ for the latter two tests. We consider 25 degrees of freedom to be the appropriate number in the J-test, as 41 instruments are employed in estimation to obtain 16 coefficient estimates. The latter number (16) is the appropriate number of degrees of freedom in the Wald test. It seems that the function xtabond2 does not correct the degrees of freedom for the number of dummies dropped in estimation[1]. The difference in the *p*-value is due to the differences in the degrees of freedom. Our results are equivalent to the results of `pgmm` for the overidentifying restrictions test (referred to as "Sargan test" in `pgmm`).

Using many instruments may have undesirable side effects such as biased coefficient estimates and standard errors. This may result in misleading inference and specification tests (see, e.g., Roodman, 2009). The number of lags of the dependent variable which are used to derive moment conditions can be limited by setting `maxLags.y` (equivalently lags of, for example, endogenous covariates can be limited via `maxLags.reg.end`). Setting `maxLags.y = 4` reduces the number of HNR moment conditions for the GMM estimation above from 27 to 17 and the total number of instruments employed in the estimation from 41 to 31.

GMM estimation with HNR and ABov moment conditions

When the "constant correlated effects" assumption stated in Equation (9) holds, the HNR moment conditions from equations in differences employed in Section GMM estimation with HNR moment conditions can be extended by the ABov moment conditions from equations in levels.

The ABov moment conditions are particularly useful for data generating processes, which are highly persistent (Blundell and Bond, 1998). In this case, identification by the HNR moment conditions from equations in differences may fail, and GMM estimation based on HNR moment conditions is documented to possess poor finite sample performance (see, e.g., Blundell and Bond, 1998; Blundell et al., 2001; Bun and Sarafidis, 2015).

In **pdynmc**, the ABov moment conditions from equations in levels can be (additionally) incorporated by:

```
use.mc.lev = TRUE
```

In principle, both time dummies and further covariates can be included in the equations in first differences and the level equations. It is recommended, though, to include the dummies only in one of the equations, as it can be shown that incorporating them in both equations renders one set of

[1]Dummies are dropped by the estimation routine in case of high collinearity.

dummies redundant for estimation – while for non-lagged dependent covariates, this equivalence does not hold.[2] We accommodate non-lagged dependent covariates in the levels equations by:

```
fur.con.lev = TRUE
```

for all subsequent estimations of this example. The results presented in column (c) of Table 2 are two-step estimates of column (a2) of Table 4 in Arellano and Bond (1991) extended by ABov moment conditions.

Including ABov moment conditions into the analysis leads to substantial changes in the coefficient estimates of the first lag of the dependent variable. Note that the results indicate a markedly higher persistence of employment and render including two lags of the dependent variable questionable (Blundell and Bond, 1998, e.g., estimate a version of the equation which contains only one lag of all covariates). Note that the coefficient estimates of the covariates, besides the first lag of the dependent variable, appear to be similar across estimations.

Equivalent results to column (c) of Table 2 can be obtained from the `pgmm` function in the **plm**-package. When replicating the results with xtabond2, `inst.stata = TRUE` in **pdynmc** ensures that results are equivalent.

GMM estimation with HNR and AS moment conditions

Recall that the linear ABov moment conditions from equations in levels encompass the nonlinear AS moment conditions (Blundell and Bond, 1998; a derivation is provided in Fritsch, 2019). Both sets of moment conditions may be useful in GMM estimation when the lag parameter is close to unity, and it can be shown that extending the HNR moment conditions by either the ABov or the AS moment conditions may identify the lag parameter – even when individual moment conditions fail to do so (Blundell and Bond, 1998; Bun and Kleibergen, 2021; Gørgens et al., 2019). The ABov moment conditions require the "constant correlated effects" assumption to hold, while the AS moment conditions only require standard assumptions to hold. Therefore, the latter may be useful in situations where the "constant correlated effects" assumption is in doubt, and the statistician aims to investigate a highly persistent dynamic process with a structure similar to Equation (3). In **pdynmc**, including nonlinear moment conditions into the analysis is available via:

```
use.mc.nonlin = TRUE
```

When extending the analysis of Arellano and Bond (1991) by the nonlinear AS moment conditions, the results differ substantially from column (b) of Table 2 and are very similar to the coefficient estimates shown in column (c) of Table 2. This indicates high persistence in the employment process that leads to lag parameters not being identified by the HNR moment conditions (Bun and Kleibergen, 2021; Gørgens et al., 2019).

Additionally, we employ iterated GMM via:

```
estimation = "iterative", max.iter = 100, iter.tol = 0.01,
```

Iterated GMM results are shown in column (e) of Table 2. The moment conditions employed are the same as in column (d) of the table. The parameter estimates obtained after 13 steps are relatively similar to those in columns (c)-(d). The ranges of the coefficient estimates across GMM iterations are displayed in Figure 2. This plot is available for two-step and iterated GMM estimates via command `plot(m5,type = "coef.range",omit1step = TRUE)`. Using command `plot(m5)` yields a scatterplot of fitted values and residuals of a fitted model object, instead.

Figure 2 indicates coefficient estimates at iteration 2 as grey open circles (the first iteration is ignored due to `omit1step = TRUE`) and estimates at the last iteration as blue diamonds. For the estimates displayed in column (e) of Table 2, we observe that the lag parameters are relatively stable across iterations and resemble the two-step estimates; for the further covariates, larger changes in coefficient estimates across iterations occur for coefficients with larger standard errors (compare w, k, and ys).

As an additional tool to investigate coefficient estimates from iterated GMM, coefficient path plots (compare Hansen and Lee, 2021, Figure 1) are provided. Figure 3 illustrates the path of coefficient estimates for lag parameter α_1 across GMM iterations and is obtained via `plot(m5,type = "coef.path",co = "L1.n")`. Argument `co` allows to draw the path(s) of particular coefficient estimates; per default, all coefficients (apart from time dummies) are included in the plot. Approximate 95% confidence bands were added to the plot for the final iteration (available by setting argument `add.se.approx = TRUE`).

[2] Note that this is the case in balanced panels. The results may also not be numerically identical across function calls for different choices of the one-step weighting matrix. For a discussion, see https://www.statalist.org/forums/forum/general-stata-discussion/general/1357268-system-gmm-time-dummies.

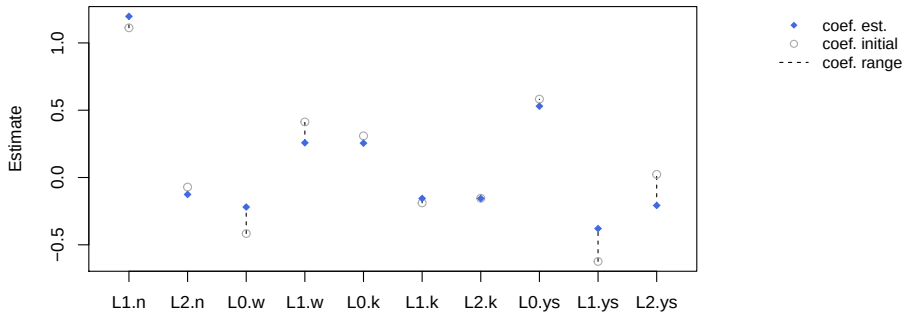


Figure 2: Estimated coefficients and corresponding coefficient ranges during iterated GMM estimation (ordinate) for the covariates (dummy variables excluded). Coefficient estimates at the initial step are displayed as grey open circles and the estimates at the last step as blue diamonds.

Figure 3: Estimated coefficient path for lag parameter α_1 during iterated GMM estimation and corresponding approximate 95% confidence bands for final iteration.

Overall, the results displayed in columns (c)-(e) of Table 2 suggest that the employment process may be highly persistent and that using only HNR moment conditions may not be sufficient to identify the parameters. In practice, contrasting GMM estimates based on HNR and AS moment conditions with GMM estimates based on HNR and ABov moment conditions can be used as a robustness check of the "constant correlated effects" assumption: When estimates differ, this may cast doubt on the assumption. Here, this is not the case as the results in column (c) are very close to those in (d) and (e).

Conclusion

R-package **pdynmc** provides a function to estimate linear dynamic panel data models based on linear and nonlinear moment conditions. The implementation reflects recent developments in the literature by including iterated GMM and offers a wide variety of configuration options. The package provides the only open source solution for GMM estimation of dynamic panels with linear and nonlinear moment conditions, aligns commercial and non-commercial software, and is implemented to enable the user to exert precise control over all functionality. Additionally, suitable visualizations of panel data structures and ranges and paths of coefficient estimates across GMM iterations are provided.

The functionality of **pdynmc** includes that it allows for general lag structures of the covariates; further controls and external instruments (if available) may also be added. The estimation routine can handle balanced and unbalanced panel datasets and provides one-step-, two-step-, and iterated estimation. Accounting for (unobserved) time-specific effects is possible by including time dummies. Estimation relies on numerical optimization of the GMM objective function. Corresponding closed-form solutions are computed – where possible – and stored beside the results from numerical

optimization. Different choices for the weighting matrix, which guides the aggregation of moment conditions in one-step GMM estimation are available. Robust standard errors are available for inference and specification testing. Nonlinear moment conditions provide a robustness check of the frequently employed constant correlated effects assumption.

Acknowledgements

We thank the executive editor and anonymous reviewers for their helpful comments and suggestions. We also thank Harry Haupt for helpful discussions, comments, and suggestions. Andrew Adrian Pua is supported by the Fundamental Research Funds for the Central Universities (20720171074), the Basic Scientific Center Project 71988101 of National Science Foundation of China, and the 111 Project (B13028).

Bibliography

S. C. Ahn and P. Schmidt. Efficient estimation of models for dynamic panel data. *Journal of Econometrics*, 68(1):5–27, 1995. URL https://doi.org/10.1016/0304-4076(94)01641-C. [p218, 219, 220]

T. Anderson and C. Hsiao. Formulation and estimation of dynamic models using panel data. *Journal of Econometrics*, 18(1):47–82, 1982. URL https://doi.org/10.1016/0304-4076(82)90095-1. [p219]

M. Arellano. *Panel Data Econometrics*. Oxford University Press, 2003. URL https://doi.org/10.1093/0199245282.001.0001. [p220, 221]

M. Arellano and S. Bond. Some Tests of Specification for Panel Data: Monte Carlo Evidence and an Application to Employment Equations. *The Review of Economic Studies*, 58(2):277–297, 1991. URL https://doi.org/10.2307/2297968. [p218, 219, 221, 222, 223, 224, 225, 226, 227]

M. Arellano and O. Bover. Another look at the instrumental variable estimation of error-components models. *Journal of Econometrics*, 68(1):29–51, 1995. URL https://doi.org/10.1016/0304-4076(94)01642-D. [p219, 220]

R. Blundell and S. Bond. Initial conditions and moment restrictions in dynamic panel data models. *Journal of Econometrics*, 87(1):115–143, 1998. URL http://doi.org/10.1016/S0304-4076(98)00009-8. [p226, 227]

R. Blundell, S. Bond, and F. Windmeijer. Estimation in dynamic panel data models: Improving on the performance of the standard GMM estimator. In B. H. Baltagi, T. B. Fomby, and R. C. Hill, editors, *Nonstationary Panels, Panel Cointegration, and Dynamic Panels*, volume 15 of *Advances in Econometrics*, pages 53–91. Emerald Group Publishing, 2001. URL https://doi.org/10.1016/S0731-9053(00)15003-0. [p221, 226]

M. J. G. Bun and F. Kleibergen. Identification robust inference for moments based analysis of linear dynamic panel data models. *Econometric Theory*, forthcoming, 2021. URL https://doi.org/10.2139/ssrn.3847636. [p219, 227]

M. J. G. Bun and V. Sarafidis. Chapter 3 – Dynamic panel data models. In B. H. Baltagi, editor, *The Oxford Handbook of Panel Data*, pages 76–110. Oxford University Press, 2015. URL https://doi.org/10.1093/oxfordhb/9780199940042.013.0003. [p219, 220, 226]

Y. Croissant and G. Millo. *Panel Data Econometrics with R: the plm package*. Wiley, 2019. [p218, 219]

Y. Croissant, G. Millo, and K. Tappe. plm: Linear models for panel data, 2020. URL https://CRAN.R-project.org/package=plm. [p223]

J. A. Doornik, M. Arellano, and S. Bond. *Panel Data estimation using DPD for Ox*, 2012. URL http://www.doornik.com/download.html. [p221]

R. Fletcher. A new approach to variable metric algorithms. *The Computer Journal*, 13(3):317–322, 1970. URL https://doi.org/10.1093/comjnl/13.3.317. [p221]

M. Fritsch. On GMM estimation of linear dynamic panel data models. University of Passau Working Papers in Business Administration B-36-19, University of Passau, 2019. URL https://www.wiwi.uni-passau.de/en/research/working-papers/. [p219, 220, 223, 227]

M. Fritsch, A. A. Y. Pua, and J. Schnurbus. pdynmc – An R-package for estimating linear dynamic panel data models based on nonlinear moment conditions, 2020. URL https://cran.r-project.org/package=pdynmc. [p218, 221]

T. Gørgens, C. Han, and S. Xue. Moment Restrictions and Identification in Linear Dynamic Panel Data Models. *Annals of Economics and Statistics*, (134):149–176, 2019. URL https://doi.org/10.15609/annaeconstat2009.134.0149. [p219, 227]

B. E. Hansen and S. Lee. Inference for Iterated GMM Under Misspecification. *Econometrica*, 89(3): 1419–1447, 2021. URL https://doi.org/10.3982/ECTA16274. [p218, 221, 227]

L. P. Hansen. Large Sample Properties of Generalized Method of Moments Estimators. *Econometrica*, 50(4):1029–1054, 1982. URL https://doi.org/10.2307/1912775. [p221]

D. Holtz-Eakin, K. Newey, Whitney, and H. S. Rosen. Estimating Vector Autoregressions with Panel Data. *Econometrica*, 56(6):1371–1395, 1988. URL https://doi.org/10.2307/1913103. [p219, 220]

C. Hsiao. *Analysis of Panel Data*. Cambridge University Press, 3 edition, 2014. URL https://doi.org/10.1017/CBO9781139839327. [p219]

J. Hwang and Y. Sun. Should we go one step further? An accurate comparison of one-step and two-step procedures in a generalized method of moments framework. *Journal of Econometrics*, 207 (2):381–405, 2018. URL https://doi.org/10.1016/j.jeconom.2018.07.006. [p218]

J. Hwang, B. Kang, and S. Lee. A doubly corrected robust variance estimator for linear GMM. *Journal of Econometrics*, forthcoming, 2021. URL https://doi.org/10.1016/j.jeconom.2020.09.010. [p221]

J. F. Kiviet. Chapter 11 – Judging Contending Estimators by Simulation: Tournaments in Dynamic Panel Data Models. In G. D. A. Phillips and E. Tzavalis, editors, *The Refinement of Econometric Estimation and Test Procedures: Finite Sample and Asymptotic Analysis*, pages 282–318. Cambridge University Press, 2007. URL https://doi.org/10.1017/CBO9780511493157. [p220]

S. Kripfganz. XTDPDGMM: Stata module to perform generalized method of moments estimation of linear dynamic panel data models, 2019. URL http://EconPapers.repec.org/RePEc:boc:bocode:s458395. [p218, 221, 223]

J. C. Nash. *Compact Numerical Methods for Computers: Linear Algebra and Function Minimisation*. CRC Press, 1990. [p221]

J. C. Nash. On Best Practice Optimization Methods in R. *Journal of Statistical Software*, 60(2):1–14, 2014. URL https://doi.org/10.18637/jss.v060.i02. [p221]

J. C. Nash. Provenance of R's Gradient Optimizers. *The R Journal*, 12(1):477–483, 2020. URL https://doi.org/10.32614/RJ-2020-029. [p221]

J. C. Nash and R. Varadhan. Unifying optimization algorithms to aid software system users: optimx for R. *Journal of Statistical Software*, 43(9):1–14, 2011. URL https://doi.org/10.18637/jss.v043.i09. [p221]

M. Pickup, P. Gustafson, D. Cubranic, and G. Evans. OrthoPanels: An R Package for Estimating a Dynamic Panel Model with Fixed Effects Using the Orthogonal Reparameterization Approach. *The R Journal*, 9(1):60–76, 2017. URL https://doi.org/10.32614/RJ-2017-003. [p218]

A. Pua, M. Fritsch, and J. Schnurbus. Large sample properties of an IV estimator based on the Ahn and Schmidt moment conditions. University of Passau Working Papers in Business Administration B-37-19, University of Passau, 2019a. URL https://www.wiwi.uni-passau.de/en/research/working-papers/. [p219]

A. Pua, M. Fritsch, and J. Schnurbus. Practical aspects of using quadratic moment conditions in linear AR(1) panel data models. University of Passau Working Papers in Business Administration B-38-19, University of Passau, 2019b. URL https://www.wiwi.uni-passau.de/en/research/working-papers/. [p219]

D. Roodman. A Note on the Theme of Too Many Instruments. *Oxford Bulletin of Economics and Statistics*, 71(1):135–158, 2009. URL https://doi.org/10.1111/j.1468-0084.2008.00542.x. [p226]

D. Roodman. XTABOND2: Stata module to extend xtabond dynamic panel data estimator, 2018. URL https://econpapers.repec.org/software/bocbocode/s435901.htm. [p223]

M. Sigmund and R. Ferstl. Panel vector autoregression in R with the package panelvar. *The Quarterly Review of Economics and Finance*, page forthcoming, 2019. URL https://doi.org/10.1016/j.qref.2019.01.001. [p218]

StataCorp. *Stata Longitudinal-Data/Panel-Data Reference Manual Release 14*. StataCorp LP, College Station, Texas, 2015. [p218]

F. Windmeijer. A finite sample correction for the variance of linear efficient two-step gmm estimators. *Journal of Econometrics*, 126(1):25–51, 2005. URL https://doi.org/10.1016/j.jeconom.2004.02.005. [p221, 222]

Markus Fritsch
University of Passau
School of Business, Economics, and Information Systems
94032 Passau
Germany
markus.fritsch@uni-passau.de

Andrew Adrian Yu Pua
MOE Key Laboratory of Econometrics
The Wang Yanan Institute for Studies in Economics
Department of Statistics, School of Economics
Fujian Key Laboratory of Statistics
Xiamen University
China
andrewypua@outlook.com

Joachim Schnurbus
University of Passau
School of Business, Economics, and Information Systems
94032 Passau
Germany
joachim.schnurbus@uni-passau.de

DChaos: An R Package for Chaotic Time Series Analysis

by Julio E. Sandubete and Lorenzo Escot

Abstract Chaos theory has been hailed as a revolution of thoughts and attracting ever-increasing attention of many scientists from diverse disciplines. Chaotic systems are non-linear deterministic dynamic systems which can behave like an erratic and apparently random motion. A relevant field inside chaos theory is the detection of chaotic behavior from empirical time-series data. One of the main features of chaos is the well-known initial-value sensitivity property. Methods and techniques related to testing the hypothesis of chaos try to quantify the initial-value sensitive property estimating the so-called Lyapunov exponents. This paper describes the main estimation methods of the Lyapunov exponent from time series data. At the same time, we present the **DChaos** library. R users may compute the delayed-coordinate embedding vector from time series data, estimates the best-fitted neural net model from the delayed-coordinate embedding vectors, calculates analytically the partial derivatives from the chosen neural nets model. They can also obtain the neural net estimator of the Lyapunov exponent from the partial derivatives computed previously by two different procedures and four ways of subsampling by blocks. To sum up, the **DChaos** package allows the R users to test robustly the hypothesis of chaos in order to know if the data-generating process behind time series behaves chaotically or not. The package's functionality is illustrated by examples.

Introduction

According to the literature, countless techniques have been developed and used to estimate the complexity of time series data; for a review see, e.g., Faggini (2014), Bradley and Kantz (2015), Tang et al. (2015). We have focused on methods derived from chaos theory which estimates the complexity of a dataset through exploring the structure of the attractor. Particularly, we have been interested in the so-called *Lyapunov exponent* (λ) as an attractor invariant measure. Quantifing chaos through this kind of quantitative measure is a key point for understanding chaotic behavior. Hence, our interest will be to test the hypothesis of chaos defined as follows:

$$\begin{array}{l} H_0 : \hat{\lambda}_k > 0 \\ H_1 : \hat{\lambda}_k \leqslant 0, \end{array} \tag{1}$$

for $k = 1, 2, 3, \ldots$ on a k-dimensional system. Reject the null hypothesis $H_0 : \hat{\lambda}_k > 0$ means lack of chaotic behavior. That is, the data-generating process does not have a chaotic attractor because it does not show the property of sensitivity to initial conditions, see Gençay and Dechert (1992). The existence of a positive Lyapunov exponent makes it possible to distinguish whether an apparently erratic, non-cyclical, and aperiodic dynamic system is random or chaotic. In this sense, a positive Lyapunov exponent allows us to evidence that a deterministic generating system exists behind that chaotic system in spite of showing an apparently random dynamic behavior. This fact would provide us to take advantage of this deterministic character to be able to seek modeling of time series using non-linear dynamic models, make reliable predictions, at least within the limits established by the sensitivity to the initial conditions, and control over the variables of these chaotic deterministic dynamic systems, see Fernández-Díaz (2019).

This ergodic measure can be defined as follows. Let $X_t = f(X_{t-1})$ be a difference equation where $f : \mathbb{R}^k \to \mathbb{R}^k$ for $t = 1, 2, 3, \ldots, n$. For a k-dimensional system, there will be k Lyapunov exponents which are given by

$$\begin{aligned} \lambda_k(X_0) &= \lim_{t\to\infty} \frac{1}{t} \{\log(|Df(X_t)| \cdot \ldots \cdot |Df(X_0)|)\} \\ &= \lim_{t\to\infty} \frac{1}{t} \{\log(|Df^t(X_0)|)\} \end{aligned} \tag{2}$$

This relation indicates the average rate of divergence or convergence of an orbit $f^t(X_0)$ starting at point X_0, where $Df^t(X_0)$ is the Jacobian evaluated along the trajectory $\{X_0, X_1, \ldots, X_t\}$. If one knows the data-generating process behind the time series, the *theoretical Lyapunov exponent* can be calculated directly using its own definition as outlined in eq.2. However, we have assumed that the true dynamics of the system is unknown because in most real-world observed time series, the data-generating process is rarely known a priori.

For this reason, we will not take into consideration estimation methods that presuppose the knowledge of the equations of the dynamical system. There are basically two kinds of methods in order to compute the *estimated Lyapunov exponent* from time-series data. The first one, the so-called *direct* approach, which directly measures the growth rate of the divergence between two neighboring trajectories with an infinitesimal difference in their initial conditions. The second one, the so-called *indirect* approach (or Jacobian-based method), which first fits a model to the data based on approximations of the trajectories in the reconstructed state space, and then the Jacobian matrices of the estimated dynamic system are used to compute the Lyapunov exponent. We will discuss both methods, although we have focused in greater detail on the Jacobian indirect methods for the reasons set out later.

There are some R packages recently developed related to nonlinear time series analysis and chaos. For instance, the **tseriesChaos** written by Narzo (2019), **nonlinearTseries** proposed by Garcia (2019) and **fNonlinear** provided by Wuertz et al. (2017). These R packages are based on ideas inspired by the time series analysis (TISEAN) project suggested by Hegger et al. (1999). All of them implement the algorithm written by Kantz (1994) using exclusively the direct method for estimating the Lyapunov exponent. The main drawbacks of the direct methods are the following: (i) it does not allow the estimation of the full spectrum of Lyapunov exponents; (ii) it is not robust to the presence of (small) measurement noise because these estimators can produce a linear scaling region giving a wrong chaotic Lyapunov exponent value even for non-chaotic systems; (iii) it does not have a satisfactory performance in detecting existing non-linearities on time-series data of moderate sample sizes; (iv) the asymptotic distribution of the estimator cannot be derived, means that it does not allow the building of formal tests.

Function	Description
`embedding`	Provides the delayed-coordinate embedding vectors backwards
`gauss.sim`	Simulates time series data from the Gauss map
`henon.sim`	Simulates time series data from the Hénon system
`jacobian.net`	Computes the partial derivatives from the best-fitted neural net model
`logistic.sim`	Simulates time series data from the Logistic map
`lyapunov`	Estimates the Lyapunov exponent through several methods
`lyapunov.max`	Estimates the largest Lyapunov exponent by the Norma-2 procedure
`lyapunov.spec`	Estimates the Lyapunov exponent spectrum by the QR decomposition
`netfit`	Fits any standard feed-forward neural net model from time series data
`rossler.sim`	Simulates time series data from the Rössler system
`summary.lyapunov`	Summary method for a lyapunov object
`w0.net`	Estimates the initial parameter vector of the neural net model

Table 1: A summary of the functions available in the **DChaos** package

In this paper, we present the **DChaos** package written by Sandubete and Escot (2021), which provides the following contributions. First, as far as we know, the **DChaos** package is the first R library that provides the Jacobian indirect methods proposed by Eckmann and Ruelle (1985) and Gençay and Dechert (1992) for estimating the Lyapunov exponents. These Jacobian indirect methods solve all drawbacks which belong to the direct methods. Particularly, this package has focused on the neural net approach following McCaffrey et al. (1992) and Nychka et al. (1992) by approximating the unknown non-linear system through a feed-forward single hidden layer neural network. Those neural net methods provide a well-fit of any unknown linear or non-linear model. They also have the advantage to allow the analytical derivation, rather than numerically, the jacobian needed for the estimation of the Lyapunov exponents.

Second, the **DChaos** package allows the use of full sample estimation, but it also provides three different blocking methods for estimating the Lyapunov exponents following McCaffrey et al. (1992) and Shintani and Linton (2004). One of them is a new proposal based on the bootstrap method. The results provided by the three blocking methods improve the case of the full sample being very similar between them. Although, on average, the bootstrap method gives better results.

Third, the **DChaos** package provides new algorithms implementing the formal test proposed by Shintani and Linton (2004), who provided a statistical framework for testing the hypothesis of chaos, based on the theoretical asymptotic properties from the neural net estimator and its variance. R users might make statistical inferences about Lyapunov exponents' significance and test the hypothesis of chaos (eq.1). Remember that the asymptotic distribution of the estimator obtained from direct methods does not exist, and that is, there is no chance to make statistical inference about chaos.

Fourth, any methods for estimating the Lyapunov exponent from time series data are previously based on the state space reconstruction procedure. Note that a key point to create a suitable reconstruction of the state-space is to fix criteria in order to estimate the embedding parameters. Researchers usually estimate them using heuristic approaches based on prescriptions proposed by, e.g., Abarbanel (1996) or Kantz and Schreiber (2004). The main drawbacks of these heuristic approaches are the following: they are not intrinsically statistical; their results are not robust; they lead to estimators whose properties are unknown or largely unexplored; they do not take into account the results of any model fit. Although the **Dchaos** package also allows the use of heuristic methods, we have implemented some alternative and consistent statistical methods based on model selection criteria following Chan and Tong (2001) which solves those disadvantages.

Fifth, instead of considering only time-series data with uniform time-frequency, e.g., 1-month, 1-day, 1-hour, 30-min, 5-min, 1-min, and so on, as R packages mentioned above, the **DChaos** package also allows the use of time series data with non-uniform time-frequency. As far as we know, the **DChaos** package is the first R library that provides the reconstruction of the pseudo-estate space from time series data with non-uniform time-frequency (which are not equally spaced in time). In this sense, this package provides a new algorithm implementing the theorem proposed by Huke and Broomhead (2007), who extended the reconstruction theorem when the dynamical system is sampled non-uniformly in time. Their results have allowed us to get the non-uniform delayed-coordinate embedding vectors from time series data with non-uniform time-frequency, e.g., tick-by-tick financial time series.

To sum up, the **DChaos** package provides a novel R interface for researchers interested in the field of chaotic dynamic systems and non-linear time series analysis and professors (and students) who teach (learn) courses related to those topics. There are 12 functions available in the **DChaos** package; see table 1. We are going to describe all of them as we go on to explain the theoretical procedure. The rest of the paper is organized as follows. Section 2 presents the state space reconstruction procedure from time series data. Section 3 gives an overview about the main estimation methods of the Lyapunov exponents and provides some technical details about the **DChaos** package. Section 4 illustrates some examples showing the package's functionality. Section 5 contains some concluding remarks.

State space reconstruction from time series data

From now on, we assume that the true data-generating process is unknown. Hence, we do not have the advantage of observing directly the state of the system X_t, let alone knowing the functional form f that generates the dynamic associated with it. Instead of that, there is an observer function that includes an additive measurement error $\psi : \mathbb{R}^{k+1} \to \mathbb{R}$ which generates observations as $x_t = \psi(X_t, \varepsilon_t)$, where ε_t is a sequence of independent and identically distributed random variables such that ε_t is independent of X_j, $0 \leqslant j \leqslant t$, for $t = 1, 2, 3, \ldots, n$. Therefore, it is assumed that all information available is the noise-contaminated sequence $\{x_t\}_{t=1}^{n}$ as a time series data. We have considered it appropriate to add a measurement noise term in the observation function because most real-world, observed time series data are usually noise-contaminated signals.

The embedding procedure allows us to get all the relevant information (invariant properties) about the unknown underlying dynamical system that generates the time series data, e.g., the Lyapunov exponents defined previously must have approximately the same value in both the true and the reconstructed state space. This fact allows us to test the hypothesis of chaos in the unknown original dynamic system (1). Any methods for estimating the Lyapunov exponent from some observed time series data are based previously on the state space reconstruction procedure. The embedding theorem proposed by Takens (1981) provides a framework to reconstruct an unknown dynamical system which gave rise to a given observed scalar time series simply by reconstructing a new state space out of successive values of the time series. We have considered the method of delayed-coordinates proposed by Ruelle and Takens (1971) to get the delayed-coordinate embedding vectors as follows. Let $\{x_t\}_{t=1}^{n}$ be the time series data. We form a sequence of delayed vectors by associating for each time period a vector in a reconstructed state space $\mathbb{R}^m$, whose coordinates satisfy the following equation:

$$x_t^m = \left(x_t, x_{t-\tau}, x_{t-2\tau}, \ldots, x_{t-(m-2)\tau}, x_{t-(m-1)\tau}\right), \tag{3}$$

where m is the *embedding dimension* and τ is the *reconstruction time-delay* (or lag).

We can construct a vector space whose axes represent all the relevant variables given by

$$
\begin{aligned}
x_1^m &= \left(x_1, x_{1-\tau}, \ldots, x_{1-(m-2)\tau}, x_{1-(m-1)\tau}\right) \\
x_2^m &= \left(x_2, x_{2-\tau}, \ldots, x_{2-(m-2)\tau}, x_{2-(m-1)\tau}\right) \\
&\vdots \\
x_n^m &= \left(x_n, x_{n-\tau}, \ldots, x_{n-(m-2)\tau}, x_{n-(m-1)\tau}\right)
\end{aligned}
$$

The underlying idea is to make copies of the measured signal with uniform time-lapse between observations and consider these delayed values as coordinates of a reconstructed state space, retrieved from the data. Notice that the reconstruction theorem assumes that the dynamical system is sampled uniformly in time. There has, however, been increasing interest in situations where observations are not uniform in time. That is, the data does not come from a series of values measured periodically in time. For instance, a motivating example would be the case of financial markets. The information generated by the interactions between traders who buy (bid orders) and sell (ask orders) financial instruments such as stocks, bonds, commodities, currencies, derivatives, and so on is apparently encoded in frequencies which are not usually equally spaced in time. Thus, if we observe a particular financial asset, i.e., a currency pair on the Foreign Exchange Market, the quotes or rates of this financial asset are sampling, by tick-by-tick intervals, which do not follow a constant rhythm. Each tick will appear when there is a change, upward or downward, in the trade price of each transaction. Can we use this kind of information to build a dynamical system model, and if so, how is it related to the true data-generating process? Huke and Broomhead (2007) showed how to extend the reconstruction theorem under certain conditions when the dynamical system is sampled non-uniformly in time. So, we can use their results to get the non-uniform delayed-coordinate embedding vectors from time series data with non-uniform time-frequency.

The three R packages mentioned before have implemented their algorithms considering only the state space reconstruction from time series data sampled uniformly in time. The **fNonlinear** package has a function called `embeddPSR`. `buildTakens` is the function that belongs to the **nonlinearTseries** package, and the **tseriesChaos** package includes the function `embedd`. Note that such packages consider the delayed-coordinate embedding vectors forward while we consider them backward. Both ways of reconstructing the attractor are correct, but we think it more convenient to define the delayed-coordinate embedding vectors backward since in time series analysis we explain the initial and future instants from what has happened in the past as in equation 3. Let us show some brief examples of how to deal with the state space reconstruction procedure from time series data.

Uniform delayed-coordinate embedding vectors. On the one hand, the first five values are shown using the `embedd` function, which belongs to **tseriesChaos** package for an embedding dimension `m=5` and a time delay equal to `d=2` . We have simulated time series data from the logistic equation contained in the **DChaos** package. The command `set.seed(34)` will set the seed for reproducibility.

```
## Simulates time-series data from the Logistic map with chaos
ts    <- DChaos::logistic.sim(a=4, n=1000)
[1] 7.47e-01 7.57e-01 7.37e-01 7.76e-01 6.95e-01 8.48e-01 5.16e-01 9.99e-01 4.22e-03

## Provides the uniform delayed-coordinate embedding vectors forwards
data  <- tseriesChaos::embedd(ts, m=5, d=2)
show(head(data, 5))

     V1/0      V1/2      V1/4       V1/6       V1/8
[1,] 0.7466701 0.7365940 0.69509054 0.51625546 0.00422337
[2,] 0.7566155 0.7760930 0.84775873 0.99894304 0.01682213
[3,] 0.7365940 0.6950905 0.51625546 0.00422337 0.06615659
[4,] 0.7760930 0.8477587 0.99894304 0.01682213 0.24711960
[5,] 0.6950905 0.5162555 0.00422337 0.06615659 0.74420600
```

On the other hand, the **DChaos** package provides the function `embedding` which arguments are (`x`, `m`, `lag`, `timelapse`) to get the delayed-coordinate embedding vectors backward considering both the uniform and the non-uniform case. If the observations are sampled at uniform time intervals, then `timelapse=FIXED`. Otherwise, `timelapse=VARIABLE` has to be specified. Note that if `FIXED` has been selected, `data` must be a `vector` or a time series object `ts` or `xts`. Otherwise, `VARIABLE` has to be specified. In this case, `data` must be a `data.frame`, a `data.table`, or a `matrix` with two columns, the date and the univariate time series as a sequence of numerical values, in that order. The date can have the following three classes: `POSIXt`, `Date`, or `Factor`.

In the latter case, the date should come in the following format `YMD H:M:OS3` considering milliseconds, e.g., 20190407 00:00:03.347. If the R users do not consider milliseconds, they must put .000 after the seconds. The function `embedding` returns the uniform or non-uniform delayed-coordinate embedding vectors backward by columns from univariate time series considering the parameter set selected by the R users. Let us show an example. Firstly, the first five values corresponding to the uniform embedding vectors set for `m=5`, `lag=2`, and `timelapse=FIXED` are shown considering the time series previously simulated from the logistic map. As we said before, we consider the delayed-coordinate embedding vectors backward while other R packages (i.e., **tseriesChaos**) consider them forward.

```
## Provides the uniform delayed-coordinate embedding vectors backwards
data  <- DChaos::embedding(ts, m=5, lag=2, timelapse="FIXED")
show(head(data, 5))
```

```
           y         x1         x2        x3        x4
1 0.00422337 0.51625546 0.69509054 0.7365940 0.7466701
2 0.01682213 0.99894304 0.84775873 0.7760930 0.7566155
3 0.06615659 0.00422337 0.51625546 0.6950905 0.7365940
4 0.24711960 0.01682213 0.99894304 0.8477587 0.7760930
5 0.74420600 0.06615659 0.00422337 0.5162555 0.6950905
```

Non-uniform delayed-coordinate embedding vectors. We have used a data set called `sbux`, which belongs to the **highfrequency** package. It contains the bid quote price data for Starbucks company. This R package provides several tools for high-frequency time series data analysis. The first five values corresponding to the non-uniform embedding vectors set for `m = 3`, `lag = 4`, and `timelapse=VARIABLE` are shown. We can get the non-uniform delayed-coordinate embedding as follows. Let $\{x_{t_i}\}_{i=1}^{n}$ be our time series taking $t_i - t_{i-1} \neq t_s - t_{s-1}\ \forall i \neq s$. We form a sequence of delayed vectors by associating with each time a vector in a reconstructed state space $\mathbb{R}^m$, whose coordinates satisfy the following equation:

$$x_{t_i}^m = \left(x_{t_i}, x_{t_i-\tau}, x_{t_i-2\tau}, \ldots, x_{t_i-(m-2)\tau}, x_{t_i-(m-1)\tau}\right), \tag{4}$$

where m is the *embedding dimension* and τ is the *reconstruction delay* (or lag). As in the uniform case (eq.3), we can construct a vector space whose axes represent all the relevant variables.

```
## Simulates tick-by-tick data from bid quote price for Starbucks company
ts2    <- highfrequency::sbux

## Provides the non-uniform delayed-coordinate embedding vectors backwards
data  <- DChaos::embedding(ts2, m=3, lag=4, timelapse="VARIABLE")
show(head(data, 5))
```

```
                                y            x1            x2
2010-07-01 15:30:09 -0.0012327924 -0.0010255359  0.0008179960
2010-07-01 15:30:11  0.0000000000 -0.0002053177 -0.0004090816
2010-07-01 15:30:12 -0.0004112688 -0.0002053177 -0.0020479221
2010-07-01 15:30:15  0.0004112688  0.0000000000 -0.0002053177
2010-07-01 15:30:16 -0.0004112688 -0.0004112688 -0.0002053177
```

Note that a key point to create a suitable reconstruction of the state space is to fix criteria in order to estimate the embedding parameters (τ and m). Researchers in this area usually estimate them using two different alternatives: a heuristic approach that mostly relies on physical or geometrical arguments and by a statistical approach. Under the heuristic approach regarding the estimation of the time delay τ, although there are other criteria, see, e.g.Abarbanel (1996), Kantz and Schreiber (2004). $\tau = 1$ is commonly used following the prescription proposed by Takens (1981). Concerning the embedding dimension m, most of the papers published consider the false nearest neighbors criteria proposed by Kennel et al. (1992). Another criteria widely used by the scientific community is to estimate the correlation dimension as a proxy of the embedding dimension using the algorithm proposed by Grassberger and Procaccia (1983).

The main drawbacks of these heuristic approaches are the following: (i) they are not intrinsically statistical; (ii) they lead to estimators whose properties are unknown or largely unexplored; (iii) they do not take into account the results of any model fit. The alternative proposed by the statistical approach solves those 3 disadvantages. The statistical approach to state space reconstruction can be viewed as the best subset selection problem within the nonparametric regression context as argued Chan and Tong (2001). The idea behind it is to select the embedding parameters, τ and m, that provide the best fit in the estimation of the Lyapunov exponents taking into account some information criteria.

For instance, the Akaike's information criterion (AIC), the bayesian information criterion (BIC), the Hannan-Quinn information criterion (HQC), or the focused information criterion (FIC).

In any case, we think that the information derived from the heuristic approaches might be still useful and should not be disregarded as complementary information. The **DChaos** allows the R users to choose between both methods. By default, it uses the statistical approach based on model selection procedures instead of heuristic techniques. Now, once we have shown how to deal with the state space reconstruction, we are going to focus on the next step.

Estimating the Lyapunov exponents from time series data

In this section, we are going to discuss the main estimation methods of the Lyapunov exponents from time series data briefly. As we said before, we have assumed that the true data-generating process is unknown because in most real-world, observed time series data, the underlying generator system is rarely known. For this reason, we will not take into consideration estimation methods that presuppose the knowledge of the equations of the dynamical system. There are mainly two kind of methods to compute the estimated Lyapunov exponent from time-series data: the direct methods and the Jacobian indirect methods. Let us begin by focusing on the direct methods.

Direct method

The direct method was first proposed by Wolf et al. (1985), and then revisited by Rosenstein et al. (1993), and by Kantz (1994); Kantz and Schreiber (2004). The underlying algorithm is explained in detail in Kantz and Schreiber (1997). The three R packages already mentioned are based on ideas inspired by the time series analysis (TISEAN) project proposed by Hegger et al. (1999). All of them implement the algorithm proposed by Kantz using exclusively the direct method. The **tseriesChaos** package includes the function `lyap.k`, the **nonlinearTseries** package has a function called `maxLyapunov`, and `lyapunovPlot` is the function that belongs to the **fNonlinear** package. We will compare the results from these packages with those provide by the **DChaos** library later. Notice that the function `lyapunovPlot` has considered exactly the same code as the function `lyap.k`, which belongs to **tseriesChaos**. For this reason, we are going to consider only the algorithms included in the R packages **tseriesChaos** and **nonlinearTseries**.

The idea behind this direct approach can be described as follows. Given a time series data of length n, embedded in a m-dimensional space reconstructed with a time delay τ, the main aim is to compute a local average of the distances between every point x_i in the embedding space and its neighbors for $1 \leqslant i \leqslant n - (m-1)\,\tau$. That is, the evolution of the logarithm of this (local) mean distance $L\,(\Delta)$ is monitored for a finite number Δ of step ahead in time. The formula for this distance can be written as

$$L\,(\Delta) = \frac{1}{T}\sum_{i=1}^{T} \ln\left(\frac{1}{\#\mathcal{U}\,(x_i,\epsilon)}\sum_{x_j\in\mathcal{U}(x_i,\epsilon)}\left|x_{i+\Delta} - x_{j+\Delta}\right|\right), \tag{5}$$

where $T = n - (m-1)\,\tau$ is the number of points in the state space that are involved in the computation, $\#\mathcal{U}\,(x_i,\epsilon)$ is the number of neighbors of each point that are closer than a distance equal to ϵ and have a temporal separation greater than a certain value. We compute the log-average distance between every point and its neighbours, and we follow it for Δ time steps ahead. Thus, for every point x_i, we have approximately a straight line which represents the evolution of the logarithm of the local mean distance. The average line of all these straight lines gives the evolution of the mean distance for the whole attractor. The slope of this average line gives us the estimated values of the Lyapunov exponent. Note that the algorithm suggested by Kantz includes the one proposed by Rosentein, Collins, and De Luca as a special case (their number of neighbors are equal to 1).

Hence, the presence of a sharp linear region in the plot of the evolution of the logarithm of the mean distance between nearby points could be a strong signal of chaotic behavior. The main advantages of the direct methods are the following: (i) their low number of estimation parameters; (ii) easy implementation; (iii) they provide direct visual feedback to the R users whether the available time series data really exhibits exponential divergence on small scales; (iv) do not involve any kind of modeling; (v) do not require assumptions on the nature of the process. Despite this, there are certain drawbacks, as we pointed before: (i) it does not allow the estimation of the full spectrum of Lyapunov exponents; (ii) it is not robust to the presence of (small) noise because these estimators can produce a linear scaling region giving a positive estimate of the Lyapunov exponent even for non-chaotic systems; (iii) it does not have a satisfactory performance in detecting existing non-linearities on time series data of moderate sample sizes; (iv) the asymptotic distribution of the estimator cannot be derived, which means that it does not allow the building of formal tests.

For these reasons, we have focused on another alternative called the jacobian indirect methods. Let us move on to explain how we have implemented this approach.

Jacobian indirect method

The **DChaos** package is the first R library that provides the Jacobian indirect methods proposed by Eckmann and Ruelle (1985) for estimating the Lyapunov exponents. The procedure behind the Jacobian indirect method can be described as follows. We have assumed following Gençay and Dechert (1992) that there exist a function $g : \mathbb{R}^m \to \mathbb{R}^m$ such that $x^m_{t_i} = g\left(x^m_{t_i-\tau}\right)$, where $x^m_{t_i}$ are the non-uniform delayed-coordinate embedding vectors which contains the uniform case taking $t_i - t_{i-1} = t_s - t_{s-1}\ \forall i \neq s$. Under the assumption that the embedding is a homeomorphism, the map g is *topologically conjugate* to the unknown dynamic system f in equation 2. This implies that certain dynamical properties of f and g are the same. In our case, the Lyapunov exponents of f and g should be the same, so we can focus on estimating the exponents from the map g. The dynamical system g may be expressed as a matrix by

$$\begin{pmatrix} x_{t_i} \\ x_{t_i-\tau} \\ \vdots \\ x_{t_i-(m-2)\tau} \\ x_{t_i-(m-1)\tau} \end{pmatrix} = g \begin{pmatrix} x_{t_i-\tau} \\ x_{t_i-2\tau} \\ \vdots \\ x_{t_i-(m-1)\tau} \\ x_{t_i-m\tau} \end{pmatrix} = \begin{pmatrix} v\left(x_{t_i-\tau}, x_{t_i-2\tau}, \ldots, x_{t_i-(m-2)\tau}, x_{t_i-(m-1)\tau}, x_{t_i-m\tau}\right) \\ x_{t_i-\tau} \\ \vdots \\ x_{t_i-(m-2)\tau} \\ x_{t_i-(m-1)\tau} \end{pmatrix} \tag{6}$$

The Jacobian corresponding to the dynamical system g will be as follows:

$$Dg = \begin{pmatrix} \frac{\partial v}{\partial x_{t_i-\tau}} & \frac{\partial v}{\partial x_{t_i-2\tau}} & \frac{\partial v}{\partial x_{t_i-3\tau}} & \cdots & \frac{\partial v}{\partial x_{t_i-(m-1)\tau}} & \frac{\partial v}{\partial x_{t_i-m\tau}} \\ 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 \end{pmatrix} \tag{7}$$

Notice that the estimation of the Lyapunov exponent by the jacobian indirect method is reduced to the estimation of the unknown nonlinear function $v : \mathbb{R}^m \to \mathbb{R}$. The different approaches that compose the indirect method differ in the algorithm used for the estimation of this function v in the Jacobian (eq.7). The main contributions focus on two different approaches. Firstly, those who use some kind of local linear regression, see Sano and Sawada (1985), Eckmann et al. (1986), Brown et al. (1991), or their extension in the form of local polynomial regression proposed by Lu and Smith (1997). Second, other approaches use nonlinear models based on neural networks, see McCaffrey et al. (1992), Nychka et al. (1992), Dechert and Gençay (1992), Whang and Linton (1999), Shintani and Linton (2003, 2004). In this sense, there are no analogous functions in the R packages already mentioned since they consider exclusively the direct method. In addition, those packages do not allow us to make inferences about the Lyapunov exponents and test if the estimated values of the Lyapunov exponents are or not statistically significant.

We have focused on the neural net approach, which is a global nonparametric method that tries to estimate the underlying dynamic system without imposing the restriction of local linearity. This approach has the following advantages over all other methods: (i) its robustness to the presence of (small) noise as the measurement errors present in most real-world, observed time series data; (ii) their satisfactory performance in detecting existing non-linearities on time series data of moderate sample sizes; (iii) allows the estimation of the full spectrum of Lyapunov exponents; (iv) the asymptotic distribution of the estimator can be derived, allowing the building of formal tests.

Neural network approach

The neural network (or neural net) estimator of the Lyapunov exponent was first proposed by McCaffrey et al. (1992) and Nychka et al. (1992) and then revisited by Gençay and Dechert (1992) and by Shintani and Linton (2003, 2004). Note that in our case, the main reason for using neural network models is not to look for the best predictive model but to estimate a model that captures the non-linear time dependence well enough and, additionally, allows us to obtain in an analytical way (instead of numerical) the jacobian functional of the unknown data-generating process (eq.7).

The estimation of this Jacobian will allow us to contrast the hypothesis of chaos (eq.1) using equation 2. Hornik et al. (1989) showed that any standard feed-forward networks with as few as one hidden layer using arbitrary squashing functions are capable of approximating any Borel measurable function from one finite-dimensional space to another for any desired degree of accuracy, providing sufficiently many hidden units. In this sense, the feed-forward networks are a class of universal approximations. Theoretically, neural nets are expected to perform better than other approximation methods, especially with high-dimensional models, since the approximation form is not so sensitive to the increasing dimension. The results proposed by the authors mentioned above have enabled us to consider a neural network with just one single hidden layer. The number of hidden units (or neurons) in the single hidden layer is determined by statistical methods based on model selection criteria, as it appears in the results of the paper. Let us show how we have obtained a consistent neural net estimator based on the robust estimation of the function v in the Jacobian (eq.7). Note that if we consider the m-dimensional reconstruction vector as defined by eq.6,

$$x_{t_i} = v\left(x_{t_i-\tau}, x_{t_i-2\tau}, \ldots, x_{t_i-(m-2)\tau}, x_{t_i-(m-1)\tau}, x_{t_i-m\tau}\right),$$

the neural network estimator can be obtained by approximating the unknown non-linear function v through a feed-forward single hidden layer network with a single output by

$$v \approx \hat{v} = \Phi_0\left[\hat{\alpha}_0 + \sum_{q=1}^{h} \hat{\omega}_{q0}\Phi_q\left(\hat{\alpha}_q + \sum_{j=1}^{m} \hat{\omega}_{jq} x_{t_i-j\tau}\right)\right], \tag{8}$$

where $\Phi_0 \in I$, $\hat{\alpha}_0$ is the estimated network bias from input, h is the number of neurones (or nodes) in the single hidden layer, $\hat{\omega}_{q0}$ are the estimated layers connection weights from input to hidden layer, Φ_q is the transfer function, which in our case is the logistic function, $\hat{\alpha}_q$ is the estimated network bias from hidden layer, m is the embedding dimension, and $\hat{\omega}_{jq}$ are the estimated layers connection weights from hidden layer to output. The issue of parameter estimation is reduced to a least squares problem in which the quantity to be minimized is defined by

$$\sum_{i=1}^{n}\left(x_{t_i} - \left[\alpha_0 + \sum_{q=1}^{h} \omega_{q0}\Phi_q\left(\alpha_q + \sum_{j=1}^{m} \omega_{jq} x_{t_i-j\tau}\right)\right]\right)^2$$

Note that in our case, the main reason for using neural network models is not to look for the best predictive model but to estimate a model that captures the non-linear time dependence well enough and, additionally, allows us to obtain in an analytical way (instead of numerical) the jacobian functional of the unknown underlying generator system (eq. 7). The estimation of this jacobian or partial derivatives will later allow us to contrast our hypothesis of chaos using equation 2. We have obtained the partial derivates of the Jacobian in eq.7, applying the chain rule to eq.8 as

$$\frac{\partial \hat{v}}{\partial x_{t_i-j\tau}} = \Phi'_0\left(z_0\right)\sum_{q=1}^{h} \hat{\omega}_{q0}\Phi'_0\left(z_q\right)\hat{\omega}_{jq}, \tag{9}$$

where

$$z_0 = \hat{\alpha}_0 + \sum_{q=1}^{h} \hat{\omega}_{q0}\Phi_q\left(z_q\right), \quad z_q = \hat{\alpha}_q + \sum_{j=1}^{m} \hat{\omega}_{jq} x_{t_i-j\tau},$$

and the estimated partial derivates are given by

$$\hat{D}g = \begin{pmatrix} \frac{\partial \hat{v}}{\partial x_{t_i-\tau}} & \frac{\partial \hat{v}}{\partial x_{t_i-2\tau}} & \frac{\partial \hat{v}}{\partial x_{t_i-3\tau}} & \cdots & \frac{\partial \hat{v}}{\partial x_{t_i-(m-1)\tau}} & \frac{\partial \hat{v}}{\partial x_{t_i-m\tau}} \\ 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 \end{pmatrix} \tag{10}$$

The **DChaos** package provides the function `netfit` whose arguments are `serie`, `m`, `lag`, `timelapse`, `h`, `w0maxit`, `wtsmaxit`, `pre.white`, `trace`, `seed.t`, `seed`, to fit the unknown non-linear function v in eq.8 through a feed-forward single hidden layer network. See R documentation for the interpretation of this parameter set. We have considered the `nnet` function that belongs to the **nnet** package to fit the neural net models. Notice that the process of adjustment to a neural network often suffers from being trapped in local optima, and different initialization strategies should be taken into account.

For this reason, we have implemented the function `w0.net` whose arguments are `x`, `y`, `m`, `h`, `rangx`, `w0maxit`, `seed.t`, `seed`. This function estimates previously the initial parameter vector of the neural network being able to set the maximum number of iterations that we want to obtain, setting `w0maxit`. In addition, by default, the neural network estimation is initialized with a fixed seed denoted by `seed.t=TRUE` with a value equal to `seed=56666459`. The R users can let the seed be fixed either randomly by `seed.t=FALSE` or even fix other values of the seed to be able to replicate the results obtained.

Best-fitted neural network models. We are going to illustrate an example considering the time series data previously simulated from the logistic map. The `netfit` function returns several objects. The best-fitted feed-forward single hidden layer neural net model is saved. It also contains some useful information about the best set of weights found, the fitted values, the residuals obtained, or the best embedding parameters set chosen. The best 10 models are displayed on the console as we show below for `m` $\in \{1:4\}$, `lag` $\in \{1:3\}$ and `h` $\in \{2:10\}$. The first column is the neural net number, the second column is the embedding dimension, the third column is the lag or reconstruction delay considered, the fourth column is the number of neurons (or nodes) in the single hidden layer, and the fifth column is the Bayesian Information Criterion (BIC) value corresponding to that neural net. Notice that the neural net models are sorted from lowest to highest BIC values. In this case, the best-fitted neural net model has the following parameter set values `m=1`, `lag=1`, and `h=9`. The BIC criterion is defined following Shintani and Linton (2003) by

$$BIC = \log(RSS) + \frac{\log(n)}{n}\left[1 + h(m+2)\right],$$

where RSS is the residual sum of squares, n is the number of observations, m is the embedding dimension, and h is the number of neurones (or nodes) used in the single hidden layer as noted above.

```
## Provides the best-fitted neural network models for certain parameter set
model <- DChaos::netfit(ts, m=1:4, lag=1:3, timelapse="FIXED", h=2:10)

 Best models:
   m lag  h   BIC
85 1   1  9 -15.96086
73 1   1  8 -15.95343
97 1   1 10 -15.91173
37 1   1  5 -15.29339
61 1   1  7 -15.26654
63 3   1  7 -15.04988
49 1   1  6 -15.04090
3  3   1  2 -14.74957
98 2   1 10 -14.17451

## Summary method for a nnet object
## Provides the best set of weights found for the best-fitted neural net model (#85)
summary(model)

a 1-9-1 network with 28 weights
options were - linear output units
 b->h1 i1->h1
 -0.91   0.38
 b->h2 i1->h2
 -6.29   4.61
 b->h3 i1->h3
  3.06  -3.11
 b->h4 i1->h4
  1.49   6.10
 b->h5 i1->h5
  0.11   0.64
 b->h6 i1->h6
  0.70   3.02
 b->h7 i1->h7
 -0.19   0.57
 b->h8 i1->h8
 -1.37   0.84
 b->h9 i1->h9
 -0.03  -1.44
```

```
 b->o h1->o h2->o h3->o h4->o h5->o h6->o h7->o h8->o h9->o
-4.75 -0.16 -3.30  2.97  0.64 -0.39  5.07 -0.45 -2.02 -2.27
```

Partial derivatives are calculated analytically from the best-fitted neural net model. The **DChaos** package provides as well the function `jacobian.net` which arguments are `model`, `data`, `m`, `lag`, `timelapse`, `h`, `w0maxit`, `wtsmaxit`, `pre.white`, `trace`, `seed.t`, `seed`, to compute the partial derivatives of the Jacobian in eq.10 from the best-fitted feed-forward single hidden layer network. See R documentation for the interpretation of this parameter set. This function returns several objects considering the parameter set selected by the user. Partial derivatives are calculated analytically from the best-fitted neural net model. It also contains some useful information about the best-fitted feed-forward single hidden layer neural net model saved, the best set of weights found, the fitted values, the residuals obtained, or the best embedding parameters set chosen. This function allows the R user uses the data previously obtained from the best-fitted neural network estimated by the `netfit` function if `model` is not empty. Otherwise, `data` has to be specified. Let us show an example. Firstly, we provide the first seven partial derivatives values (`dx1`) corresponding to the best-fitted neural net model estimated above (neural net #85).

```
## Computes analytically the partial derivatives from the best-fitted neural net model
## showed in the netfit example  (#85)
jacobian <- DChaos::jacobian.net(model=model)
show(head(jacobian$jacobian))
```

```
         dx1
1 -1.9701373
2 -2.0499734
3 -1.8893034
4 -2.2064735
5 -1.5568866
6 -2.7836348
7 -0.1313633
```

Now, we are going to provide the results obtained from the function `jacobian.net` without setting the best-fitted neural net model estimated previously. We have considered the dataset simulated from the logistic map again. We have chosen the following parameter set values `m=3`, `lag=1`, `timelapse=FIXED`, `h=2:10`, `w0maxit=100`, `wtsmaxit=1e6`, `pre.white=FALSE`, `trace=1`, `seed.t=TRUE`, and `seed=56666459`. We show below the first seven partial derivatives values (`dx1`, `dx2`, `dx3`) corresponding to compute the partial derivatives of the Jacobian following eq.10 from the best-fitted feed-forward single hidden layer network (neural net #6).

```
## Partial derivatives are calculated analytically without setting previously
## any neural net model
jacobian <- DChaos::jacobian.net(data=ts, m=3:3, lag=1:1, timelapse="FIXED", h=2:10)
```

```
 Best models:
  m lag  h   BIC
6 3   1  7 -15.04988
1 3   1  2 -14.74957
7 3   1  8 -13.73532
4 3   1  5 -13.63287
3 3   1  4 -13.29422
8 3   1  9 -13.08082
2 3   1  3 -11.79369
9 3   1 10 -11.78703
5 3   1  6 -11.61098
```

```
show(head(jacobian$jacobian))
```

```
         dx1           dx2           dx3
1 -1.8880449 -0.0003353808  0.0020567805
2 -2.2071082 -0.0007068399  0.0014469903
3 -1.5552770  0.0001417462  0.0026879632
4 -2.7896525 -0.0011615596  0.0003214048
5 -0.1343617  0.0029934521  0.0053918764
6 -3.9235626 -0.0012869863 -0.0015386951
7  3.9269526  0.0116947029  0.0046071740
```

Lyapunov exponent estimation and inference

In this section, we provide the right procedure to obtain a consistent estimator of the Lyapunov exponent from the partial derivates estimated following eq.10. In addition, we are going to illustrate how to test the hypothesis of chaos (eq.1) based on the theoretical asymptotic properties of the neural net estimator. The kth Lyapunov exponent is given by

$$\hat{\lambda}_k = \lim_{M\to\infty} \frac{1}{M} \log \mu_k \left(\left| \hat{D}g^M \right| \right), \tag{11}$$

where μ_k is the kth largest eigenvalue provided by $\hat{D}g^M = \hat{D}g\left(x_{t_M}\right) \cdot \hat{D}g\left(x_{t_{M-1}}\right) \cdot \ldots \cdot \hat{D}g\left(x_{t_1}\right)$ for $k = 1, 2, 3, \ldots, m$. $\hat{D}g\left(\right)$ are the partial derivates estimated above following eq.10. Note that it is necessary to distinguish between the sample size n used for estimating the partial derivatives of the Jacobian in eq.9 and the block length M, defined in eq.11, which is the number of evaluation points (number of products of the Jacobian) used for estimating the kth Lyapunov exponent. Since the number of evaluation points is less than or equal to n, M can be also understood as the sample size of a subsample. We have taken into account in our algorithms both the full sample and three different methods of subsampling by blocks, as we show below in table 2. The first column shows the blocking methods considered. The second, third, and fourth column give the sample size, the block length, and the block number of each subsampling method, respectively. The fifth column provides the way in which the algorithm picks the position of each element inside the block where each block B corresponds to one row. The bootstrap blocking method takes random samples without replacement by each block.

Blocking method	Sample size	Block length	Block number	Block subset
Full	n			
Non-overlapping	Blocks	M	$B = n/M$	$\begin{cases} 1, 2, \ldots, M \\ M+1, M+2, \ldots, 2M \\ \vdots \\ (B-1)M+1, (B-1)M+2, \ldots, BM \end{cases}$
Equally spaced	Blocks	M	$B = n/M$	$\begin{cases} 1, 1+B, 1+2B, \ldots, 1+(M-1)B \\ 2, 2+B, 2+2B, \ldots, 2+(M-1)B \\ \vdots \\ B, 2B, 3B, \ldots, BM \end{cases}$
Bootstrap	Blocks	M	$B = 100$	Randomly

Table 2: Blocking methods for figuring out the neural network estimator of the kth Lyapunov exponent by Jacobian indirect methods.

The asymptotic properties of the non parametric neural network estimator of the Lyapunov exponent $\hat{\lambda}_k$ was derived by Shintani and Linton Shintani and Linton (2004). They provided a statistical framework for testing the hypothesis of chaos (eq.1) based on the neural net estimator of the Lyapunov exponent and the consistent estimator of its variance. Their results showed the asymptotic normality of the Lyapunov exponent estimator by

$$\sqrt{M}\left(\hat{\lambda}_k - \lambda_k\right) \sim N\left(0, \varphi_k\right), \tag{12}$$

where φ_k is the variance of the kth Lyapunov exponent estimator (eq.2). They proved that $\hat{\varphi}_k$ is a consistent variance estimator of φ_k. It can be defined as follows:

$$\hat{\varphi}_k \equiv Var\left(\hat{\lambda}_k\right) = \lim_{M\to\infty} Var\left(\frac{1}{\sqrt{M}} \sum_{t=1}^{M} \eta_{k,t}\right), \tag{13}$$

where M is the subsampling size, that is equal to n for the full sample. The quadratic spectral kernel function $\eta_{k,t}$ is given by $\eta_{k,t} = \xi_{k,t} - \hat{\lambda}_k$ following Shintani and Linton (2004). The parameter $\xi_{k,t}$ is obtained by

$$\xi_{k,t} = \frac{1}{2}\log\mu_k\left(\left|\hat{D}g^t\right|\right) - \frac{1}{2}\log\mu_k\left(\left|\hat{D}g^{t-1}\right|\right), \tag{14}$$

where $t = 1, 2, \ldots, M$. Then, a feasible test statistics were introduced, and a one-sided test was proposed for the purpose of testing the hypothesis of chaos (eq.1) based on the theoretical asymptotic properties of the neural net estimator. That is, we would know if the estimated Lyapunov exponent values are or not statistically significant. Hence, our interest will be to test the null hypothesis $H_0 : \hat{\lambda}_k > 0$ against the alternative $H_1 : \hat{\lambda}_k \leqslant 0$. The test statistics can be defined as follows:

$$\hat{t}_k = \frac{\hat{\lambda}_k}{\sqrt{\hat{\varphi}_k / M}} \sim N\left(0, \hat{\varphi}_k\right) \tag{15}$$

Hence, we will reject the null hypothesis if $\hat{t}_k \leqslant -z_\alpha$, where z_α is the critical value that satisfies $\Pr\left[Z \geqslant z_\alpha\right] = \alpha$ with Z being a standard normal random variable and α is the significance level. Under the null hypothesis H_0 that the data-generating process is chaotic, the neural net estimator $\hat{\lambda}_k$ leads to asymptotically valid inferences in that the associated p-value follows a normal distribution on $N\left(0, \hat{\varphi}_k\right)$. Rejecting the null hypothesis $H_0 : \hat{\lambda}_k > 0$ means lack of chaotic behavior. Thus, we have used these results to calculate the standard error of the Lyapunov exponent estimator and investigate the statistical significance of the sign of the exponents.

Lyapunov exponent estimator by jacobian indirect method using a neural net approach. The **DChaos** package provides several ways to figure out robustly the neural net estimator of the kth Lyapunov exponent. On the one hand, if the R users have previously obtained the partial derivatives from the `jacobian.net` function, they can apply directly the function `lyapunov.spec`, which estimates the Lyapunov exponent spectrum taking into account the QR decomposition procedure. They can also use the function `lyapunov.max`, which estimates only the largest Lyapunov exponent considering the Norma-2 procedure. The arguments of both functions are the same (`data`, `blocking`, `B`, `doplot`). See R documentation for the interpretation of this parameter set. Hence, the **DChaos** package allows the R users to choose between two different procedures to obtain the neural net estimator of the kth Lyapunov exponent and four ways of subsampling by blocks: full sample, non-overlapping sample, equally spaced sample, and bootstrap sample; for a review see table 2.

Note that the **DChaos** package provides 8 internal functions (one for each procedure and blocking method), which estimate the Lyapunov exponents consistently (eq.11). These functions return several objects considering the parameter set selected by the user. The largest Lyapunov exponent (`lyapunov.max`) or the Lyapunov exponent spectrum (`lyapunov.spec`) by each blocking method are estimated. They also contain some useful information about the estimated jacobian, the best-fitted feed-forward single hidden layer neural net model, the best set of weights found, the fitted values, the residuals obtained, the best embedding parameters set chosen, the sample size, or the block length considered by each blocking method. These functions provide the standard error, the z test value, and the p-value for testing the hypothesis of chaos (eq.1). Rejecting the null hypothesis H_0 means lack of chaotic behavior. That is, the data-generating process does not have a chaotic attractor because it does not show the property of sensitivity to initial conditions. The blocking methods split the time-series data into several blocks to estimate a Lyapunov exponent for each subsample. The R users can choose the non-overlapping sample (`blocking = NOVER`), the equally spaced sample (`blocking = EQS`), or the bootstrap sample (`blocking = BOOT`). The mean and median values of the Lyapunov exponent for each block or subsample are saved. By default, we recommend using the median value as it is more robust to the presence of outliers. Notice that parameter `B` (a non-negative integer denoting the number of bootstrap iterations) will only be considered if the R users choose the bootstrap blocking method.

Now, we are going to compare the results obtained from the `lyapunov.max` and `lyapunov.spec` functions for estimating the largest Lyapunov exponent and the Lyapunov exponent spectrum respectively. We have considered the dataset simulated from the logistic map again, the best-fitted neural net model estimated previously (neural net #6) which the best embedding parameters set chosen is `m=3`, `lag=1`, `h=7`, and the partial derivatives are `dx1`, `dx2`, `dx3`. We show below the estimation of the largest Lyapunov exponent provided by the Norma-2 procedure and the Lyapunov exponent spectrum taking into account the results obtained by the QR decomposition procedure (bootstrap blocking method). The results provided by both methods are very similar between them (0.6943782 and 0.6942063 respectively), but the`lyapunov.spec` function also allows the estimation of the full spectrum of the Lyapunov exponents. The estimated values by both methods are certainly close to the theoretical value (0.6931472), and they are statistically significant at the 99% confidence level. This fact evidences the consistency of the proposed algorithms.

```
## Provides the largest Lyapunov exponent by the Norma-2 procedure considering the
## bootstrap blocking method from the best-fitted neural net model and the partial
## derivatives showed in the jacobian.net example.
exponent <- DChaos::lyapunov.max(data=jacobian, blocking="BOOT", doplot=FALSE)

## Provides summary statistics of the results given in an object of class lyapunov
summary(exponent)

Call:
Largest Lyapunov exponent

Coefficients:
             Estimate Std. Error   z value Pr(>|z|)
Exponent    0.6943782 0.00443619  1783.442        1
---
Procedure: Norma-2 by bootstrap blocking method
Embedding dimension: 3, Time-delay: 1, No. hidden units: 7
Sample size: 997, Block length: 82, No. blocks: 1000

## Provides the Lyapunov exponent spectrum by the QR decomposition procedure considering
## the bootstrap blocking method from the best-fitted neural net model and the partial
## derivatives showed  in the jacobian.net example as well.
exponent <- DChaos::lyapunov.spec(data=jacobian, blocking="BOOT", doplot=FALSE)

## Provides summary statistics of the results given in an object of class lyapunov
summary(exponent)

Call:
Lyapunov exponent spectrum

Coefficients:
             Estimate Std. Error   z value Pr(>|z|)
Exponent 1  0.6942063 0.00309979  1670.773        1
Exponent 2 -3.5818150 0.00915042 -3873.927        0
Exponent 3 -3.7257022 0.00898081 -4053.132        0
---
Procedure: QR decomposition by bootstrap blocking method
Embedding dimension: 3, Time-delay: 1, No. hidden units: 7
Sample size: 997, Block length: 82, No. blocks: 1000
```

On the other hand, the **DChaos** package provides an all-in-one function called `lyapunov`. We have considered it appropriate to incorporate a function that unifies the whole process to make it easier and more intuitive for the R users. The `lyapunov()` function has the following usage:

```
lyapunov(data, m, lag, timelapse, h, w0maxit, wtsmaxit, pre.white, lyapmethod,
         blocking, B, trace, seed.t, seed, doplot)
```

This function has fifteen main arguments where only the first one is mandatory. These are:

`data`: a `vector`, a time-series object `ts` or `xts`, a `data.frame`, a `data.table`, or a `matrix` depending on the method selected in `timelapse`. If `FIXED` has been selected, `data` must be a `vector` or a time-series object `ts` or `xts`. Otherwise, `VARIABLE` has to be specified. In this case, `data` must be a `data.frame`, a `data.table`, or a `matrix`.

`m`: a non-negative integer denoting a lower and upper bound for the embedding dimension (Default 1:4).

`lag`: a non-negative integer denoting a lower and upper bound for the the reconstruction delay (Default 1:1).

`timelapse`: a character denoting if the time-series data are sampled at uniform time-frequency e.g., 1-month, 1-day, 1-hour, 30-min, 5-min, 1-min, and so on, `FIXED` or non-uniform time-frequency, which are not equally spaced in time `VARIABLE` (Default `FIXED`).

`h`: a non-negative integer denoting a lower and upper bound for the number of neurons (or nodes) in the single hidden layer (Default 2:10).

`w0maxit`: a non-negative integer denoting the maximum iterations to estimate the initial parameter vector of the neural net models (Default 100).

`wtsmaxit`: a non-negative integer denoting the maximum iterations to estimate the weights parameter vector of the neural net models (Default 1e6).

`pre.white`: a logical value denoting if the user wants to use as points to evaluate the partial derivatives the delayed vectors filtered by the neural net model chosen `TRUE` or not `FALSE` (Default `TRUE`).

`lyapmethod`: a character denoting the procedure chosen to estimate the Lyapunov exponent. If `LLE` has been selected, the function will estimate only the largest Lyapunov exponent through the Norma-2 method. If `SLE` has been selected, the function will estimate the Lyapunov exponent spectrum through the QR decomposition. Otherwise, `ALL` has to be specified. In this case, the function will estimate the Lyapunov exponent considering both procedures (Default `SLE`).

`blocking`: a character denoting the blocking method chosen for figuring out the Lyapunov exponent. Available options are `FULL` if the user considers the full sample, `NOVER` if the user considers the non-overlapping sample, `EQS` if the user considers the equally spaced sample, `BOOT` if the user considers the bootstrap sample, or `ALL` if the user considers all of them (Default `BOOT`).

`B`: a non-negative integer denoting the number of bootstrap iterations (Default 1000).

`trace`: a binary value denoting if the user wants to print the output on the console `1` or not `0` (Default `1`).

`seed.t`: a logical value denoting if the user wants to fix the seed `TRUE` or not `FALSE` (Default `TRUE`).

`seed`: a non-negative integer denoting the value of the seed selected if `seed.t = TRUE` (Default `56666459`).

`doplot`: a logical value denoting if the user wants to draw a plot `TRUE` or not `FALSE`. If it is `TRUE`, the evolution of the Lyapunov exponent values are represented for the whole period considering the different procedures and blocking methods chosen by the user. The default value is `TRUE`.

The `lyapunov()` function provides (at the same time) the delayed-coordinate embedding vector from time-series data (eq.3), estimates the best-fitted neural net model from the delayed-coordinate embedding vectors (eq.8), calculates analytically the partial derivatives from the chosen neural nets model (eq.10). Finally, the neural net estimator of the kth Lyapunov exponent (eq.11) is obtained from the partial derivatives computed previously. This function returns several objects considering the parameter set selected by the user. The largest Lyapunov exponent (`lyapmethod = LLE`), the Lyapunov exponent spectrum (`lyapmethod = SLE`), or both (`lyapmethod = ALL`) by each blocking method are estimated. They also contain some useful information about the estimated Jacobian, the best-fitted feed-forward single hidden layer neural net model, the best set of weights found, the fitted values, the residuals obtained, the best embedding parameters set chosen, the sample size, or the block length considered by each blocking method. This function also provides the standard error, the z test value, and the p-value for testing the hypothesis of chaos (eq.1). As we said before, rejecting the null hypothesis H_0 means lack of chaotic behavior. That is, the data-generating process does not have chaotic attractor because it does not show the property of sensitivity to initial conditions.

Now, we are going to show an example of this all-in-one function. We provide the results obtained from the `lyapunov` function considering the dataset simulated from the logistic map again for the following parameter set values `m=3:3`, `lag=1:1`, `timelapse=FIXED`, `h=2:10`, `w0maxit=100`, `wtsmaxit=1e6`, `trace=1`, `seed.t=TRUE`, `seed=56666459`, and `doplot=FALSE`. In this case, we have estimated the Lyapunov exponent spectrum taking into account the results provided by the QR decomposition procedure (`lyapmethod=SLE`) and all blocking methods (`blocking=ALL`). Note that the delayed-coordinate embedding vector, the best-fitted neural net model (with a lower BIC value), and the partial derivatives are internally calculated according to the parameter set chosen (all-in-one way). The **DChaos** package provides the `summary` method for class `"lyapunov"` called `summary.lyapunov` as we show below.

Let us point out briefly some useful information to understand the results presented in this example. For the estimation based on blocking methods, median values of all used blocks are given. For the block length M, the **DChaos** package uses $M = int\left[c \times (n/\log n)^{1/6}\right]$ with $c = 36.2$ where $int\,[A]$ signifies the integer part of A following Shintani and Linton (2004). The number of blocks B depends on the sample size n of each time series data; for a review, see table 2. QS Kernel with optimal bandwidth has been used for the heteroskedasticity and autocorrelation consistent covariance matrix estimation following Andrews (1991). In this example, the results provided by the three blocking methods improve the case of the full sample being very similar between them. This result is consistent with the recommendation proposed by Shintani and Linton (2004) on the use of blocking methods instead of full sampling when estimating the Lyapunov exponent.

```
## Provides the Lyapunov exponent spectrum by several blocking methods (all-in-one)
exponent <- DChaos::lyapunov(ts, m=3:3, lag=1:1, timelapse="FIXED", h=2:10, w0maxit=100,
                    wtsmaxit=1e6, pre.white=TRUE, lyapmethod="SLE", blocking="ALL",
                    B=100, trace=1, seed.t=TRUE, seed=56666459, doplot=FALSE)

 Best models:
  m lag  h       BIC
8 3   1  9 -13.37732
9 3   1 10 -12.90353
7 3   1  8 -12.76361
4 3   1  5 -12.60372
2 3   1  3 -12.20489
5 3   1  6 -11.88894
6 3   1  7 -11.85880
1 3   1  2 -11.75219
3 3   1  4 -10.76974

## Summary method for a lyapunov object
## Provides summary statistics of the results given in an object of class lyapunov
summary(exponent)

Call:
Lyapunov exponent spectrum

Coefficients:
             Estimate Std. Error    z value Pr(>|z|)
Exponent 1  0.6922016 0.08452517   74.15722        1
Exponent 2 -2.7082326 0.09158355 -267.77832        0
Exponent 3 -3.2056094 0.10913227 -265.98941        0
---
Procedure: QR decomposition by bootstrap blocking method
Embedding dimension: 3, Time-delay: 1, No. hidden units: 9
Sample size: 997, Block length: 82, No. blocks: 100

... only the first method is shown (see lyapunov object)
```

Analysis of chaotic time-series data adding a measurement noise

Firstly, let us illustrate how adding just a (small) measurement noise to a well-known deterministic dynamic system as the logistic map. This fact increases the dispersion of the huge amount of points that describe the attractor with the consequent inaccuracy; see figure 1. We have added to each time-series data a normal multinomial error term denoted by $\varepsilon_t \sim N(0, s)$ with different variance values s. The R code used to obtain the graphs shown below is the following. We have used the dataset simulated previously from the logistic map with chaotic behavior.

```
## The user should make a loop adding a measurement noise with several variance values
par(mfrow=c(2,3))
noise  <-  c(0,0.01,0.02,0.03,0.04,0.05)
for (i in 1:length(noise)){
  X    <-  logistic.sim(a=4, n=1000, s=noise[i])
           plot(X[c(1:999)],X[c(2:1000)], cex=.4, xlab= "X(t-1)",ylab="X(t)",
           main=paste("Measurement noise","s =",noise[i]))
}
```

In order to test the robustness of the different methods available to estimate the Lyapunov exponent, we are going to compare the results provided by the **tseriesChaos** package and the **nonlinearTseries** package with those of the **DChaos** package taking into account different measurement noise levels. We have considered four well-known chaotic dynamic systems. These datasets are available on the **DChaos** library; see Table 3.

The commands `install.packages("DChaos")` and `library(DChaos)` will download, install, and load the **DChaos** package so it can be used. The command `set.seed(34)` also will set the seed for reproducibility. To save CPU time, we have set the embedding dimension $1 \leqslant m \leqslant 7$, the time delay $\tau = 1$, the number of nodes in the single hidden layer $2 \leqslant h \leqslant 10$, and the length of all time series data

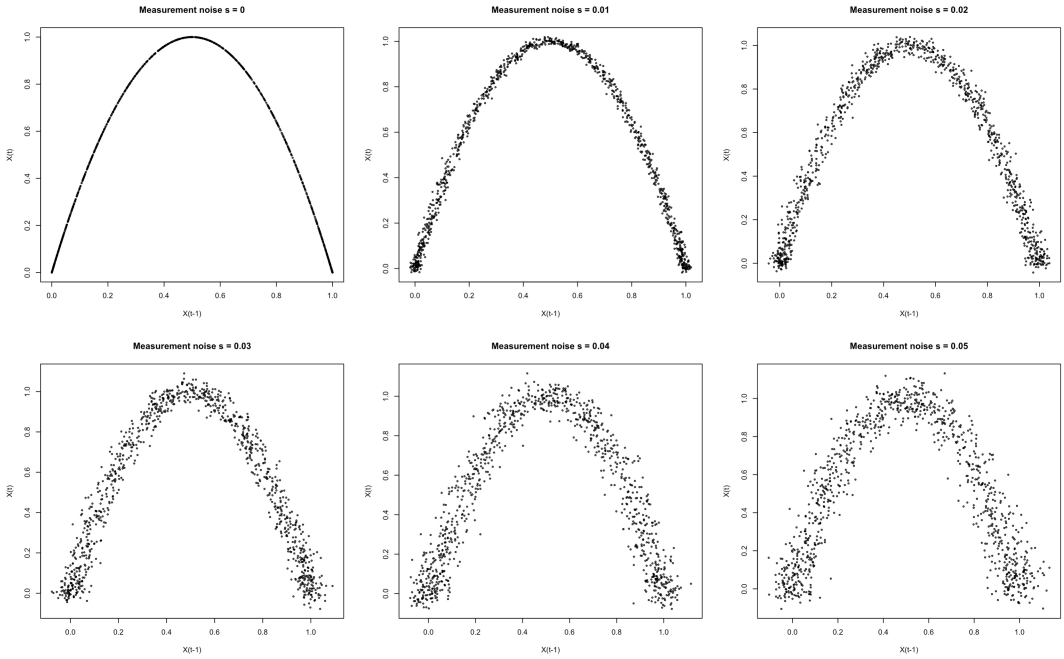


Figure 1: Logistic map attractor adding a measurement noise with several variance values

is $n = 1000$. We have considered only the bootstrap blocking method, and the number of bootstrap iterations is $B = 1000$. In this case, 63 different neural nets models have been estimated from each of the 24000 simulated series in order to obtain the results shown in table 4.

Note that the neural net models are sorted from lowest to highest BIC values. Then each best-fitted neural net model (with a lower BIC value) is considered to estimate the 24000 largest Lyapunov exponents. The mean square error (MSE) is calculated between the theoretical value and the value estimated by several direct and indirect methods. To do that, we have used the Monte Carlo method. We have done 1000 repetitions by different initial conditions when simulating the time-series data from the four dynamic systems and six measurement noise levels considered.

The MSE values based on the estimation of the largest Lyapunov exponent from the direct methods provided by the **tseriesChaos** and **nonlinearTseries** packages are denoted by D1 and D2, respectively. Those obtained by the Jacobian indirect methods through the **DChaos** library are denoted by N2 (`lyapunov.max`) and QR (`lyapunov.spec`) regarding the Norma-2 and QR decomposition procedures, respectively. The results shown in table 4 provide the following comments.

First, we can remark that our algorithms are robust to the presence of (small) measurement errors because the results obtained are comparable to those which are noise-free. Although as the noise increases, the error committed increases, but it is not proportional in any case. Second, the indirect methods provide better estimates than direct methods in all the experiments we have conducted. The algorithms proposed by the **tseriesChaos** package behave better than those of the **nonlinearTseries** library. Between the two methods available in the **DChaos** package, we do not observe significant differences.

Dynamic system	Equations	Parameters	λ_{th}
Logistic	$x_t = \mu x_{t-1}\left(1 - x_{t-1}\right)$	$\mu = 4$	0.69314
Gauss	$x_t = e^{-\alpha x_{t-1}^2} + \beta$	$\alpha = 6.2, \beta = -0.5$	0.38367
Hénon	$x_t = 1 - a x_{t-1}^2 + y_{t-1}$	$a = 1.4$	0.41921
	$y_t = b x_{t-1}$	$b = 0.3$	-1.63479
Rössler	$\dot{x} = -y - z$	$a = 0.2$	0.07143
	$\dot{y} = x + ay$	$b = 0.2$	0.00000
	$\dot{z} = b + \left(x - c\right) z$	$c = 5.7$	-0.53943

Table 3: Theoretical Lyapunov exponents (λ_{th}) from time-series data available on **DChaos** package.

Logistic map	$s = 0$	$s = 0.01$	$s = 0.02$	$s = 0.03$	$s = 0.04$	$s = 0.05$
D1 direct method	0.0001220	0.0056643	0.0030120	0.003006	0.0033485	0.0030913
D2 direct method	0.4802315	0.4765133	0.4814125	0.4815446	0.4790305	0.4830895
N2 indirect method	0.0000324	0.0000382	0.0000691	0.0000994	0.0001314	0.0001532
QR indirect method	0.0000331	0.0000348	0.0000672	0.0000986	0.0000997	0.0001124
Gauss map						
D1 direct method	0.0015270	0.0111180	0.0205349	0.0293853	0.0275621	0.0336681
D2 direct method	0.1474216	0.1480353	0.1477251	0.1464405	0.1481204	0.1476371
N2 indirect method	0.0000436	0.0000526	0.0000555	0.0000678	0.0000719	0.0000944
QR indirect method	0.0000618	0.0000656	0.0000672	0.0000782	0.0000817	0.0000924
Hénon system						
D1 direct method	0.0035650	0.0067221	0.0092761	0.0100339	0.0141379	0.0189926
D2 direct method	0.3121588	0.3133259	0.3145991	0.3115671	0.3226997	0.3178221
N2 indirect method	0.0000365	0.0000486	0.0000635	0.0000761	0.0000899	0.0000917
QR indirect method	0.0000318	0.0000451	0.0000589	0.0000601	0.0000866	0.0000932
Rössler system						
D1 direct method	0.0024471	0.0049521	0.0063189	0.0072719	0.0127326	0.0174911
D2 direct method	0.6398841	0.6412752	0.6388524	0.6396631	0.6451333	0.6499127
N2 indirect method	0.0002477	0.0003529	0.0005997	0.0006122	0.0009521	0.0019947
QR indirect method	0.0003168	0.0004891	0.0006070	0.0007155	0.0008190	0.0009268

Table 4: The mean square error (MSE) values based on the estimation of the largest Lyapunov exponent from direct methods provided by the **tseriesChaos** (D1) and **nonlinearTseries** (D2) packages are showed. Also, those obtained by the Jacobian indirect methods through the **DChaos** library are presented (N2 for `lyapunov.max` and QR for `lyapunov.spec`).

Third, the direct methods are surely less flexible and robust but much faster and still informative. The Jacobian indirect methods based on the neural net approach seems to perform well for every noisy time series data. The price we have to pay is a greater computational complexity from two points of view, the computing time and the tuning parameters (node weights). Fourth, we have only focused on the largest Lyapunov exponent as direct methods do not estimate the full spectrum. In addition, those methods do not allow us to make inferences about it. In our case, we will be able to do so. Hence, we are going to focus on the reliability of the proposed methods. We want to know the power and size of our algorithms.

Finally, we will focus on testing the reliability of the algorithms provided by the **DChaos** package. For this purpose, we have calculated the so-called size and power of our hypothesis test. Due to the test is based on probabilities, there is always a chance of making an incorrect conclusion. When one does a hypothesis test, two types of errors are possible. As a reminder, when the null hypothesis is true, and we reject it, we make a type I error. The probability of making a type I error is denoted by α, which is the significance level that we set for our hypothesis test. An α of 0.05 indicates that we are willing to accept a 5% chance that we are wrong when we reject the null hypothesis. To lower this risk, we must use a lower value for α. However, using a lower value for alpha means that we will be less likely to detect a true difference if one really exists. The probability of not rejecting the null hypothesis when it is true (not committing a type I error) is called the size of the test. When the null hypothesis is false, and we fail to reject it, we make a type II error. The probability of making a type II error is called beta, and is often denoted by β. We can decrease our risk of committing a type II error by ensuring our test has enough power. We can do this by ensuring our sample size is large enough to detect a practical difference when one truly exists. The probability of rejecting the null hypothesis when it is false (not committing a type II error) is called the power of the test.

In order to understand the interrelationship between the type error I and II, in this case, consider the following. As we pointed out before, feasible test statistics were introduced, and a one-sided test was proposed for the purpose of testing the hypothesis of chaos (eq.1) based on the theoretical asymptotic properties of the neural net estimator. Under the null hypothesis H_0 that the data-generating process is chaotic, the neural net estimator $\hat{\lambda}_k$ leads to asymptotically valid inferences in that the associated p-value follows a normal distribution on $N(0, \hat{\varphi}_k)$. Hence, our interest will be to test the null hypothesis $H_0 : \hat{\lambda}_k > 0$ against the alternative $H_1 : \hat{\lambda}_k \leqslant 0$. We will reject the null hypothesis if $\hat{t}_k \leqslant -z_\alpha$, where z_α is the critical value that satisfies $\Pr[Z \geqslant z_\alpha] = \alpha$ with Z being a standard normal random variable and α is the significance level. Rejecting the null hypothesis $H_0 : \hat{\lambda}_k > 0$ means lack of chaotic behavior. That is, the data-generating process does not have a chaotic attractor because it does not show the property of sensitivity to initial conditions. Thus, we have used these results to calculate the standard error of the Lyapunov exponent estimator and investigate the statistical significance of the sign of the exponents in order to test the reliability of the proposed algorithms.

$n = 50$	$s = 0$	$s = 0.01$	$s = 0.02$	$s = 0.03$	$s = 0.04$	$s = 0.05$
Logistic map	2.40	2.60	3.60	3.50	3.30	4.10
Gauss map	2.70	3.60	4.20	4.70	4.30	4.70
Hénon system	2.90	3.70	3.90	4.00	4.50	4.30
Rössler system	3.10	3.50	4.70	4.90	4.50	5.10
$n = 100$						
Logistic map	1.20	1.50	2.30	2.50	3.40	3.90
Gauss map	1.50	1.70	1.90	2.70	3.10	3.50
Hénon system	1.70	1.90	2.20	3.30	3.70	4.00
Rössler system	1.70	1.80	2.30	4.10	3.70	4.70
$n = 200$						
Logistic map	0.30	0.50	1.20	1.40	1.50	2.10
Gauss map	0.50	0.80	0.90	1.10	1.40	2.50
Hénon system	0.40	0.60	0.70	1.80	1.90	2.20
Rössler system	0.50	1.10	1.30	1.70	2.50	3.30

Table 5: Rejection percentages at the 5% significance level from the dynamic systems considered with a chaotic behaviour. These results provide the size of our hypothesis test.

Note that the results shown in tables 5-6 are given in terms of rejection percentages of the tests at the 5% significance level over 1000 Monte Carlo replications. The rejection percentages give an indication of the size of the tests (tab.5) and the power of the tests (tab.6). We have chosen $n = 50, 100, 200$ as we want to check the reliability of the algorithms in small sample sizes. The four dynamic systems used are listed in tab.3. Table 5 provides the results from chaotic dynamic systems which have the following parameter set values: $\mu = 4$ (logistic map); $\alpha = 6.2, \beta = -0.5$ (Gauss map); $a = 1.4, b = 0.3$ (Hénon system), and $a = 0.2, b = 0.2, c = 5.7$ (Rössler system). Table 6 gives the results from non-chaotic dynamic systems which have the following parameter set values: $\mu = 3.2$ (logistic map); $\alpha = 4.9, \beta = -0.58$ (Gauss map); $a = 1.2, b = 0.1$ (Hénon system), and $a = 0.1, b = 0.1, c = 7$ (Rössler system).

The data given in tables 5-6 provide the following comments. First, we can point out that the reliability of the tests is solid at the 5% significance level. The rejection percentages in almost every situation, even for $n = 50$, are low when the null hypothesis H_0 is true and high when H_0 is false. Second, the results provide satisfactory performance in moderate sample sizes. This fact is really important for those researchers who work with short time series data. Third, the empirical size decrease and the empirical power increase as the sample size increase which means that our tests are consistent as well. Fourth, the noise-contaminated data are comparable to those which are noise-free. Although as the noise increases, the errors committed increase but not significantly. Fifth, the results shown are fairly robust with respect to the parameter values of the four dynamic systems considered.

Conclusion

The main feature of chaos is the well-known initial-value sensitivity property. The study of measures to quantify the initial-value sensitive property began with the introduction of Lyapunov exponents. So quantifying chaos through this kind of quantitative measure is a key point for understanding chaotic behavior. This paper describes the main statistical estimation methods of the Lyapunov exponent from time series data. At the same time, we present the **DChaos** library. This package provides an R interface for chaotic time series data analysis. R itself and all packages used are available from the Comprehensive R Archive Network (CRAN) at `https://CRAN.R-project.org/`.

R users may compute the delayed-coordinate embedding vectors from time series data, estimates the best neural net model, calculates the partial derivatives directly from the chosen neural network model. Finally, they can estimate both the largest Lyapunov exponent through the Norma-2 procedure and the Lyapunov exponent spectrum through the QR decomposition procedure taking into account the full sample and three different methods of subsampling by blocks. The results provided by Shintani and Linton (2004) have enabled us to obtain a consistent Lyapunov exponent estimator and test the chaotic hypothesis based on the theoretical asymptotic properties of our neural net estimators.

Finally, looking to the future, our focus will be on extending the functionality of this package. Let us remark that the different contributions that compose the Jacobian indirect methods differ in

$n = 50$	$s = 0$	$s = 0.01$	$s = 0.02$	$s = 0.03$	$s = 0.04$	$s = 0.05$
Logistic map	99.30	98.90	98.40	97.70	97.50	96.90
Gauss map	99.40	99.00	98.70	98.50	97.10	97.00
Hénon system	99.20	98.90	98.60	97.90	97.40	96.90
Rössler system	98.70	98.10	97.40	97.00	96.70	96.20
$n = 100$						
Logistic map	99.70	99.00	98.70	98.20	97.70	97.20
Gauss map	99.70	99.30	98.90	98.70	98.40	97.70
Hénon system	99.60	99.00	98.80	98.20	97.70	97.10
Rössler system	99.40	98.80	98.30	97.70	97.20	96.90
$n = 200$						
Logistic map	99.90	99.40	99.00	98.70	98.20	97.90
Gauss map	100.00	99.60	99.30	99.00	98.80	98.20
Hénon system	99.80	99.30	99.00	98.60	98.10	97.80
Rössler system	99.70	99.00	98.70	98.20	97.50	97.00

Table 6: Rejection percentages at the 5% significance level from the dynamic systems considered with a non-chaotic behaviour. These results provide the power of our hypothesis test.

the algorithm used for the estimation of the Jacobian. So far, we have focused on the neural net approach, which tries to estimate the underlying dynamic system without imposing the restriction of local linearity. We are working on three more Jacobian indirect methods that try to estimate the Jacobian by local polynomial kernel regressions, by local neural nets models, and by convolutional neural nets models. We are interested in developing the proposed algorithms from multivariate time series data. We are also focusing on how to apply parallelization and big data techniques to the design of the proposed algorithms in order to obtain computational efficiency when applying them to massive databases. We will keep providing useful tools and robust algorithms related to the analysis of chaotic time-series data.

Acknowledgment

This work was supported by the Government of Spain (grant RTI2018-094901-B-I00); the Complutense University of Madrid (Faculty of Statistical Studies); the San Pablo CEU University (Faculty of Economics); and the Data Analysis in Social and Gender Studies and Equality Policies Research Group (www.ucm.es/aedipi/).

Bibliography

H. D. Abarbanel. *Analysis of observed chaotic data*. Springer, 1996. [p234, 236]

D. W. K. Andrews. Heteroskedasticity and autocorrelation consistent covariance matrix estimation. *Econometrica*, 59(3):817–858, 1991. URL https://doi.org/10.2307/2938229. [p245]

E. Bradley and H. Kantz. Nonlinear time-series analysis revisited. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 25(9):097610, 2015. URL https://doi.org/10.1063/1.4917289. [p232]

R. Brown, P. Bryant, and H. D. Abarbanel. Computing the lyapunov spectrum of a dynamical system from an observed time series. *Physical Review A*, 43(6):2787, 1991. URL https://doi.org/10.1103/PhysRevA.43.2787. [p238]

K.-S. Chan and H. Tong. *Chaos: a statistical perspective*. Springer-Verlag, 2001. URL https://doi.org/10.1007/978-1-4757-3464-5. [p234, 236]

W. D. Dechert and R. Gençay. Lyapunov exponents as a nonparametric diagnostic for stability analysis. *J Appl Econ*, 7(S1):S41–S60, dec 1992. URL https://doi.org/10.1002/jae.3950070505. [p238]

J. P. Eckmann and D. Ruelle. Ergodic theory of chaos and strange attractors. *Rev Mod Phys*, 57:617–656, 1985. URL https://doi.org/10.1103/RevModPhys.57.617. [p233, 238]

J. P. Eckmann, S. O. Kamphorst, D. Ruelle, and S. Ciliberto. Liapunov exponents from time series. *Phys Rev A*, 34:4971–4979, Dec 1986. URL https://doi.org/10.1103/PhysRevA.34.4971. [p238]

M. Faggini. Chaotic time series analysis in economics: Balance and perspectives. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 24(4):042101, 2014. URL https://doi.org/10.1063/1.4903797. [p232]

A. Fernández-Díaz. *Chaos theory: Current and future research and applications*. McGraw-Hill, 2019. [p232]

C. A. Garcia. *nonlinearTseries: Nonlinear Time Series Analysis*, 2019. URL https://CRAN.R-project.org/package=nonlinearTseries. R package version 0.2.6. [p233]

R. Gençay and W. D. Dechert. An algorithm for the n lyapunov exponents of an n-dimensional unknown dynamical system. *Physica D*, 59(1):142–157, 1992. URL https://doi.org/10.1016/0167-2789(92)90210-E. [p232, 233, 238]

P. Grassberger and I. Procaccia. Measuring the strangeness of strange attractors. *Physica D: Nonlinear Phenomena*, 9(1-2):189–208, 1983. URL https://doi.org/10.1007/978-0-387-21830-4_12. [p236]

R. Hegger, H. Kantz, and T. Schreiber. Practical implementation of nonlinear time series methods: The tisean package. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 9(2):413–435, 1999. URL https://doi.org/10.1063/1.166424. [p233, 237]

K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359 – 366, 1989. URL https://doi.org/10.1016/0893-6080(89)90020-8. [p239]

J. P. Huke and D. S. Broomhead. Embedding theorems for non-uniformly sampled dynamical systems. *Nonlinearity*, 20(9):2205–2244, 2007. URL https://doi.org/10.1088/0951-7715/20/9/011. [p234, 235]

H. Kantz. A robust method to estimate the maximal lyapunov exponent of a time series. *Physics Letters A*, 185(1):77 – 87, 1994. URL https://doi.org/10.1016/0375-9601(94)90991-1. [p233, 237]

H. Kantz and T. Schreiber. Determinism and predictability. *Nonlinear time series analysis*, pages 42–57, 1997. [p237]

H. Kantz and T. Schreiber. *Nonlinear time series analysis*, volume 7. Cambridge university press, 2004. [p234, 236, 237]

M. B. Kennel, R. Brown, and H. D. I. Abarbanel. Determining embedding dimension for phase-space reconstruction using a geometrical construction. *Phys Rev A*, 45:3403–3411, 1992. URL https://doi.org/10.1103/PhysRevA.45.3403. [p236]

Z. Lu and R. L. Smith. Estimating local lyapunov exponents. *Fields Institute Communications*, 11: 135–151, 1997. [p238]

D. F. McCaffrey, S. Ellner, A. R. Gallant, and D. W. Nychka. Estimating the lyapunov exponent of a chaotic system with nonparametric regression. *J Am Stat Assoc*, 87(419):682–695, 1992. URL https://doi.org/10.1080/01621459.1992.10475270. [p233, 238]

A. F. D. Narzo. *tseriesChaos: Analysis of Nonlinear Time Series*, 2019. URL https://CRAN.R-project.org/package=tseriesChaos. R package version 0.1-13.1. [p233]

D. Nychka, S. Ellner, A. R. Gallant, and D. McCaffrey. Finding chaos in noisy systems. *J R Stat Soc Series B Stat Methodol*, 54(2):399–426, 1992. URL https://www.jstor.org/stable/2346135. [p233, 238]

M. T. Rosenstein, J. J. Collins, and C. J. D. Luca. A practical method for calculating largest lyapunov exponents from small data sets. *Physica D: Nonlinear Phenomena*, 65(1):117 – 134, 1993. URL https://doi.org/10.1016/0167-2789(93)90009-P. [p237]

D. Ruelle and F. Takens. On the nature of turbulence. *Communications in Mathematical Physics*, 20(3): 167–192, 1971. URL https://doi.org/10.1007/BF01646553. [p234]

J. E. Sandubete and L. Escot. *DChaos: Chaotic Time Series Analysis*, 2021. URL https://CRAN.R-project.org/package=DChaos. R package version 0.1-6. [p233]

M. Sano and Y. Sawada. Measurement of the lyapunov spectrum from a chaotic time series. *Physical review letters*, 55(10):1082, 1985. URL https://doi.org/10.1103/PhysRevLett.55.1082. [p238]

M. Shintani and O. Linton. Is there chaos in the world economy? a nonparametric test using consistent standard errors. *International Economic Review*, 44(1):331–357, 2003. URL https://doi.org/10.1111/1468-2354.t01-1-00073. [p238, 240]

M. Shintani and O. Linton. Nonparametric neural network estimation of Lyapunov exponents and a direct test for chaos. *J Econom*, 120(1):1–33, may 2004. URL https://doi.org/10.1016/S0304-4076(03)00205-7. [p233, 238, 242, 245, 249]

F. Takens. *Detecting strange attractors in turbulence*. Springer Berlin Heidelberg, 1981. [p234, 236]

L. Tang, H. Lv, F. Yang, and L. Yu. Complexity testing techniques for time series data: A comprehensive literature review. *Chaos Solitons & Fractals*, 81:117–135, 2015. URL https://doi.org/10.1016/j.chaos.2015.09.002. [p232]

Y.-J. Whang and O. Linton. The asymptotic distribution of nonparametric estimates of the lyapunov exponent for stochastic time series. *J Econom*, 91(1):1 – 42, 1999. URL https://doi.org/10.1016/S0304-4076(98)00047-5. [p238]

A. Wolf, J. B. Swift, H. L. Swinney, and J. A. Vastano. Determining lyapunov exponents from a time series. *Physica D*, 16(3):285 – 317, 1985. URL https://doi.org/10.1016/0167-2789(85)90011-9. [p237]

D. Wuertz, T. Setz, and Y. Chalabi. *fNonlinear: Rmetrics - Nonlinear and Chaotic Time Series Modelling*, 2017. URL https://CRAN.R-project.org/package=fNonlinear. R package version 3042.79. [p233]

Julio E. Sandubete
Faculty of Statistical Studies
Complutense University of Madrid
28040 Madrid, Spain
ORCiD: 0000-0002-1518-0417
E-mail: jsandube@ucm.es

Lorenzo Escot
Faculty of Statistical Studies
Complutense University of Madrid
28040 Madrid, Spain
ORCiD: 0000-0002-6734-6327
E-mail: escot@ucm.es

IndexNumber: An R Package for Measuring the Evolution of Magnitudes

by Alejandro Saavedra-Nieves and Paula Saavedra-Nieves

Abstract Index numbers are descriptive statistical measures useful in economic settings for comparing simple and complex magnitudes registered, usually in two time periods. Although this theory has a large history, it still plays an important role in modern today's societies where big amounts of economic data are available and need to be analyzed. After a detailed revision on classical index numbers in literature, this paper is focused on the description of the R package **IndexNumber** with strong capabilities for calculating them. Two of the four real data sets contained in this library are used for illustrating the determination of the index numbers in this work. Graphical tools are also implemented in order to show the time evolution of considered magnitudes simplifying the interpretation of the results.

Introduction

The problem of reducing a large amount of available microeconomic data is common in dynamic and modern economies. Individuals of today's societies consume services of hundred of commodities over a year, and most producers use and produce hundreds of individual products and services. This overwhelming abundance of data is usually summarized through index numbers theory. *Index numbers* are descriptive statistical measures useful in order to compare or measure changes in simple and complex magnitudes over time. The goal is usually to determine possible increases or decreases and, more generally, trend changes. The situations to be compared are in no way restricted; they may be two time periods (hours, days, months, or years); two places (two cities or countries); or two groups of people (economically active and inactive population). For simplicity in the exposition, we refer to temporal index numbers in this paper. The rest of the situations mentioned could be considered modifying the notation slightly.

Although index numbers are considered a classical statistical tool, the problem of how to construct them is as much one of economic theory as of statistical technique, see Frisch (1936). This can be checked by analyzing their considerable history. Initial works involving index numbers date back to the early 18th century. During the creation of a college in 1440–1460, it was stipulated that any member had to leave it if his richness exceeded five pounds per year. The Anglican Bishop Fleetwood desired to know if, according to the price evolution, this promise could be kept three centuries later. Then, he studied the evolution of prices corresponding to four products (wheat, meat, drink, and clothing) from 1440 to 1700. He concluded that five pounds in 1440–1460 had the same value as 30 pounds in 1700. More details can be found in Fleetwood (1707). Dûtot (1754) studied the diminution of the money value analyzing the incomes of kings Louis XII and Louis XV. In order to know which of them had the largest disposable income, he considered the prices of several goods of different nature as such as a chicken, a rabbit, a pigeon, or the day's work value. From the discovery of the Americas, the astronomy professor of Padoue analyzed the evolution of prices in 1764. He considered the prices of three commodities (grains, wine, and cooking oil), and he studied their variation from 1500 to 1750. Evelyn (1798) can be seen as a precursor of index numbers establishing a price index number from 1050 to 1800. In this work, notions as the year of reference and relative prices were introduced. According to Kendall (1977), the real father of index numbers is Joseph Lowe. Many problems dealing with their construction were presented in Lowe (1822). In fact, Lowe's measurement would be known as the Laspeyres index later. In the second half of the 19th century, statisticians developed many advances in this setting. Jevons (1863) recommended considering the geometrical mean in order to construct an index number. Between 1864 and 1880, Laspeyres (1864, 1871), Drobisch (1871), and Paasche (1874) worked on the evolution of prices for material goods from the approach of weighted index numbers. Palgrave (1886) proposed to weigh the relative prices by the total amount of the considered good. Fisher (1922) defined a new index number calculated as the geometric mean of the Laspeyres and the Paasche index numbers. In the same period, Marshall and Edgeworth proposed to calculate weighted means. For an in-depth review on this topic, see Kendall (1969), Allen (1982) or more recently, CPI Manual (2004), or Dodge (2008).

Nowadays, economists continue to use index numbers to make comparisons over time despite their long background. In fact, the main applications of index numbers are either economic, or they occur in related fields as demography or technology. In such settings, the magnitudes to be compared through index numbers usually come in pairs, one of price and the other a matching one of quantity. This pair may be designed to account for the variation in an aggregate value, as when movements in the aggregated expenditure of consumers are analyzed into the two components of changes in

prices and in real consumption. Some more recent contributions in index numbers theory are exposed next. Barnett (1980) focused on economic monetary aggregates from this approach. Changes in food prices were analyzed in Lamm (1980). Index numbers in chain, or more commonly chain indices, are considered in Forsyth and Fowler (1981). Boyle (1988) analyzed the volume of Irish agricultural output from 1960 to 1982. Scanner data on coffee sales are studied in de Haan and Opperdoes (1997). Hill (1999) shows how a comparison of price levels across a group of countries can be made by chaining Fisher index numbers across a spanning tree. Inequality and poverty in several regions of Thailand are studied through the construction of urban and rural cost of living and welfare indices in Kakwani and Hill (2002). Dumagan (2002) showed that the Fisher index could be numerically approximated by other superlative index numbers. In Reinsdorf et al. (2002), additive decompositions of the Fisher index are derived in order to know how much each item contributes to its overall change. From data from United States, Canada, France, Germany, Italy, Japan, and the United Kingdom, drug price and quantity index numbers are considered in Danzon and Chao (2000). Ang et al. (2004) used a generalized version of the Fisher index to analyze CO_2 emission. In Boyd and Roop (2004), the structural change in energy intensity is studied. Exploring the duality between a return to dollar definition of profit and the generalized distance function, the relationship between the Laspeyres, Paasche and Fisher productivity index numbers is established in Zofío and Prieto (2006). Hill (2006) showed an illustration on index numbers also using scanner data. An application to major crops in Manitoba is presented in Coyle (2007). According to Diewert and Nakamura (2007), Paasche, Laspeyres, or Fisher index number formula is useful in order to manage the total factor productivity growth. The importance of the hedonic imputation method in price index numbers is analyzed in Hill and Melser (2008) from a data set containing house prices for three regions in Sydney over a three years period. The impact of time aggregation on price change estimates for several supermarket item categories is considered in Ivancic et al. (2011). Białek (2012) proposed a general price index formula with the Fisher, Laspeyres, and Paasche indices as its particular cases. Białek (2014) presented an original price index, and its performance is analyzed through a simulation study where it is compared to several classical price index numbers. A generalized version of the Fisher index is considered in Su and Ang (2014) in order to analyze changes in the carbon emissions embodied in China's exports. O'Donnell (2018) analyzed the productivity change defined as measures of output quantity change divided by measures of input quantity change. Zhen et al. (2019) constructed panel price index numbers using retail scanner data in order to compare consumption costs across space and time.

This paper is focused on the description of the R package **IndexNumber** (Saavedra-Nieves and Saavedra-Nieves, 2021), available from the *Comprehensive R Archive Network* at `https://CRAN.R-project.org/package=IndexNumber`, and its capabilities for calculating classical index numbers. It is organized as follows. Index numbers are formally defined in Section 2.2. Section 2.2.1 introduces simple index numbers. Concretely, simple index numbers in series and in chain are distinguished. In Section 2.2.2, non-weighted and weighted complex index numbers are presented. Details on the usage of **IndexNumber** package are considered from Section 2.3. The four real data sets contained in the library are also described briefly. Note that two of them are used in this paper in order to illustrate the calculation of index numbers. They are available on the website of the *Spanish Statistical Office (INE)*, `http://www.ine.es`.

Preliminaries on index numbers

Index numbers are statistical measures that are useful to compare single and multiple magnitudes for the same interval of time. In both cases, this comparison is made with respect to an element of the mentioned series that is called base period or reference. Some examples of simple magnitudes are prices of a good, sold amounts of a product, or other general individual values. However, most of the time, comparing these simple quantities has not practical interest. If the goal is to analyze some real phenomena where many variables are involved, complex indices must be considered. Using these ideas, index numbers are usually classified into the following two groups:

- An index number is said to be simple if it corresponds to the ratio of two values of the same variable, measured in two different instants. Therefore, a *simple index number* provides the variation that the single magnitude has suffered between two different time periods.
  For instance, a simple index number of the price will give the relative variation of the price between the current period and the period taken as reference.
- Most of the time, comparing prices, amounts, or values of a single product individually is not of interest in practice. If the goal is to analyze some real situations where different variables have influence, a *complex number index* has to be considered. It globally summarizes the information of the different magnitudes involved in the problem.
  For instance, the evolution cost of life in a country is a common case where it is necessary to select a set of goods or variables that give information about it. The relative importance of each of the goods considered must be measured and taken into account.

A wider overview of both classes of index numbers is included in the next sections. In particular, we distinguish the different subclasses belonging to each of them and their possible relationships.

Simple index numbers

A *simple index number* is a statistical indicator of the percentage of variation of a single magnitude in two different instants. Simple index numbers are usually classified according to the element that we take as reference. In particular, we distinguish two types of simple index numbers. First, we describe simple index numbers in series, when the first value of the series is taken as the reference value, and simple index numbers in chain (or chain indices), when the reference is the immediately previous value in the serie.

In what follows, we assume that $X = \{x_0, x_1, \ldots, x_T\}$ denotes the observations of the magnitude X for the $T+1$ time instants considered. Besides, x_0 is usually taken as the base period. Most common simple index numbers are individually referred to variables in real-world situations as the followings:

- the *price of a good*, denoted by p;
- the *amount of produced or sold product*, denoted by q; or
- the *value of a good*, denoted by v. This value is usually obtained as the product of the price and the amount variables.

In this section, we illustrate the usage of simple index numbers on a real example. Table 1 shows the number (thousands) of economically active women and men in Spain from the first trimester of 2002 (taken as a reference value). Remark that four trimesters of each year are denoted by T1, T2, T3 and T4, respectively.

Stages	2002 (T1)	2002 (T2)	2002 (T3)	2002 (T4)	2003 (T1)	2003 (T2)	...
Total of women	7442.70	7580.80	7670.20	7751.50	7868.70	7977.80	...
Total of men	11192.30	11289.40	11445.10	11472.80	11552.50	11661.40	...

Table 1: Number (thousands) of economically active women and men in Spain from first trimester of 2002.

This dataset, included in **IndexNumber** package, can be obtained from the Economically Active Population Survey (EPA) elaborated by the Spanish Statistics National Institute (INE).

Simple index numbers in series

Let x_0 and x_t be the values of the variable X where x_0 corresponds to the base period and $t \in \{0, 1, \ldots, T\}$, respectively. Thus, the value of the simple index number in series for X in t is defined as follows:

$$I_0^t(X) = \frac{x_t}{x_0} \cdot 100. \tag{1}$$

For each $t \in \{0, 1, \ldots, T\}$, this measure has a natural interpretation. Fixed a certain variable of interest X, the index number in series in t shows the percentage of variation of the magnitude in this instant of time with respect to the reference value (in this case, the initial one).

Using this type of index number, some usual magnitudes can be formally defined as it is indicated next:

- When prices are considered, the *relative price* of a product i, also called simple price index, can be determined as
$$p_0^t = \frac{p_{it}}{p_{i0}},$$
where p_{it} denotes the price at instant t and p_{i0}, the price in the base period.
- The *relative amount* of a product i can be written as
$$q_0^t = \frac{q_{it}}{q_{i0}},$$
where q_{it} denotes the produced or sold amount at instant t and q_{i0}, the amount for the base period.
- Finally, the *relative value* of a product i has the next expression:
$$v_0^t = \frac{v_t}{v_0} = \frac{p_{it}q_{it}}{p_{i0}q_{i0}} \cdot 100 = p_0^t \cdot q_0^t \cdot 100,$$
where v_t denotes the value of the good at instant t and v_0, the value of the base period.

Below, we illustrate the usage of simple index numbers on the two series included in Table 1. Thus, the simple index numbers in series for the economically active women and men in Spain from the first trimester of 2002 is given in Table 2.

Stages	2002 (T1)	2002 (T2)	2002 (T3)	2002 (T4)	2003 (T1)	2003 (T2)	...
Index number for women	100.00	101.86	103.06	104.15	105.73	107.19	...
Index number for men	100.00	100.87	102.26	102.51	103.22	104.19	...

Table 2: Simple index numbers in series for number (thousands) of economically active women and men in Spain from the first trimester of 2002.

Comparing the evolution of the index numbers in series for the population of women and men shown in Table 2 has an interest, for instance, to analyze the effect of variable sex in the Spanish labor market. Note that the number of economically active women and men in the second trimester of 2003 is 7.2% and 4.2% larger than in the reference time, respectively. Therefore, women increasing is slightly larger than men.

Simple index numbers in chain

Below, we introduce another approach of simple index numbers. Contrary to our previous assumptions, this new setting arises when the reference value is not the initial one; rather we take the value immediately preceding. Let x_t and x_{t-1} be two values of a variable X observed in two consecutive instants t and $t-1$, being $t \in \{1, 2, \ldots, T\}$. Thus, the value of the *index number in chain* or *chain index* (cf. Forsyth and Fowler, 1981) that corresponds to an instant t, with $t \in \{1, 2, \ldots, T\}$, is defined as follows:

$$IC^t(X) = \frac{x_t}{x_{t-1}} \cdot 100. \tag{2}$$

Again, this index number can be naturally interpreted. It is worth mentioning that these measures the variation of the characteristic under study with respect to the previous value in a fixed instant t. For instance, these index numbers reflect the percentage variation that the variable experiments between two consecutive values in time series settings.

To illustrate this definition, we take again the example considered in the previous section. For this subset of values, we obtain again the evolution of the amount of economically active people (per sex) in Spain under this new approach.

Stages	2002 (T1)	2002 (T2)	2002 (T3)	2002 (T4)	2003 (T1)	2003 (T2)	...
Index number for women	100.00	101.86	101.18	101.06	101.51	101.39	...
Index number for men	100.00	100.87	101.38	100.24	100.70	100.94	...

Table 3: Simple index numbers in chain for number (thousands) of economically active women and men in Spain from the first trimester of 2002.

Table 3 shows the evolution in time of the number (thousands) of economically active women and men in Spain from the trimester of 2002 (see Table 1). According to the obtained results, we emphasize as relevant that the number of economically active women and men in the second trimester of 2003 increases 1.4% and 0.9%, respectively, with respect to the previous trimester.

Relationship between simple index numbers in series and in chain

This section briefly introduces some comments on the relations between simple index numbers in series and in chain. In this way, one can be obtained from another (and vice versa) without having to use the exact values of the magnitude under study.

Take x_t the value of the variable X in instant t, with $t \in \{1, 2, \ldots, T\}$. Thus, a simple index in chain can be obtained from a simple index in series due to the relation

$$IC^t(X) = \frac{x_t}{x_{t-1}} \cdot 100 = \frac{\frac{x_t}{x_0}}{\frac{x_{t-1}}{x_0}} \cdot 100 = \frac{I_0^t(X)}{I_0^{t-1}(X)} \cdot 100.$$

For example, from simple index numbers in series for women shown in Table 2, it is possible to obtain the corresponding simple index number in chain for women also for the first trimester of 2003 (contained in Table 3). That is,

$$IC^{2003(T1)}(X) = \frac{7868.70}{7751.50} \cdot 100 = \frac{105.72}{104.15} \cdot 100 = 101.51.$$

Besides, simple indices in series can be equivalently obtained for each $t \in \{1, \ldots, T\}$ from indices

in chain:

$$I_0^t(X) = \frac{x_t}{x_0} \cdot 100 = \frac{x_t}{x_{t-1}} \frac{x_{t-1}}{x_{t-2}} \cdots \frac{x_2}{x_1} \frac{x_1}{x_0} \cdot 100 = \frac{IC^t(X)}{100} \cdot \frac{IC^{t-1}(X)}{100} \cdot \cdots \cdot \frac{IC^2(X)}{100} \cdot \frac{IC^1(X)}{100} \cdot 100.$$

For instance, if we consider the simple index numbers in chain shown in Table 3, it is possible to obtain the simple index number in series for the first trimester of 2003 contained in Table 2. That is, we check that

$$I_0^{2003(T1)}(X) = \frac{100.70}{100} \cdot \frac{100.24}{100} \cdot \frac{101.38}{100} \cdot \frac{100.87}{100} \cdot 100.$$

Variation Rate

In this section, we formally introduce another measure of the evolution of a magnitude. Furthermore, we relate it to the simple index numbers previously defined.

The *variation rate* of the observations in the instants t and $t-1$, with $t \in \{1, \ldots, T\}$, is denoted by $\text{Rate}^t(X)$ for each $t \in \{1, \ldots, T\}$. It can be calculated from the simple index in chain as follows:

$$\text{Rate}^t(X) = \frac{x_t - x_{t-1}}{x_{t-1}} \cdot 100 = IC^t(X) \cdot 100.$$

This definition can be extended to any pair of instants in $\{0, 1, 2, \ldots, T\}$. Take $t_1, t_2 \in \{0, 1, 2, \ldots, T\}$ such that $t_1 < t_2$. Let x_{t_1} be the value of a variable measure in instant t_1 and let x_{t_2} be the value of a variable measure in a later instant t_2. Formally, the *variation rate* of X in t_2 with respect to t_1 is

$$\text{Rate}_{t_1}^{t_2}(X) = \frac{x_{t_2} - x_{t_1}}{x_{t_1}} \cdot 100.$$

Note that

$$\text{Rate}_{t_1}^{t_2}(X) = \frac{x_{t_2} - x_{t_1}}{x_{t_1}} \cdot 100 = \left(\frac{x_{t_2}}{x_{t_1}} - 1 \right) \cdot 100 = I_{t_1}^{t_2}(X) - 100,$$

where $I_{t_1}^{t_2}(X) = \frac{x_{t_2}}{x_{t_1}} \cdot 100$, also satisfying

$$I_{t_1}^{t_2}(X) = IC^{t_2}(X) \cdot IC^{t_2-1}(X) \cdot \ \ldots \ \cdot IC^{t_1+1}(X).$$

In particular, if the consecutive observations correspond to two different years, months, or trimesters the variation rate is called interanual variation rate, monthly variation rate, and quarterly variation rate, respectively. From the data shown in Table 2, the quarterly variation rates have been calculated in Table 4.

Stages	2002 (T2)	2002 (T3)	2002 (T4)	2003 (T1)	2003 (T2)	...
Rate for women	1.86	1.18	1.06	1.51	1.39	...
Rate for men	0.87	1.38	0.24	0.70	0.94	...

Table 4: Quarterly variation rate for number (thousands) of economically active women and men in Spain.

Average Variation Rate

Finally, we introduce a third method to measure the evolution of a given magnitude X between the instants t and $t+k$, with $t \in \{0, \ldots, T-k\}$ and $k > 0$. If x_{t+k} denotes the observation at instant $t+k$ and x_t the corresponding to instant t, the *average variation rate* of the variable X between instants t and $t+k$ is defined as the constant rate T_k that allows obtaining the observation x_{t+k} at time $t+k$ from observation x_t at time t.

Then, it is possible to write:

$$x_{t+1} = x_t + \frac{T_k}{100} \cdot x_t = \left(\frac{100 + T_k}{100} \right) \cdot x_t$$

$$x_{t+2} = x_{t+1} + \frac{T_k}{100} \cdot x_{t+1} = \left(\frac{100 + T_k}{100} \right)^2 \cdot x_t$$

$$x_{t+3} = x_{t+2} + \frac{T_k}{100} \cdot x_{t+2} = \left(\frac{100 + T_k}{100} \right)^2 \cdot x_t$$

$$\vdots$$

$$x_{t+k} = x_{t+k-1} + \frac{T_k}{100} \cdot x_{t+k-1} = \left(\frac{100 + T_k}{100} \right)^k \cdot x_t$$

Therefore, $x_{t+k} = \left(\frac{100+T_k}{100}\right)^k \cdot x_t$ and, as consequence, $\frac{x_{t+k}}{x_t} = \left(\frac{100+T_k}{100}\right)^k$. Then,

$$T_k = 100 \cdot \left(\frac{x_{t+k}}{x_t}\right)^{1/k}.$$

From data shown in Table 2, we have calculated the quarterly average variation rate in 2002 for women and men as

$$\left(\sqrt[3]{\frac{7751.50}{7442.70}} - 1\right) \cdot 100 = 1.36 \text{ and } \left(\sqrt[3]{\frac{11472.80}{11192.30}} - 1\right) \cdot 100 = 0.83, \text{ respectively.}$$

According to the results obtained, the average variation rate for women is considerably bigger than the corresponding for men in this period.

Complex index numbers

Most of the time, comparing variables marginally does not provide real information. There are many phenomena in the real world that involve several variables. Of course, simple index numbers described in the previous section could be naturally applied for each of these variables separately. However, comparing these magnitudes can be not realistic and, for this reason, *complex index numbers* have to be introduced. A complex index number summarizes the information of the different marginal index numbers related to the set of variables of interest.

One of the most relevant examples that illustrates the use of complex index numbers is briefly described next. The evolution of living cost in a country is a common case where it is necessary to select a set of goods or variables that give information about it. The relative importance of each of the goods considered may be taken into account. This is the case of the *Consumer Price Index* (CPI) in Spain. The sets of good considered for calculating it follows the *International Classification of Consumption according to Purpose* (COICOP) prepared by the Statistical Division of the United Nations. This classification is also used by other countries. In this way, comparisons between different geographical areas make sense.

When all variables are not of equal relevance, it is possible to add complementary information for weighting each magnitude corresponding to its degree of importance. Depending on the use of this additional information, two classes of complex index numbers are distinguished in literature:

- In several practical cases, the relative weight of each involved variable has no interest. Fixed a magnitude, the class of required index numbers to be used in this setting are named *non-weighted complex index numbers*.
- The use of *weighted index numbers* allow greater importance to be attached to some items. For instance, real information other than simply the change in price over time can be used. Factors as quantity sold or quantity consumed for each item can also be considered.

In what follows, we assume that $X = (X_1, \ldots, X_n)$ denotes the collection of n magnitudes registered for n different products. For each $j = 1, \ldots, n$, $X_j = \{x_{j0}, x_{j1}, \ldots, x_{jT}\}$ denotes the observations of the magnitude X_j for the $T+1$ instants of time considered. Analogously to the simple case, we refer x_{j0} as the base period.

In practice, the available information can be summarized in a table such as Table 5.

Time \ Products	1	2	...	n
0	x_{10}	x_{20}	...	x_{n0}
1	x_{11}	x_{21}	...	x_{n1}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
T	x_{1T}	x_{2T}	...	x_{nT}

Table 5: Evolution of a set of magnitudes X from time 0 to T.

The most common complex indices jointly involve (some of) the variables in real-world situations that we enumerate below:

- the *price of a collection of n goods*, denoted by $p = (p_1, \ldots, p_n)$, where $p_j = \{p_{j0}, p_{j1}, \ldots, p_{jT}\}$ denotes the prices for $T+1$ instants and for each product $j = 1, \ldots, n$;
- the *amount of produced or sold products*, denoted by q, where $q_j = \{q_{j0}, q_{j1}, \ldots, q_{jT}\}$ denotes the amounts of product for $T+1$ instants and for each product $j = 1, \ldots, n$; or

- the *value of n goods* is given by $v = (v_1, \ldots, v_n)$. In this case, $v_j = \{v_{j0}, v_{j1}, \ldots, v_{jT}\}$ denotes the values of product j, with $j \in \{1, \ldots, n\}$ for $T + 1$ instants.

To illustrate the usage of complex index numbers, we take the example described in Table 6. It shows the unitary value (euros) of prices of combustibles and other energy resources for the main home in Spain from 2005 to 2015. In this case, the data source is again the Spanish Statistics National Institute (INE).

	Electricity (Kwh)	Natural municipal gas (m^3)	Liquified gas (kilo)	Liquified combustibles (litre)	Solid combustibles (kilo)
2006	0.14	0.70	1.00	0.69	0.12
2007	0.14	0.78	0.97	0.66	0.10
2008	0.15	0.83	1.03	0.81	0.11
2009	0.16	0.87	0.93	0.68	0.10
2010	0.17	0.79	1.04	0.72	0.12
2011	0.19	0.77	1.15	0.91	0.12
2012	0.22	0.82	1.19	0.96	0.12
2013	0.23	1.00	1.34	0.99	0.13
2014	0.24	1.07	1.35	0.89	0.15
2015	0.25	1.04	1.22	0.77	0.13

Table 6: Unitary value (euros) of combustibles and other energy resources for the main home in Spain from 2005 to 2015.

Table 7 shows the (thousands of units) consumption of combustibles and other energy resources for the main home in Spain from 2005 to 2015. This dataset is closely related to the prices presented in Table 6. Of course, this data set can also be obtained from the Spanish Statistics National Institute (INE).

	Electricity (Kwh)	Natural municipal gas (m^3)	Liquified gas (kilo)	Liquified combustibles (litre)	Solid combustibles (kilo)
2006	50623635	3617285	1057488	2297923	1306920
2007	51990501	3266575	1066857	2454265	1602799
2008	54990338	3473851	1210607	2274326	1556673
2009	59749470	3730349	1113642	2505711	1724222
2010	69751162	3954065	987112	2345215	1584123
2011	67574654	4466072	926824	1974662	1414234
2012	62878557	4576052	943632	2029733	1733591
2013	56017871	4116079	867695	1952593	2071152
2014	49177739	3653055	868743	2180866	2077766
2015	48541712	3795339	818183	2176533	2161208

Table 7: Consumption (thousands of units) of combustibles and other energy resources for the main home in Spain from 2005 to 2015.

Next, we formally describe non-weighted and weighted index numbers.

Non-weighted complex index numbers

Complex index numbers analyze several magnitudes that measure the evolution of a set of goods or services. The goal here is to find a statistical measure in order to summarize the information shown, for instance, in Table 5. In particular, knowing the variation of a magnitude in time t with respect to the base period has an interest. In this sense, it is worth mentioning that non-weighted complex index numbers can be easily obtained as the arithmetic, geometric and harmonic means of the simple index numbers in series for the considered magnitudes. Anyway, their mathematical expressions are described below:

- The *Sauerbeck index* (cf. Sauerbeck, 1895) at time t for X, $S^t(X)$, is calculated as the arithmetic mean of the simple index in series for the n involved magnitudes at t:

$$S^t(X) = \frac{1}{n} \sum_{i=1}^{n} \frac{x_{it}}{x_{i0}} \cdot 100, \text{ for each } t \in \{0, \cdots, T\}. \tag{3}$$

- The *Geometric mean index* at time t, $G^t(X)$, is calculated as follows:

$$G^t(X) = \sqrt[n]{\prod_{i=1}^{n} \frac{x_{it}}{x_{i0}}} \cdot 100, \text{ for each } t \in \{0, \cdots, T\}. \tag{4}$$

Given a collection of n magnitudes, the geometric mean index at t is obtained as n^{th}-root of the product of simple index numbers for X at time t. See more details in Jevons (1863).

- The *Harmonic mean index* at time t, $H^t(X)$, is determined by

$$H^t(X) = \frac{n}{\sum_{i=1}^{n} \frac{x_{i0}}{x_{it}}} \cdot 100, \text{ for each } t \in \{0, \cdots, T\}. \tag{5}$$

It is initially introduced in Jevons (1865) and Coggeshall (1886).

- The *Bradstreet-Dûtot index* at time t, $BD^t(X)$, is introduced in Walsh (1901). Its value is obtained as the ratio between the means of the magnitude in time t and the magnitude in the base period as follows:

$$BD^t(X) = \frac{\sum_{i=1}^{n} x_{it}}{\sum_{i=1}^{n} x_{i0}} \cdot 100, \text{ for each } t \in \{0, \cdots, T\}. \tag{6}$$

If X denotes the matrix p of prices of a set of goods or services along a period of time, these index numbers are specifically considered *non-weighted complex index numbers for prices*. Thus, it arises the *Sauerbeck index* at time t for prices, $S^t(p)$; the *Geometric mean index* at time t for prices, $G^t(p)$; the *Harmonic mean index* at time t for prices, $H^t(p)$; and the *Bradstreet-Dûtot index* at time t for prices, $BD^t(p)$. Their usage will be illustrated on the set prices of combustibles and other energy resources for the main home in Spain from 2005 to 2015. From the information in Table 6, the four index numbers previously described are numerically shown in Table 8, and their evolution is depicted in Figure 1.

t	$S^t(p)$	$G^t(p)$	$H^t(p)$	$BD^t(p)$
2006	100.00	100.00	100.00	100.00
2007	97.48	97.06	96.64	100.00
2008	107.56	107.08	106.60	110.57
2009	102.69	101.64	100.61	103.40
2010	108.53	108.26	108.00	107.17
2011	118.52	117.76	116.99	118.49
2012	126.48	124.97	123.48	124.91
2013	138.59	137.35	136.05	139.25
2014	142.65	141.66	140.70	139.62
2015	133.81	131.37	129.14	128.68

Table 8: Non-weighted complex price indexes in seriea for the unitary value of combustibles and energy resources for the main home in Spain from 2006 to 2015.

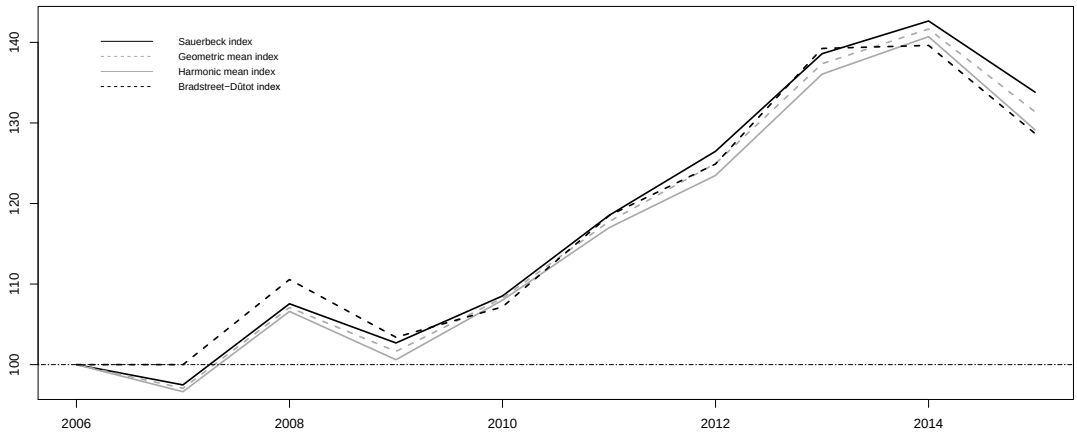


Figure 1: Joint evolution of the non-weighted complex price indexes in series for the unitary value of combustibles and energy resources for the main home in Spain from 2006 to 2015.

However, the variation of a given magnitude in time t with respect to the previous period may also be of interest. Below, we alternatively enumerate the mathematical expressions of the previous indices based on index numbers in chain for the magnitudes.

- The *Carli index* (cf. Carli, 1804) at time t for X, $C^t(X)$, is calculated as the arithmetic mean of the

simple index in chain for the n involved magnitudes at t:

$$C^t(X) = \frac{1}{n}\sum_{i=1}^{n}\frac{x_{it}}{x_{i\,t-1}}\cdot 100, \text{ for each } t \in \{1,\cdots,T\}. \tag{7}$$

- The *Jevons index* at time t, $J^t(X)$, is calculated following Jevons (1863) as follows:

$$J^t(X) = \sqrt[n]{\prod_{i=1}^{n}\frac{x_{it}}{x_{i\,t-1}}}\cdot 100, \text{ for each } t \in \{1,\cdots,T\}. \tag{8}$$

  Given a collection of n magnitudes, the geometric mean index at t is obtained as n^{th}-root of the product of simple index numbers for X in chain at time t.
- The *Dûtot index* at time t, $D^t(X)$, is introduced in Dûtot (1754). Its value is obtained as the ratio between the means of the magnitude in time t and the magnitude in time $t-1$ as follows:

$$D^t(X) = \frac{\sum_{i=1}^{n} x_{it}}{\sum_{i=1}^{n} x_{i\,t-1}}\cdot 100, \text{ for each } t \in \{1,\cdots,T\}. \tag{9}$$

Again, if X denotes the matrix of prices of a set of goods or services along a period of time (p), these index numbers are named as the *Carli index* at time t for prices, $C^t(p)$; the *Jevons index* at time t for prices, $J^t(p)$; and the *Dûtot index* at time t for prices, $D^t(p)$. To conclude this section, we obtain them for the set prices of combustibles and other energy resources for the main home in Spain from 2005 to 2015, detailed in Table 6. These index numbers are numerically detailed in Table 9.

t	$C^t(p)$	$J^t(p)$	$D^t(p)$
2006	100.00	100.00	100.00
2007	101.11	101.09	101.79
2008	113.51	113.43	115.89
2009	100.94	100.18	126.56
2010	105.75	105.27	144.02
2011	103.58	103.29	155.49
2012	110.65	110.49	168.01
2013	107.00	106.32	163.98
2014	100.72	100.39	154.09
2015	93.08	92.79	153.29

Table 9: Non-weighted complex price indexes in chain for the unitary value of combustibles and energy resources for the main home in Spain from 2006 to 2015.

All of the index numbers described are easy to be computed. However, they present an important disadvantage: they do not take into account the relative importance of each product.

Weighted complex index numbers

For analyzing the evolution of a given magnitude X, it is very common to use an alternative magnitude Y through the value of Y in the reference or the actual period to weight complex index numbers. The information relative to this alternative variable can be summarized in a table such as Table 10. For instance, the use of the amount of production of different products or the use of prices may result of interest, depending on the setting under study.

Time \ Products	1	2	...	n
0	y_{10}	y_{20}	...	y_{n0}
1	y_{11}	y_{21}	...	y_{n1}
⋮	⋮	⋮	⋮	⋮
T	y_{1T}	y_{2T}	...	y_{nT}

Table 10: Evolution of a set of magnitudes Y from time 0 to T.

Next, the main weighted complex price index numbers for a given magnitude X, taking Y as weight, are formally described:

- The *Laspeyres index* (Laspeyres, 1871) analyzes the variations of X using Y as weight. In this sense, the weights considered for product i are $x_{i0}\cdot y_{i0}$ (note that both values are referred to

the base period). Then, this complex index is defined as the weighted arithmetic means of the simple index numbers:

$$L^t(X,Y) = \frac{\sum_{i=1}^n \frac{x_{it}}{x_{i0}} x_{i0}y_{i0}}{\sum_{i=1}^n x_{i0}y_{i0}} \cdot 100 = \frac{\sum_{i=1}^n x_{it}y_{i0}}{\sum_{i=1}^n x_{i0}y_{i0}} \cdot 100, \text{ for each } t \in \{0, \cdots, T\}. \quad (10)$$

The main disadvantage of the Laspeyres index is that it assumes that the weights do not vary in time. This hypothesis is not always realistic in some practical settings.

- The *Paasche index* is an alternative index to the Laspeyres index introduced in Paasche (1874), when the weighted criteria is $x_{i0} \cdot y_{it}$. Therefore, it can be formally written as:

$$P^t(X,Y) = \frac{\sum_{i=1}^n \frac{x_{it}}{x_{i0}} x_{i0} \cdot y_{it}}{\sum_{i=1}^n x_{i0} \cdot y_{it}} \cdot 100 = \frac{\sum_{i=1}^n x_{it}y_{it}}{\sum_{i=1}^n x_{i0}y_{it}} \cdot 100, \text{ for each } t \in \{0, \cdots, T\}. \quad (11)$$

- The *Marshall-Edgeworth index* (cf. Marshall, 1887, Edgeworth, 1887) is an agregative weighted measure where weights are $y_{i0} + y_{it}$. Therefore, it can be calculated as:

$$E^t(X,Y) = \frac{\sum_{i=1}^n x_{it}(y_{i0} + y_{it})}{\sum_{i=1}^n x_{i0}(y_{i0} + y_{it})} \cdot 100, \text{ for each } t \in \{0, \cdots, T\}. \quad (12)$$

- The *Fisher index* is equal to the geometric mean of the index numbers under the approaches of Laspeyres and Paasche:

$$F^t(X,Y) = \sqrt{L^t(X,Y) \cdot P^t(X,Y)}, \text{ for each } t \in \{0, \cdots, T\}. \quad (13)$$

For instance, see more details in Fisher (1922).

Note that other values can be defined (as we will see below). The choice of using a specific index formula often relies on the availability of data. According to the previous comments, the Laspeyres index does not require information on the products of the current period. Then, the Laspeyres formula is usually preferred for the calculation of complex indices, which are typically released rapidly before information for the current period could have been collected.

In what follows, p denotes the matrix of prices of a set of goods or services along a period of time and q is the matrix of the total amounts of goods in the same period. Thus, the *weighted complex price index numbers* analyze the time evolution of prices by introducing the variation of the physical production or the consumption of a set of goods or services. The weights are obtained by multiplying the price of a product in an instant of time t by the consumption in the base period or the actual period. Hence, the *Laspeyres price index*, $L^t(p,q)$, the *Paasche price index*, $P^t(p,q)$, the *Marshall-Edgeworth price index*, $E^t(p,q)$, and the *Fisher price index*, $F^t(p,q)$, are naturally defined in prices settings.

Under this approach, a new complex index for v can be naturally introduced under the approach of the Bradstreet-Dûtot index. It is based on the notion of the value of good indicated by v. It can be calculated as follows:

$$IV_0^t(p,q) = \frac{V_t(p,q)}{V_0(p,q)} = \frac{\sum_{i=1}^n p_{it}q_{it}}{\sum_{i=1}^n p_{i0}q_{i0}}, \text{ for each } t \in \{0, \cdots, T\}. \quad (14)$$

It satisfies that $IV_0^t(p,q) = L^t(p_0,q) \cdot P^t(p_0,q) = L^t(p,q_0) \cdot P^t(p,q_0) = F^t(p_0,q) \cdot F^t(p,q_0)$.

From data contained in Tables 6 and 7, these five index numbers were determined. They are in Table 11 (from tenth to twelfth column), and they are graphically depicted in Figure 2.

t	$L^t(p,q)$	$P^t(p,q)$	$E^t(p,q)$	$F^t(p,q)$	$IV_0^t(p,q)$
2006	100.00	100.00	100.00	100.00	100.00
2007	101.31	100.99	101.15	101.15	101.79
2008	110.23	109.89	110.06	110.06	115.89
2009	112.11	112.06	112.09	112.09	126.56
2010	115.76	116.69	116.27	116.22	144.02
2011	127.77	128.35	128.08	128.06	155.49
2012	142.72	143.32	143.04	143.02	168.01
2013	153.98	154.43	154.21	154.20	163.98
2014	158.54	158.62	158.58	158.58	154.09
2015	158.20	158.23	158.21	158.21	153.29

Table 11: Weighted complex price indexes for the unitary value of combustibles and energy resources for the main home in Spain from 2006 to 2015.

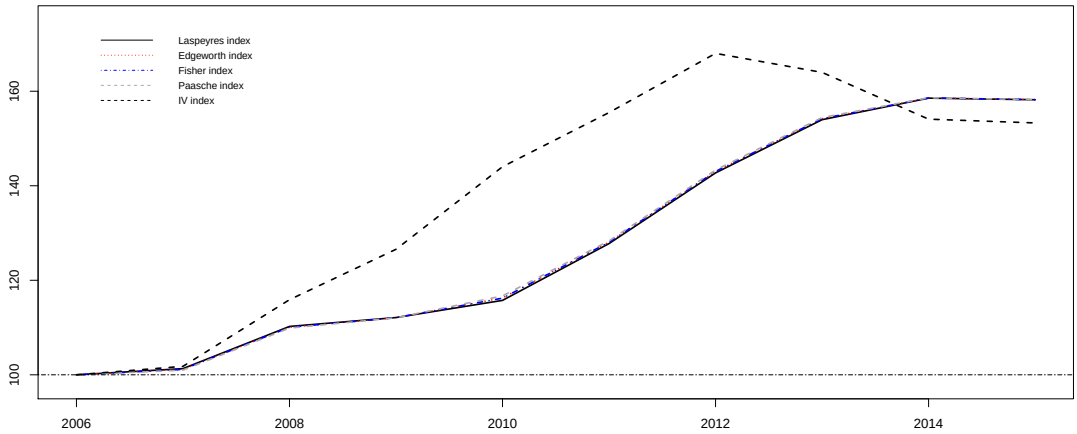


Figure 2: Joint evolution of the weighted complex price indexes for the unitary value of combustibles and energy resources for the main home in Spain from 2006 to 2015.

Otherwise, the *weighted complex production index numbers* analyze the time evolution of the amount of product by introducing the variation of the price of the goods or services as weight. Their obtaining is analogous to the previous one. The weights are obtained by multiplying the amount of a product in an instant of time t by the prince in the base period or the actual period. Thus, we dealt with the *Laspeyres production index*, $L^t(q,p)$; the *Paasche production index*, $P^t(q,p)$; or the *Fisher production index* $F^t(q,p)$.

IndexNumber in practice

This section presents an overview of the structure of the package. **IndexNumber** is a tool that R users can use in order to determine several classical index numbers that describe the evolution of a single magnitude or a set of magnitudes. This software helps the user to calculate faster these statistical measures. Functions in this library automatize the required operations for the computation of index numbers. First, we will describe the real data sets included in the package. Then, the functions implemented are detailed. Of course, other libraries exist in R dealing with index numbers theory. In particular, **micEconIndex** (Henningsen, 2017), **IndexNumR** (White, 2021), and **PriceIndices** (Bialek, 2021) packages also allow to compute complex index numbers but only when the considered magnitudes are prices and quantities. It is worth mentioning that **IndexNumber** library was designed under a more general perspective by extending to any type of magnitude. Moreover, none of the above-referenced packages implement simple index numbers, and they do not offer graphical tools to facilitate the analysis of time evolution series either. Additionally, **IndexNumber** can be seen as an additional basic library that can also be exploited by non-experts R users. For instance, inputs of functions are numeric vectors or matrices containing the magnitude values, much more flexible than data structures that other packages consider. As for computational complexity, it is also relatively simple because, unlike **IndexNumR** that implements multilateral methods, the number of required elementary operations is smaller. Finally, it is convenient to note that **IndexNumber** package also provides four new recent real data sets.

Data sets in IndexNumber

Index numbers have been theoretically introduced in previous sections using two real data sets included in the package **IndexNumber**. However, we decide to include them in the package because they could be used directly by the users, avoiding search and download. Besides, two additional data sets were also included. All of them are available in the website of the Spanish Statistical Office (INE), `http://www.ine.es`. These four data sets are briefly described below:

- Firstly, the data set `ActivePeople` was considered as an example in order to illustrate the simple index numbers. It contains information separately on the number (thousands) of economically active women and men in Spain from the first trimester in 2002 to the fourth one in 2019.
- Secondly, `ECResources` is a data set containing as variables, the unitary value (euros) and consumption (thousands of units) of several combustibles and other energy resources for the

main home in Spain from 2006 to 2015. It was used in this paper for illustrating complex index numbers.

- An additional data set called `Mortgages` was also included in the package. In this case, the variables correspond to the number of mortgages constituted on urban properties in Spain from 2003 to 2018, distinguished between the kind of mortgages entities (banks, saving banks and other types). The corresponding mortgages amounts (thousands of euros) were also included as variables.
- Finally, the variables in the data set `Food` are the unitary value (euros) and consumed amount (thousands of units) of the main types of food in Spain from 2006 to 2015.

Once the package is installed and loaded, a full description of these data sets is shown through `help(ActivePeople)`, `help(ECResources)`, `help(Mortgages)`, and `help(Food)`, respectively.

Functions in IndexNumber

IndexNumber package includes several functions that enable users to determine the index numbers, simple and complex (weighted and non-weighted), described in previous sections. The functions incorporated in the package are summarized in Table 12.

Function	Description
`aggregated.index.number`	Function to obtain several non-weighted index numbers: the Sauerbeck index number (3), the Geometric index number (4), the Harmonic index number (5) the Bradstreet-Dûtot index number (6), the Carli index number (7), the Jevons index number (8) and the Dûtot index number in (9).
`edgeworth.index.number`	Function to calculate the Marshall-Edgeworth index number (12).
`fisher.index.number`	Function to calculate the Fisher index number (13).
`index.number.chain`	Function to calculate the simple index number in chain (2).
`index.number.serie`	Function to calculate the simple index number in series (1).
`laspeyres.index.number`	Function to determine the Laspeyres index number (10).
`paasche.index.number`	Function to obtain the Paasche index number (11).

Table 12: Summary of functions in the **IndexNumber** package.

Users can obtain different kinds of index numbers by introducing the associated parameters in the corresponding function. Table 13 describes the different options to determine those index numbers whose implementation was included in **IndexNumber** package. However, not all of the mentioned options are required since only some of them are specific for each particular class of index number. Thus, Table 14 summarizes the arguments associated with each function. Examples of usage for the implemented functions are described in the next section.

Argument	Description
`x`	A matrix that contains the magnitude(s) under study. In each column, it contains the magnitude of a different product considered. Thus, we have `nrow(x)` values of a magnitude for `ncol(x)` products. Notice that if we intend to analyze a single magnitude, `x` corresponds to a vector of length equal to the total instants of time registered.
`y`	A matrix that contains that magnitude used as weight. In each column, it contains another magnitude associated to each different product along the time. Thus, we have `nrow(x)` values of magnitudes for the set of `ncol(x)` products. It is only required for obtaining those weighted index numbers mentioned in the paper.
`base`	A chain of characters that indicates the nature of the index number. If we introduce `base="serie"`, we compare each value with respect to the initial one. In this case, it is said to be an index number in series. Otherwise, if we introduce `base="chain"`, we obtain the index number in chain, by comparing each value with the immediately previous value.

type	A chain of characters to indicate the type of non-weighted index number to evaluate the evolution of a set of magnitudes (even for different products). By considering base="serie", if we introduce type="arithmetic", we obtain the Sauerbeck index number in (3). If we introduce type="geometric", we obtain the Geometric index in (4). If we choose type="harmonic", we obtain the Harmonic mean index in (5). If we write type="BDutot", we will obtain the Bradstreet-Dûtot index in (6). This argument is only required in the function aggregated.index.number. Otherwise, if we take base="chain" and type="Carli", we obtain the Carli index number in (7). If we introduce type="Jevons", we obtain the Jevons index in (8) and if we choose type="Dutot", we obtain the Dûtot index in (9). This argument is only required in the function aggregated.index.number.
name	A chain of characters to indicate the name of the variable under study.
opt.plot	A Boolean variable that indicates if a graphical description of the index number along the different stages is required. If it is desired, opt.plot=TRUE, else opt.plot=FALSE.
opt.summary	A Boolean variable that indicates if a basic statistical summary of the index number is required. If it is desired, opt.summary=TRUE, else opt.summary=FALSE.

Table 13: Summary of arguments for functions in the **IndexNumber** package.

Function	x	y	base	type	name	opt.plot	opt.summary
aggregated.index.number	✓	-	✓	✓	✓	✓	✓
edgeworth.index.number	✓	✓	-	-	✓	✓	✓
fisher.index.number	✓	✓	-	-	✓	✓	✓
index.number.chain	✓	-	-	-	✓	✓	✓
index.number.serie	✓	-	-	-	✓	✓	✓
laspeyres.index.number	✓	✓	-	-	✓	✓	✓
paasche.index.number	✓	✓	-	-	✓	✓	✓

Table 14: Arguments for each function in the **IndexNumber** package.

Examples of using IndexNumber

In what follows, we describe several examples of the application of the **IndexNumber** package that is used to illustrate its performance. Initially, a user has to incorporate the package from the CRAN in the R Console. After its installation, the next code allows its usage:

```
> library("IndexNumber")
```

Below, it is shown how to use the different functions implemented for determining index numbers on the real data sets presented in the preliminaries section.

Simple index numbers in series in R

The example that we consider describes the obtaining of the simple index numbers in series (1) in R software on the real data partially given in Table 1. Remember that it depicts the number (thousands) of economically active women and men in Spain. As we mentioned before, this information is also included in the data set ActivePeople in **IndexNumber** package. The first trimester of 2002 is considered as the reference value.

Using index.numer.serie() function, we obtain the simple index number in series for the first instants of time in the example.

```
> index.number.serie(ActivePeople$TotalWomen[1:15],name="Woman",opt.plot=TRUE,
                     opt.summary = TRUE)

Index number in serie

Summary

Min.=101.855509425343

Stage=1
```

```
Max.=117.978690528975

Stage=13

$Summary
 Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
101.9   106.1   110.4   110.3   114.6   118.0

$`Index number`
   Stages  Woman Index number
1       0 7442.7     100.0000
2       1 7580.8     101.8555
3       2 7670.2     103.0567
4       3 7751.5     104.1490
5       4 7868.7     105.7237
6       5 7977.8     107.1896
7       6 8093.3     108.7415
8       7 8190.9     110.0528
9       8 8249.7     110.8428
10      9 8348.9     112.1757
11     10 8430.8     113.2761
12     11 8564.6     115.0738
13     12 8635.2     116.0224
14     13 8780.8     117.9787
15     14 8769.1     117.8215
```

We include a graphical summary of the evolution of this magnitude under the fixed criteria in Figure 3, by using `opt.plot=TRUE`. We also summarize the most relevant information in terms of the instant in which the maximum and minimum values are reached choosing `opt.summary=TRUE`.

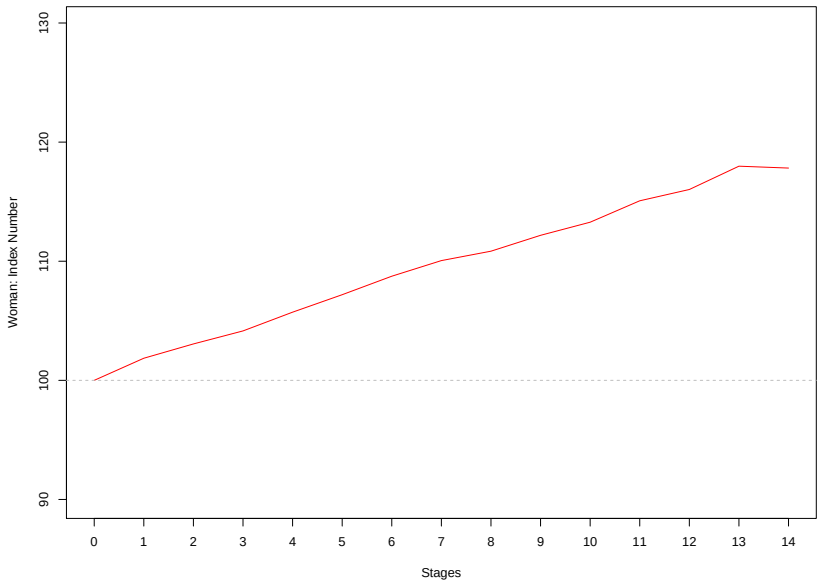


Figure 3: Evolution of the simple index number in series for the number (thousands) of economically active women in Spain.

Analogously, the user can obtain the corresponding results for the data associated to `TotalMen`. The required code in this case is the shown below:

```
index.number.serie(ActivePeople$TotalMen[1:15],name="Man",opt.plot=TRUE,opt.summary = TRUE)
```

Results contained in Table 2 have been obtained using both functions of **IndexNumber** package in R.

Simple index numbers in chain in R

Again, we take the data set `ActivePeople` to determine the corresponding simple index number in chain (2) for the number (thousands) of economically active women in Spain. Note that the reference value in each instant of time is the immediately previous one in the series.

Alternatively, we use `index.numer.chain()` function for obtaining the simple index number in chain for the first instants of time for the variable considered in the example.

```
> index.number.chain(ActivePeople$TotalWomen[1:15],name="Woman",opt.plot=TRUE,
                     opt.summary = TRUE)

Index number in chain

Summary

Min.=99.8667547376093

Stage=14

Max.=101.855509425343

Stage=2

$Summary
 Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
99.87  101.00  101.20  101.18  101.50  101.86

$`Index number`
   Stages  Woman Index number
1       0 7442.7    100.00000
2       1 7580.8    101.85551
3       2 7670.2    101.17930
4       3 7751.5    101.05995
5       4 7868.7    101.51197
6       5 7977.8    101.38651
7       6 8093.3    101.44777
8       7 8190.9    101.20594
9       8 8249.7    100.71787
10      9 8348.9    101.20247
11     10 8430.8    100.98097
12     11 8564.6    101.58704
13     12 8635.2    100.82432
14     13 8780.8    101.68612
15     14 8769.1     99.86675
```

Also in this case, the option `opt.summary=TRUE` summarizes the most relevant information about the corresponding simple index number. The option `opt.plot=TRUE` provides a graphical representation of the evolution of the magnitude as Figure 4 depicts.

Table 3 also includes the numerical analysis of the evolution of economically active men in Spain. To determine these values, we use the following code:

```
index.number.chain(ActivePeople$TotalMen[1:15],name="Man",opt.plot=TRUE,opt.summary = TRUE)
```

Non-weighted complex index numbers in R

In this section, we illustrate the usage of **IndexNumber** to determine non-weighted complex index numbers.

As in its theoretical description, we take the example described in Table 6 to show its performance in practice. That table describes the unitary value of prices of several energy resources for the period 2005-2015. As we have mentioned, this data set is available in library **IndexNumber** under the name `ECResources` (in particular, from the second to the sixth column).

```
> ECResources[,2:6]
   ElectricityPrice NaturalGasPrice LiquifiedGasPrice LiquifiedCombustiblesPrice SolidCombustiblesPrice
```

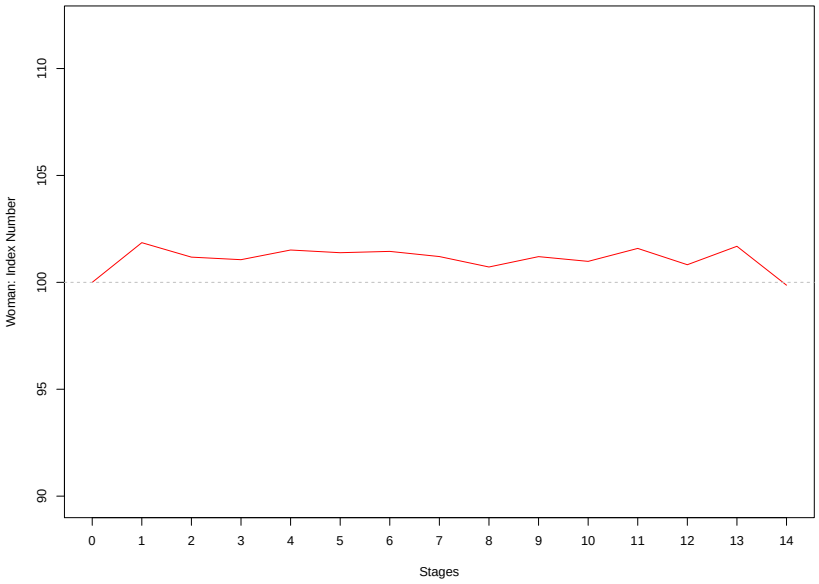


Figure 4: Evolution of the simple index number in chain for the number (thousands) of economically active women in Spain.

```
1          0.14          0.70          1.00          0.69          0.12
2          0.14          0.78          0.97          0.66          0.10
3          0.15          0.83          1.03          0.81          0.11
4          0.16          0.87          0.93          0.68          0.10
5          0.17          0.79          1.04          0.72          0.12
6          0.19          0.77          1.15          0.91          0.12
7          0.22          0.82          1.19          0.96          0.12
8          0.23          1.00          1.34          0.99          0.13
9          0.24          1.07          1.35          0.89          0.15
10         0.25          1.04          1.22          0.77          0.13
```

First, we describe the R procedures that provide the variations of a magnitude in time t with respect to the base period by using non-weighted index numbers. For this purpose, we have to introduce the option `base="serie"`. Notice that all the values included in Table 8 were obtained by using the code in R that we show in the current section.

Sauerbeck index in R

Next, we determine the Sauerbeck index (3) in R software. To this aim, we use the corresponding function `aggregated.index.number()` by adding, as option, `type="arithmetic"`. Recall that it corresponds to an average by stages. In this case, we also include a graphical description of the joint evolution of prices in Figure 5 with `opt.plot=TRUE`, and a numerical summary of such magnitude (with `opt.summary=TRUE`).

```
> aggregated.index.number(ECResources[,2:6],base="serie",type="arithmetic",
                          name="Prices",opt.plot=TRUE,opt.summary=TRUE)

Aggregate index number

Arithmetic

Summary

Min.=97.4828157349897

Stage=1

Max.=142.654244306418
```

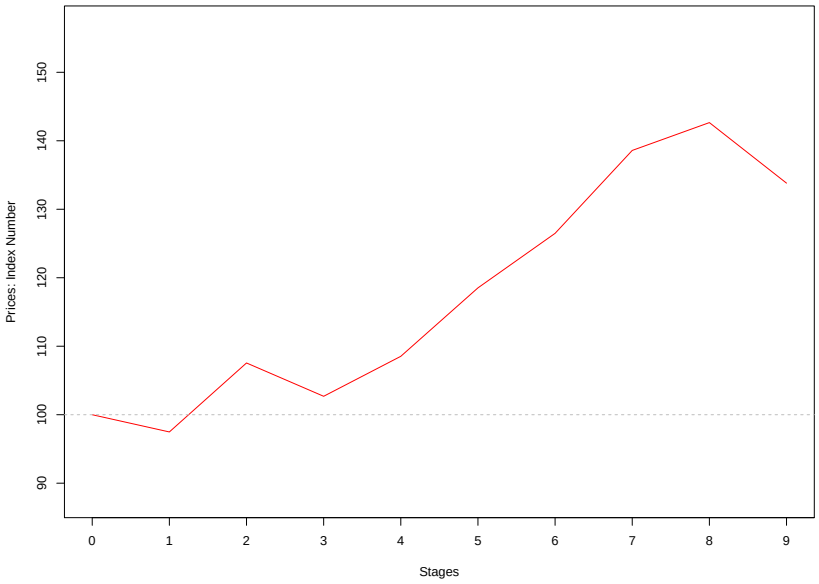


Figure 5: Sauerbeck index evolution for the prices of several energy resources for the period 2005-2015.

```
Stage=8

$Summary
 Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
97.48  107.55  118.52  119.59  133.81  142.65

$`Agg. index number`
   Stages Prices 1 Prices 2 Prices 3 Prices 4 Prices 5 Agg. index number
1       0     0.14     0.70     1.00     0.69     0.12         100.00000
2       1     0.14     0.78     0.97     0.66     0.10          97.48282
3       2     0.15     0.83     1.03     0.81     0.11         107.55445
4       3     0.16     0.87     0.93     0.68     0.10         102.69110
5       4     0.17     0.79     1.04     0.72     0.12         108.52671
6       5     0.19     0.77     1.15     0.91     0.12         118.51967
7       6     0.22     0.82     1.19     0.96     0.12         126.48323
8       7     0.23     1.00     1.34     0.99     0.13         138.59089
9       8     0.24     1.07     1.35     0.89     0.15         142.65424
10      9     0.25     1.04     1.22     0.77     0.13         133.81408
```

Geometric mean index in R

The second approach that we consider is the one given by the Geometric mean index (4). To obtain for the case of prices of the energetic resources, we slightly change the parameters of the function `aggregated.index.number()`. We have to introduce the parameter `type="geometric"` on it.

```
aggregated.index.number(ECResources[,2:6],base="serie",type="geometric",
                        name="Prices",opt.plot=FALSE,opt.summary=FALSE)
```

The results have the same structure as the previous case that we have explained. For this reason, they have not already been included here.

Harmonic mean index in R

The third option of a non-weighted complex index is the Harmonic mean index (5). With regard to the previous cases, the main difference is the parameter `type` that, in this case, has to take the value `"harmonic"`.

```
aggregated.index.number(ECResources[,2:6],base="serie",type="harmonic",
                        name="Prices",opt.plot=FALSE,opt.summary=FALSE)
```

The scheme of showing the results maintains also in this scenario.

Bradstreet-Dûtot index in R

The Bradstreet-Dûtot index (6) is determined in R software by using `aggregated.index.number()` with the parameter `type="BDutot"`. We illustrate the case of obtaining the indicated index for the prices of energetic resources. Again, the output has the same structure as the previous cases.

```
aggregated.index.number(ECResources[,2:6],base="serie",type="BDutot",
                        name="Prices",opt.plot=FALSE,opt.summary=FALSE)
```

Secondly, we describe examples of usage of **IndexNumber** package that involves non-weighted and weighted index numbers in chain. Specifically, we show those ones required for obtaining the results in Table 9. They involves the usage of `base="chain"`.

Carli index in R

The Carli index (7) is obtained in R software through `aggregated.index.number()` with `type="Carli"`. The following R code determines the indicated index for the prices of energetic resources with an analogous structure for the output.

```
aggregated.index.number(ECResources[,2:6],base="chain",type="Carli",
                        name="Prices",opt.plot=FALSE,opt.summary=FALSE)
```

Jevons index in R

The Jevons index (8) can be determined in R software by using `aggregated.index.number()` with the parameter `type="Jevons"`. Again, we illustrate the case of obtaining the indicated index for the prices of energetic resources. The required code is the one displayed below:

```
aggregated.index.number(ECResources[,2:6],base="chain",type="Jevons",
                        name="Prices",opt.plot=FALSE,opt.summary=FALSE)
```

Dûtot index in R

The Dûtot index (9) is determined in R with `aggregated.index.number()` with additionally including `type="BDutot"`. Finally, we illustrate the case of obtaining this index for the prices of energetic resources.

```
aggregated.index.number(ECResources[,2:6],base="chain",type="Dutot",
                        name="Prices",opt.plot=FALSE,opt.summary=FALSE)
```

Weighted complex index numbers in R

Next, we enumerate the capabilities of **IndexNumber** package in R software to determine weighted complex index numbers. The results in Table 11 were also obtained by using the functions of R that we show below.

Again, we pretend to obtain the evolution of the unitary value of prices of several energy resources for the period 2005-2015. However, in this case, we weight their values by the total amount of consumed energy resources given in Table 7. This information is also included in the data set `ECResources` of the package **IndexNumber**.

```
> ECResources[,7:11]
   ElectricityConsumed NaturalGasConsumed LiquifiedGasConsumed LiquifiedCombustiblesConsumed SolidCombustiblesConsumed
1             50623635            3617285              1057488                       2297923                    1306920
2             51990501            3266575              1066857                       2454265                    1602799
3             54990338            3473851              1210607                       2274326                    1556673
4             59749470            3730349              1113642                       2505711                    1724222
5             69751162            3954065               987112                       2345215                    1584123
6             67574654            4466072               926824                       1974662                    1414234
7             62878557            4576052               943632                       2029733                    1733591
8             56017871            4116079               867695                       1952593                    2071152
9             49177739            3653055               868743                       2180866                    2077766
10            48541712            3795339               818183                       2176533                    2161208
```

Laspeyres index in R

Using weights, the first alternative that we describe is devoted to obtain the Laspeyres index (10) in R. **IndexNumber** package allows this through `laspeyres.index.number()` by adding these as parameters: the matrix of the magnitudes to be evaluated, the matrix containing the weights, and several options of graphical and numerical representation for the results.

```
> laspeyres.index.number(ECResources[,2:6],ECResources[,7:11],
                         name="Price",opt.plot=TRUE,opt.summary=TRUE)

Laspeyres index number

Summary

Min.=101.309108829536

Stage=1

Max.=158.535309198466

Stage=8

$Summary
 Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
101.3   112.1   127.8   131.2   154.0   158.5

$`Agg. index number`
Stages Price 1 Price 2 Price 3 Price 4 Price 5 Agg. index number
1       0    0.14    0.70    1.00    0.69    0.12          100.0000
2       1    0.14    0.78    0.97    0.66    0.10          101.3091
3       2    0.15    0.83    1.03    0.81    0.11          110.2332
4       3    0.16    0.87    0.93    0.68    0.10          112.1124
5       4    0.17    0.79    1.04    0.72    0.12          115.7457
6       5    0.19    0.77    1.15    0.91    0.12          127.7677
7       6    0.22    0.82    1.19    0.96    0.12          142.7184
8       7    0.23    1.00    1.34    0.99    0.13          153.9749
9       8    0.24    1.07    1.35    0.89    0.15          158.5353
10      9    0.25    1.04    1.22    0.77    0.13          158.2000
```

As before, if the option `opt.plot=TRUE` is considered, the output of the function also includes a graphical representation in which the joint evolution can be analyzed as Figure 6 depicts.

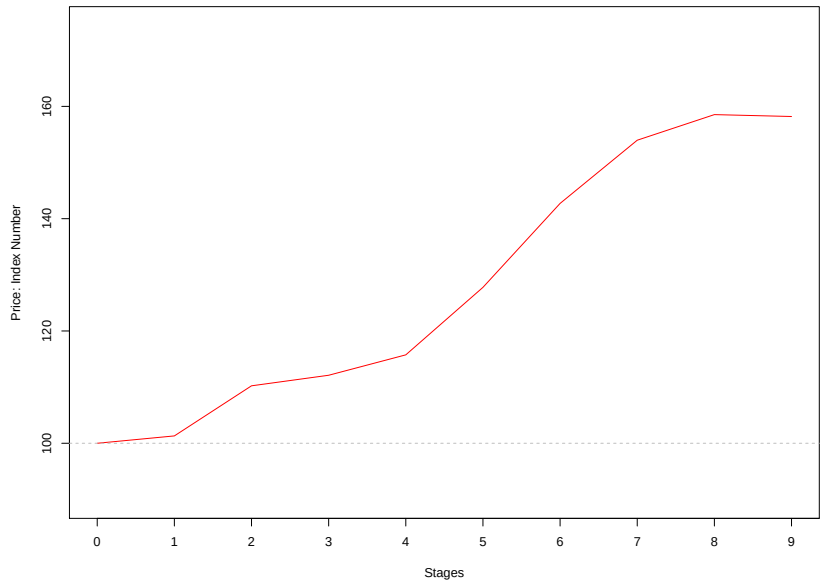


Figure 6: Laspeyres index evolution for the prices of several energy resources for the period 2005-2015.

Paasche index in R

Analogously, the joint evolution of magnitudes under the Paasche index (11) can be also determined by using **IndexNumber** package in R software. More specifically, `paasche.index.number()` provides

the mentioned weighted index number. The results of the function follows a similar structure as the function previously mentioned, and using the same graphical and numerical options.

```
paasche.index.number(ECResources[,2:6],ECResources[,7:11],
                     name="Price",opt.plot=TRUE,opt.summary=TRUE)
```

Marshall-Edgeworth index in R

The obtaining of the Marshall-Edgeworth index number (12) is also possible in R software through the use of **IndexNumber** package. In particular, we have to use the function `edgeworth.index.number()` with the above-mentioned options for obtaining graphics and summaries.

```
edgeworth.index.number(ECResources[,2:6],ECResources[,7:11],
                       name="Price",opt.plot=FALSE,opt.summary=FALSE)
```

Fisher index in R

Here, we consider the case of determining the Fisher index (13). In this case, `fisher.index.number()` provides a measure of the considered magnitude. Again, the options `opt.plot` and `opt.summary` allow the obtaining of additional information that may be of interest to the user.

```
fisher.index.number(ECResources[,2:6],ECResources[,7:11],
                    name="Price",opt.plot=FALSE,opt.summary=FALSE)
```

A complex index for v in R

The complex index for v given in (14) can be easily obtained as the Bradstreet-Dûtot index for the value in each instant of time. Recall that the value is obtained as the product of the amount of good by its price in each instant of time. Thus, the function `aggregated.index.number()` provides the value of this new index number as follows.

```
aggregated.index.number(ECResources[,2:6]*ECResources[,7:11],
                        base="serie",type="BDutot",name="Prices",opt.plot=FALSE,
                        opt.summary=FALSE)
```

Concluding Remarks

This paper discusses the implementation in R of classical index numbers used for comparing magnitudes mainly in economic contexts. Therefore, the **IndexNumber** package provides R users a set of functions to calculate index numbers. Concretely, this library allows the calculation of simple index numbers in series and in chain. Furthermore, complex index numbers are also implemented. In particular, the non-weigthed index numbers included are the Sauerbeck, Geometric mean, Harmonic mean, Bradstreet-Dûtot, Carli, Jevons, and Dûtot indexes; as weighted index numbers, the Laspeyres, Paasche, Marshall-Edgeworth, Fisher, and Bradstreet-Dûtot indexes were considered. Additionally, this package contains graphical tools in order to facilitate the results visualization and four real data sets that can be used as illustrative examples. Moreover, the use of this library could be easily combined with other classical packages focused on time series analysis. Future research and development plans for forthcoming versions of the package include the addition of new index numbers already considered in the literature that can be dealt with in the framework presented above. Of course, the corresponding graphical tools should also be implemented.

Acknowledgments

A. Saavedra-Nieves acknowledges the financial support of FEDER/Ministerio de Ciencia, Innovación y Universidades - Agencia Estatal de Investigación under grant MTM2017-87197-C3-3-P. P. Saavedra-Nieves acknowledges the financial support of Ministerio de Economía y Competitividad of the Spanish government under grants MTM2016-76969-P and MTM2017-089422-P, and by the Xunta de Galicia through the European Regional Development Fund (Grupos de Referencia Competitiva ED431C-2017/38).

Bibliography

R. G. D. Allen. *Index numbers in theory and practice*. Springer, 1982. URL https://doi.org/10.4324/9780203788677. [p253]

B. W. Ang, F. Liu, and H.-S. Chung. A generalized Fisher index approach to energy decomposition analysis. *Energy Economics*, 26(5):757–763, 2004. URL https://doi.org/10.1016/j.eneco.2004.02.002. [p254]

W. A. Barnett. Economic monetary aggregates an application of index number and aggregation theory. *Journal of econometrics*, 14(1):11–48, 1980. URL https://doi.org/10.1016/0304-4076(80)90070-6. [p254]

J. Białek. Proposition of a general formula for price indices. *Communications in Statistics-Theory and Methods*, 41(5):943–952, 2012. URL https://doi.org/10.1080/03610926.2010.533238. [p254]

J. Białek. Simulation study of an original price index formula. *Communications in Statistics-Simulation and Computation*, 43(2):285–297, 2014. URL https://doi.org/10.1080/03610918.2012.700367. [p254]

J. Bialek. *PriceIndices: Calculating Bilateral and Multilateral Price Indexes*, 2021. URL https://CRAN.R-project.org/package=PriceIndices. R package version 0.0.4. [p263]

G. A. Boyd and J. M. Roop. A note on the Fisher ideal index decomposition for structural change in energy intensity. *The Energy Journal*, 25(1), 2004. URL https://doi.org/10.5547/ISSN0195-6574-EJ-Vol25-No1-5. [p254]

G. Boyle. The economic theory of index numbers: Empirical tests for volume indices of agricultural output. *Irish Journal of Agricultural Economics and Rural Sociology*, pages 1–20, 1988. URL https://www.jstor.org/stable/25556589. [p254]

G.-R. Carli. Del valore e della proporzione dei metalli monetati. *Scrittori Classici Italiani Di Economia Politica*, 13:297–366, 1804. [p260]

F. Coggeshall. The arithmetic, geometric, and harmonic means. *The Quarterly Journal of Economics*, 1(1): 83–86, 1886. URL https://doi.org/10.2307/1883111. [p260]

B. T. Coyle. Aggregation of price risk over commodities: An economic index number approach. *American Journal of Agricultural Economics*, 89(4):1085–1097, 2007. URL https://doi.org/10.1111/j.1467-8276.2007.01023.x. [p254]

CPI Manual. *Consumer Price Index Manual: Theory and Practice*. OECD, UN, Eurostat, and The World Bank by ILO, Geneva, 2004. [p253]

P. M. Danzon and L.-W. Chao. Cross-national price differences for pharmaceuticals: how large, and why? *Journal of health economics*, 19(2):159–195, 2000. URL https://doi.org/10.1016/S0167-6296(99)00039-9. [p254]

J. de Haan and E. Opperdoes. Estimation of the Coffee Price index Using scanner data: the choice of the Micro index. In *third meeting of the Ottawa Group*, pages 16–18, 1997. [p254]

W. E. Diewert and A. O. Nakamura. The measurement of productivity for nations. *Handbook of econometrics*, 6:4501–4586, 2007. URL https://doi.org/10.1016/S1573-4412(07)06066-7. [p254]

Y. Dodge. *The concise encyclopedia of statistics*. Springer Science & Business Media, 2008. URL https://doi.org/10.1007/978-0-387-32833-1. [p253]

M. Drobisch. Ueber Mittelgrössen und die Anwendbarkeit derselben auf die Berechnung des Steigens und Sinkens des Geldwerths. *Berichte der mathematisch-physicalischen Classe der königlich Sächsischen Gesellschaft der Wissenschaften*, 1, 1871. [p253]

J. C. Dumagan. Comparing the superlative Törnqvist and Fisher ideal indexes. *Economics Letters*, 76(2): 251–258, 2002. URL https://doi.org/10.1016/S0165-1765(02)00049-6. [p254]

C. Dûtot. *Reflexions politiques sur les finances, et le commerce*, volume 2. les freres Vaillant [and] N. Prevost, 1754. [p253, 261]

F. Y. Edgeworth. Measurement of change in value of money i. *First Memorandum presented to the British Association for the Advancement of Science. Reprinted in his Papers Relating to Political Economy*, 1: 198–259, 1887. [p262]

S. G. Evelyn. An account of some endeavours to ascertain a standard of weight and measure. Philos. *Trans*, 113, 1798. [p253]

I. Fisher. *The making of index numbers: a study of their varieties, tests, and reliability*, volume 1. Houghton Mifflin, 1922. URL https://doi.org/10.2307/1883934. [p253, 262]

W. Fleetwood. *Chronicon Preciosum: Or, an Account of English Money: The Price of Corn, and Other Commodities, for the Last 600 Years. In a Letter to a Student in the University of Oxford*. Charles Harper, 1707. [p253]

F. Forsyth and R. F. Fowler. The theory and practice of chain price index numbers. *Journal of the Royal Statistical Society: Series A (General)*, 144(2):224–246, 1981. URL https://doi.org/10.2307/2981921. [p254, 256]

R. Frisch. Annual survey of general economic theory: The problem of index numbers. *Econometrica: Journal of the Econometric Society*, pages 1–38, 1936. URL https://doi.org/10.2307/1907119. [p253]

A. Henningsen. *micEconIndex: Price and Quantity Indices*, 2017. URL https://CRAN.R-project.org/package=micEconIndex. R package version 0.1-6. [p263]

R. J. Hill. Comparing price levels across countries using minimum-spanning trees. *Review of Economics and Statistics*, 81(1):135–142, 1999. URL https://doi.org/10.1162/003465399767923881. [p254]

R. J. Hill. When does chaining reduce the Paasche–Laspeyres spread? An application to scanner data. *Review of Income and Wealth*, 52(2):309–325, 2006. [p254]

R. J. Hill and D. Melser. Hedonic imputation and the price index problem: an application to housing. *Economic Inquiry*, 46(4):593–609, 2008. URL http://doi.org/10.1111/j.1465-7295.2007.00110.x. [p254]

L. Ivancic, W. E. Diewert, and K. J. Fox. Scanner data, time aggregation and the construction of price indexes. *Journal of Econometrics*, 161(1):24–35, 2011. URL https://doi.org/10.1016/j.jeconom.2010.09.003. [p254]

W. S. Jevons. *A Serious Fall in the Value of Gold Ascertained: And Its Social Effects Set Forth*. E. Stanford, 1863. [p253, 260, 261]

W. S. Jevons. On the variation of prices and the value of the currency since 1782. *Journal of the Statistical Society of London*, 28(2):294–320, 1865. URL https://doi.org/10.2307/2338419. [p260]

N. Kakwani and R. J. Hill. Economic theory of spatial cost of living indices with application to Thailand. *Journal of Public Economics*, 86(1):71–97, 2002. URL https://doi.org/10.1016/S0047-2727(00)00174-2. [p254]

M. Kendall. Studies in the history of probability and statistics, XXI. The early history of index numbers. *Revue de l'Institut International de Statistique*, pages 1–12, 1969. URL https://doi.org/10.2307/1402090. [p253]

M. Kendall. *The early history of index numbers*, volume II. Studies in the History of Statistics and Probability, Griffin, London, 1977. URL https://doi.org/10.2307/1402090. [p253]

R. Lamm. Index numbers and changes in food prices. *Agricultural Economics Research*, 32(2):41–43, 1980. [p254]

E. Laspeyres. Hamburger Waarenpreise 1851–1863 und die californisch-australischen Goldentdeckungen seit 1848. *Jahrbücher für Nationalökonomie und Statistik*, 3(1):81–118, 1864. [p253]

K. Laspeyres. Ix. Die Berechnung einer mittleren Waarenpreissteigerung. *Jahrbücher für Nationalökonomie und Statistik*, 16(1):296–318, 1871. [p253, 261]

J. Lowe. The Present State of England in Regard to Agriculture. *Trade & Finance*, 1822. [p253]

A. Marshall. Remedies for fluctuations of general prices. *The Contemporary review, 1866-1900*, 51: 355–375, 1887. [p262]

C. J. O'Donnell. Measures of Productivity Change. In *Productivity and Efficiency Analysis*, pages 93–143. Springer, 2018. URL https://doi.org/10.1007/s11123-012-0311-1. [p254]

H. Paasche. Ueber die Preisentwicklung der letzten Jahre nach den Hamburger Börsennotirungen Jahrbücher für Nationalökonomie und Statistik, 23, 1874. [p253, 262]

R. Palgrave. Currency and Standard of Value in England, France and India and the Rates of Exchange Between these Countries. *Memorandum submitted to the Royal Commission on Depression of Trade and Industry*, pages 312–90, 1886. [p253]

M. B. Reinsdorf, W. E. Diewert, and C. Ehemann. Additive decompositions for Fisher, Törnqvist and geometric mean indexes. *Journal of Economic and Social Measurement*, 28(1-2):51–61, 2002. URL http://dx.doi.org/10.3233/JEM-2003-0194. [p254]

A. Saavedra-Nieves and P. Saavedra-Nieves. *IndexNumber: Index Numbers in Social Sciences*, 2021. R package version 1.3.2. [p254]

A. Sauerbeck. Index Numbers of Prices. *The Economic Journal*, 5(18):161–174, 1895. URL https://doi.org/10.2307/2955755. [p259]

B. Su and B. Ang. Attribution of changes in the generalized Fisher index with application to embodied emission studies. *Energy*, 69:778–786, 2014. URL https://doi.org/10.1016/j.energy.2014.03.074. [p254]

C. M. Walsh. *The measurement of general exchange-value*, volume 25. Macmillan, 1901. [p260]

G. White. *IndexNumR: Index Number Calculation*, 2021. URL https://CRAN.R-project.org/package=IndexNumR. R package version 0.2.0. [p263]

C. Zhen, E. A. Finkelstein, S. A. Karns, E. S. Leibtag, and C. Zhang. Scanner data-based panel price indexes. *American journal of agricultural economics*, 101(1):311–329, 2019. URL http://doi.org/10.1093/ajae/aay032. [p254]

J. L. Zofío and A. M. Prieto. Return to dollar, generalized distance function and the Fisher productivity index. *Spanish Economic Review*, 8(2):113–138, 2006. URL http://dx.doi.org/10.1007/s10108-006-9004-0. [p254]

Alejandro Saavedra-Nieves
Departamento de Estatística, Análise Matemática e Optimización
Universidade de Santiago de Compostela
Spain
ORCID: 0000-0003-1251-6525
alejandro.saavedra.nieves@usc.es

Paula Saavedra-Nieves
Departamento de Estatística, Análise Matemática e Optimización
Universidade de Santiago de Compostela
Spain
ORCID: 0000-0002-3224-8858
paula.saavedra@usc.es

garchx: Flexible and Robust GARCH-X Modeling

by Genaro Sucarrat

Abstract The **garchx** package provides a user-friendly, fast, flexible, and robust framework for the estimation and inference of GARCH(p, q, r)-X models, where p is the ARCH order, q is the GARCH order, r is the asymmetry or leverage order, and 'X' indicates that covariates can be included. Quasi Maximum Likelihood (QML) methods ensure estimates are consistent and standard errors valid, even when the standardized innovations are non-normal or dependent, or both. Zero-coefficient restrictions by omission enable parsimonious specifications, and functions to facilitate the non-standard inference associated with zero-restrictions in the null-hypothesis are provided. Finally, in the formal comparisons of precision and speed, the **garchx** package performs well relative to other prominent GARCH-packages on CRAN.

Introduction

In the Autoregressive Conditional Heteroscedasticity (ARCH) class of models proposed by Engle (1982), the variable of interest ϵ_t is decomposed multiplicatively as

$$\epsilon_t = \sigma_t \eta_t, \tag{1}$$

where $\sigma_t > 0$ is the standard deviation of ϵ_t, and η_t is a real-valued standardized innovation with mean zero and unit variance (e.g., the standard normal). Originally, Engle (1982) interpreted ϵ_t as the error term of a dynamic regression of inflation so that σ_t is the uncertainty of the inflation forecast. However, ARCH models have also proved to be useful in many other areas. The field in which they have become most popular is finance. There, ϵ_t is commonly interpreted as a financial return, either raw or mean-corrected (i.e., ϵ_t has mean zero) so that σ_t is a measure of the variability or volatility of return. In Engle and Russell (1998), it was noted that the ARCH framework coul also be used to model non-negative variables, say, the trading volume of financial assets, the duration between financial trades, and so on. Specifically, suppose y_t denotes a non-negative variable, say, volume, and μ_t is the conditional expectation of y_t. Engle and Russell (1998) noted that in the expression $\epsilon_t^2 = \sigma_t^2 \eta_t^2$ implied by the ARCH model, if you replace ϵ_t^2 with y_t and σ_t^2 with μ_t, then it follows straightforwardly that μ_t is the conditional expectation of y_t. This is justified theoretically since the underlying estimation theory does not require that ϵ_t has mean zero. The observation made by Engle and Russell (1998) spurred a new class of models, which is known as the Multiplicative Error Model (MEM); see Brownlees et al. (2012) for a survey. The practical implication of all this is that ARCH-software can, in fact, be used to estimate MEMs by simply feeding the package in question with $\sqrt{y}_t$ rather than ϵ_t. The fitted values of σ_t^2 become the fitted values of μ_t, the error term is defined by y_t/μ_t, and the inference theory and other statistical results usually carry over straightforwardly. In conclusion, the ARCH-class of models provides a flexible framework that can be used in a very wide range of empirical applications.

Prominent GARCH-packages on CRAN

Although a large number of specifications of σ_t have been proposed, the most common in empirical applications are variants of the Generalised ARCH (GARCH) proposed by Bollerslev (1986). In particular, the plain GARCH(1,1) is ubiquitous:

$$\sigma_t^2 = \omega + \alpha_1 \epsilon_{t-1}^2 + \beta_1 \sigma_{t-1}^2, \qquad \omega > 0, \quad \alpha_1 \geq 0, \quad \beta_1 \geq 0. \tag{2}$$

By analogy with an ARMA(1,1), the conditional variance σ_t^2 is modeled as a function of the recent past, where the ϵ_{t-1}^2 is referred to as the ARCH(1) term, and σ_{t-1}^2 is referred to as the GARCH(1) term. The non-negativity of ϵ_t^2, together with the constraints on the parameters ω, α_1 and β_1, ensure σ_t^2 is strictly positive. Another way of ensuring that σ_t^2 is strictly positive is by modeling its logarithm, $\ln \sigma_t^2$, as for example in the log-ARCH class of models proposed by Geweke (1986), Pantula (1986), and Milhøj (1987). Here, however, the focus is exclusively on non-logarithmic specifications of σ_t. Also, multivariate GARCH specifications are not covered.

The most prominent packages on CRAN that are commonly used to estimate variants of (2) are **tseries** (Trapletti and Hornik, 2019), **fGarch** (Wuertz et al., 2020), and **rugarch** (Ghalanos, 2020). In

tseries, the function `garch()` enables estimation of the GARCH(p, q) specification

$$\sigma_t^2 = \omega + \sum_{i=1}^{p} \alpha_i \epsilon_{t-i}^2 + \sum_{j=1}^{q} \beta_j \sigma_{t-j}^2, \qquad \omega > 0, \quad \alpha_i \geq 0, \quad \beta_j \geq 0. \tag{3}$$

Notable features of `garch()` include simplicity and speed. With respect to simplicity, it is appealing that a plain GARCH(1,1) can be estimated by the straightforward and simple command `garch(eps)`, where `eps` is the vector or series in question, i.e., ϵ_t in (1). As for speed, it is the fastest among the packages compared here, and outside the R universe, it is also likely to be one of the fastest. Indeed, a formal speed comparison (see Section 4) reveals the relative speed provided by `garch()` can be important, particularly if the number of observations is large or if many models have to be estimated (as in simulations). A notable limitation of (3) is that it does not allow for asymmetry terms, e.g., $I_{\{\epsilon_{t-1}<0\}}\epsilon_{t-1}^2$, also known as 'leverage', or covariates. Asymmetry effects are particularly common in daily stock returns, where its presence implies that volatility in day t is higher if the return on the previous day, ϵ_{t-1}, is negative. Often, such asymmetry effects are attributed to leverage.

In **fGarch**, asymmetry effects are possible. Specifically, the function `garchFit()` enables estimation of the Asymmetric Power GARCH (APARCH) specification

$$\sigma_t^\delta = \omega + \sum_{i=1}^{p} \alpha_i |\epsilon_{t-i}|^\delta + \sum_{j=1}^{q} \beta_j \sigma_{t-j}^\delta + \sum_{k=1}^{r} \gamma_k I_{\{\epsilon_{t-k}<0\}} |\epsilon_{t-k}|^\delta, \qquad \gamma_k \geq 0, \tag{4}$$

where $\delta > 0$ is the power parameter, and the γ_k's are the asymmetry parameters. The power parameter δ is rarely different from 2 in empirical applications, but it does provide the added flexibility of modeling, say, the conditional standard deviation ($\delta = 1$) directly if the user wishes to do so. Another feature of `garchFit()` is that other densities than the normal can be used in the ML estimation, for example, the skewed normal or the skewed Student's t. In theory, this provides more efficient estimates asymptotically if η_t is skewed or more heavy-tailed than the normal. In finite samples, however, the actual efficiency may be more dependent on how estimation is carried out numerically. Also, additional density parameters may increase the possibility of numerical problems. To alleviate this potential problem, the package offers a non-normality robust coefficient-covariance along the lines of Bollerslev and Wooldridge (1992) in combination with normal ML. The coefficient-covariance of Bollerslev and Wooldridge (1992) does not, however, provide robustness to the dependence of the η_t's. Finally, **fGarch** also offers the possibility of specifying the mean equation as an ARMA model. That is, $\epsilon_t = y_t - \mu_t$, where μ_t is the ARMA specification. Theoretically, joint estimation of μ_t and σ_t may improve the asymptotic efficiency compared with, say, a two-step estimation approach, where μ_t is estimated in the first step, and σ_t is estimated in the second using the residuals from the first step. In practice, however, the joint estimation may, in fact, reduce the actual efficiency. The reasons for this are the increase in the number of parameters to be estimated and the increased possibility of numerical problems due to the increase in the number of parameters to be estimated.

A limitation of **fGarch** is that it does not allow for additional covariates ('X') in (4). This can be a serious limitation since additional conditioning variables like $high - low$, realized volatility, interest rates, and so on may help to predict or explain volatility in substantial ways. The **rugarch** package remedies this. Most of the non-exponential specifications offered by **rugarch** are contained in

$$\sigma_t^\delta = \omega + \sum_{i=1}^{p} \alpha_i |\epsilon_{t-i}|^\delta + \sum_{j=1}^{q} \beta_j \sigma_{t-j}^\delta + \sum_{k=1}^{r} \gamma_k I_{\{\epsilon_{t-k}<0\}} |\epsilon_{t-k}|^\delta + \sum_{l=1}^{s} \lambda_l x_{l,t-1}, \quad \lambda_l \geq 0, \quad x_{l,t-1} \geq 0, \tag{5}$$

where the $x_{l,t-1}$'s are the covariates. However, it should be mentioned that the package also enables the estimation of additional models, e.g., the Component GARCH model and the Fractionally Integrated GARCH model, amongst others. These additional models are not the focus here. Note that the covariates in (5) need not enter as lagged of order 1. That is, $x_{l,t-1}$ may denote a variable that is lagged of order 2, say, w_{t-2}, and so on. A variable may also enter as unlagged, w_t. However, it is not clear what the theoretical requirements are for consistent estimation, in this case, due to simultaneity issues. Just as in **fGarch**, the **rugarch** package also enables a non-normality robust coefficient-covariance, ML estimation with non-normal densities, and the joint estimation of an ARMA specification in the mean together with σ_t. To the best of my knowledge, no other CRAN package offers more univariate GARCH specifications than **rugarch**.

What does garchx offer that is not already available?

The **garchx** package[1] aims to provide a simple, fast, flexible, and robust – both theoretically and numerically – framework for GARCH-X modeling. The specifications that can be estimated are all contained within

$$\sigma_t^2 = \omega + \sum_{i=1}^{p} \alpha_i \epsilon_{t-i}^2 + \sum_{j=1}^{q} \beta_j \sigma_{t-j}^2 + \sum_{k=1}^{r} \gamma_k I_{\{\epsilon_{t-k}<0\}} \epsilon_{t-k}^2 + \sum_{l=1}^{s} \lambda_l x_{l,t-1}. \tag{6}$$

While this implies a restriction of $\delta = 2$ compared with the **rugarch** package, **garchx** enables several additional features that are not available in the above-mentioned packages:

i) *Robustness to dependence*. Normal ML estimation is usually consistent when the η_t's are dependent over time, see e.g., Escanciano (2009) and Francq and Thieu (2018). This is useful, for example, when the conditional skewness, conditional kurtosis, or conditional zero-probability of η_t is time-varying and dependent on the past in unknown ways. In these cases, however, the non-normality robust coefficient-covariance of Bollerslev and Wooldridge (1992) is not valid. Optionally, **garchx** offers the possibility of using the dependence (and non-normality) robust coefficient-covariance derived by Francq and Thieu (2018).

ii) *Inference under nullity*. In applied work, it is frequently of interest to test whether a coefficient differs from zero. The permissible parameter-space of GARCH models, however, is bounded from below by zero. Accordingly, non-standard inference is required when the value of a null-hypothesis lies on the zero-boundary; see Francq and Thieu (2018). The **garchx** package offers functions to facilitate such tests, named `ttest0()` and `waldtest0()`, respectively, based on the results by Francq and Thieu (2018).

iii) *Zero-constrained coefficients by omission*. If one or more coefficients are indeed zero, then it may be desirable to obtain estimates under zero-constraints on these coefficients. For example, if ϵ_t is the error term in a regression of quarterly inflation, then it may be desirable to estimate a GARCH(4,4) model in which the parameters associated with orders 1, 2, and 3 are restricted to zero. That is, it is desirable to estimate

$$\sigma_t^2 = \omega + \alpha_4 \epsilon_{t-4}^2 + \beta_4 \sigma_{t-4}^2.$$

Another example is the non-exponential Realized GARCH of Hansen et al. (2012), which is simply a GARCH(0,1)-X. That is, the ARCH(1) coefficient is set to zero. Zero-constrained coefficients do not only provide a more parsimonious characterization of the process in question. They may also make estimation more efficient and stable numerically since fewer parameters need to be estimated. **rugarch** offers a feature in which coefficients can be fixed to zero. However, its approach is not by omission. In other words, using `coef` in **rugarch** to extract the coefficients in the GARCH(4,4) example above will return a vector of length 9 rather than of length 3, while the coefficient-covariance returned by **rugarch** will be 3×3. This makes multiple hypothesis testing with Wald tests tedious in constrained models. In **garchx**, by contrasts, Wald tests in constrained models are straightforward since the zeros are due to omission. The vector returned by `coef` is of length 3 and the coefficient-covariance is 3×3.

iv) *Computation of the asymptotic coefficient-covariance*. Knowing the value of the theoretical, asymptotic coefficient-covariance matrix is needed for a formal evaluation of an estimator. For GARCH models, these expressions are not available in explicit form. The **garchx** offers a function, `garchxAvar()`, that computes them by combining simulation and numerical differentiation. To illustrate the usage of `garchxAvar()`, a small Monte Carlo study is undertaken. While the results of the study suggest all four packages return approximately unbiased estimates in large samples, they also suggest **tseries** and **rugarch** are less robust numerically than **fGarch** and **garchx** under default options. In addition, the simulations reveal the standard errors of **tseries** can be substantially biased downwards when η_t is non-normal. A bias is expected since **tseries** does not offer a non-normality robust coefficient-covariance. However, the bias is larger than suggested by the underlying estimation theory.

Table 1 provides a summary of the features offered by the four packages.

The rest of the article is organised as follows.[2] The next section provides an overview of the **garchx** package and its usage. Thereafter, the `garchxAvar()` function is illustrated by means of a Monte Carlo study of the large sample properties of the packages. Next, a speed comparison of the packages is undertaken. While **tseries** is the fastest for the specifications it can estimate, **garchx** is notably faster than **fGarch** and **rugarch** in all the experiments that are conducted. Finally, the last section concludes.

[1] On CRAN since 9 April 2020.

[2] All commands and code-executions were carried out in R version 3.6.3 on a Windows 10 64bit machine.

	tseries	fGarch	rugarch	garchx
GARCH(p, q)	Yes	Yes	Yes	Yes
Asymmetry		Yes	Yes	Yes
Power GARCH		Yes	Yes	
Covariates (X)			Yes	Yes
Additional GARCH models			Yes	
Non-normality robust vcov		Yes	Yes	Yes
Dependence robust vcov				Yes
Computation of asymptotic vcov				Yes
Constrained estimation			Yes	
Zero-constraints by omission				Yes
Inference under null-restrictions				Yes
Normal (Q)ML	Yes	Yes	Yes	Yes
Non-normal ML		Yes	Yes	
ARMA in the mean		Yes	Yes	

Table 1: A feature-based comparison of selected R packages that offer GARCH-estimation: **tseries** version 0.10-47 (Trapletti and Hornik, 2019), **fGarch** version 3042.83.2 (Wuertz et al., 2020), **rugarch** version 1.4-2 (Ghalanos, 2020), and **garchx** version 1.1 (Sucarrat, 2020).

The garchx package

Estimation theory

Let $\mathcal{F}_{t-1}$ denote the sigma-field generated by past observables. Formally, in the **garchx** package, ϵ_t is expected to satisfy $\epsilon_t^2 = \sigma_t^2\eta_t^2$, (6) and

$$E(\eta_t^2|\mathcal{F}_{t-1}) = 1 \qquad \text{for all } t. \tag{7}$$

The conditional unit variance assumption in (7) is very mild since it does not require that the distribution of η_t is identical over time, nor that η_t is independent of the past. In particular, the assumption is compatible with a time-varying conditional skewness $E(\eta_t^3|\mathcal{F}_{t-1})$ that depends on the past in unknown ways, a time-varying conditional kurtosis $E(\eta_t^4|\mathcal{F}_{t-1})$ that depends on the past in unknown ways, and even a time-varying conditional zero-probability $Pr(\eta_t = 0|\mathcal{F}_{t-1})$ that depends on the past in unknown ways. Empirically, such forms of dependence are common; see e.g., Hansen (1994) and Sucarrat and Grønneberg (2020). GARCH models in which the η_t's are dependent are often referred to as semi-strong after Drost and Nijman (1993).

Subject to suitable regularity conditions, the normal ML estimator provides consistent and asymptotically normal estimates of semi-strong GARCH models; see Francq and Thieu (2018). Specifically, they show that

$$\sqrt{T}(\widehat{\boldsymbol{\vartheta}} - \boldsymbol{\vartheta}_0) \xrightarrow{d} N(\mathbf{0}, \boldsymbol{\Sigma}), \qquad \boldsymbol{\Sigma} = \boldsymbol{J}^{-1}\boldsymbol{I}\boldsymbol{J}^{-1}, \quad \boldsymbol{J} = E\left(\frac{\partial^2 l_t(\boldsymbol{\vartheta}_0)}{\partial\boldsymbol{\vartheta}\partial\boldsymbol{\vartheta}'}\right), \quad \boldsymbol{I} = E\left(\frac{\partial l_t(\boldsymbol{\vartheta}_0)}{\partial\boldsymbol{\vartheta}}\frac{\partial l_t(\boldsymbol{\vartheta}_0)}{\partial\boldsymbol{\vartheta}'}\right), \tag{8}$$

where

$$\widehat{\boldsymbol{\vartheta}} = \arg\min_{\boldsymbol{\vartheta}} \frac{1}{T}\sum_{t=1}^{T} l_t(\boldsymbol{\vartheta}), \qquad l_t(\boldsymbol{\vartheta}) = \frac{\epsilon_t^2}{\sigma_t^2(\boldsymbol{\vartheta})} + \ln\sigma_t^2(\boldsymbol{\vartheta}), \tag{9}$$

is the (normal) Quasi ML (QML) estimate of the true parameter $\boldsymbol{\vartheta}_0$. If the η_t's are independent of the past, then

$$\boldsymbol{\Sigma} = \Big(E(\eta_t^4) - 1)\Big)\,\boldsymbol{J}^{-1}, \tag{10}$$

This is essentially the univariate version of the non-normality robust coefficient-covariance of Bollerslev and Wooldridge (1992). It is easily estimated since the standardized residuals can be used to obtain an estimate of $E(\eta_t^4)$, and a numerical estimate of the Hessian $\boldsymbol{J}$ is returned by the optimizer. In the **garchx** package, the estimate of (10) is referred to as the `"ordinary"` coefficient-covariance. Of course, the expression returned by the software is the estimate of the finite sample counterpart $\boldsymbol{\Sigma}/T$, where T is the sample size. In other words, the standard errors are equal to the square root of the diagonal of

the estimate $\widehat{\mathbf{\Sigma}}/T$. If, instead, the η_t's are not independent of the past, then

$$\mathbf{\Sigma} = \boldsymbol{J}^{-1}\boldsymbol{I}\boldsymbol{J}^{-1}, \qquad \boldsymbol{I} = E\left[\left\{E\left(\frac{\epsilon_t^4}{\sigma_t^4(\boldsymbol{\vartheta}_0)}\Big|\mathcal{F}_{t-1}\right) - 1\right\}\frac{1}{\sigma_t^4(\boldsymbol{\vartheta}_0)}\frac{\partial\sigma_t^2(\boldsymbol{\vartheta}_0)}{\partial\boldsymbol{\vartheta}}\frac{\partial\sigma_t^2(\boldsymbol{\vartheta}_0)}{\partial\boldsymbol{\vartheta}'}\right]. \tag{11}$$

In the **garchx** package, the estimate of this expression is referred to as the `"robust"` coefficient-covariance. Again, the expression returned by the software is the estimate of the finite sample counterpart $\mathbf{\Sigma}/T$. It should be noted that the estimation of (11) is computationally much more demanding than (10) since an estimate of $\partial\sigma_t^2(\boldsymbol{\vartheta}_0)/\partial\boldsymbol{\vartheta}$ in $\boldsymbol{I}$ is computed at each t. More details about how this is implemented is contained in the Appendix.

Basic usage of garchx

For illustration, the `spyreal` dataset in the **rugarch** package is used, which contains two daily financial time series: The SPDR SP500 index open-to-close return and the realized kernel volatility. The data are from Hansen et al. (2012) and goes from 2002-01-02 to 2008-08-29. The following code loads the data and stores the daily return – in percent – in an object named `eps`:

```
library(xts)
data(spyreal, package = "rugarch")
eps <- spyreal[,"SPY_OC"]*100
```

Note that the data `spyreal` is an object of class **xts** (Ryan and Ulrich, 2014). Accordingly, the object `eps` is also of class `xts`.

The basic interface of **garchx** is similar to that of `garch()` in **tseries**. For example, the code

```
garchx(eps)
```

estimates a plain GARCH(1,1), and returns a print of the result (implicitly, `print.garchx()` is invoked):

```
Date: Wed Apr 15 09:19:41 2020
Method: normal ML
Coefficient covariance: ordinary
Message (nlminb): relative convergence (4)
No. of observations: 1661
Sample: 2002-01-02 to 2008-08-29

              intercept      arch1     garch1
Estimate:   0.005945772 0.05470749 0.93785529
Std. Error: 0.002797459 0.01180603 0.01349976

Log-likelihood: -2014.6588
```

Alternatively, the estimation result can be stored to facilitate the subsequent extraction of information:

```
mymod <- garchx(eps)
coef(mymod)       #coefficient estimates
fitted(mymod)     #fitted conditional variance
logLik(mymod)     #log-likelihood (i.e., not the average log-likelihood)
nobs(mymod)       #no. of observations
predict(mymod)    #generate predictions of the conditional variance
print(mymod)      #print of estimation result
quantile(mymod)   #fitted quantile(s), the default corresponds to 97.5% value-at-risk
residuals(mymod)  #standardized residuals
summary(mymod)    #summarise with summary.default
toLatex(mymod)    #LaTeX print of result (equation form)
vcov(mymod)       #coefficient-covariance
```

The series returned by `fitted`, `quantile`, and `residuals` are of class **zoo** (Zeileis and Grothendieck, 2005).

To control the ARCH, GARCH, and asymmetry orders, the argument `order`, which takes a vector of length 1, 2, or 3, can be used in a similar way to as in the `garch()` function of **tseries**:

- `order[1]` controls the GARCH order
- `order[2]` controls the ARCH order
- `order[3]` controls the asymmetry order

For example, the following code estimate, respectively, a GARCH(1,1) with asymmetry and a GARCH(2,1) without asymmetry:

```
garchx(eps, order = c(1,1,1))  #garch(1,1) w/asymmetry
garchx(eps, order = c(1,2))    #garch(2,1)
```

To illustrate how covariates can be included via the `xreg` argument, the lagged realized volatility from the `spyreal` dataset can be used:

```
x <- spyreal[,"SPY_RK"]*100
xlagged <- lag(x)  #this lags, since x is an xts object
xlagged[1] <- 0    #replace NA-value with 0
```

The code

```
garchx(eps, xreg = xlagged)
```

estimates a GARCH(1,1) with the lagged realized volatility as covariate, i.e.,

$$\sigma_t^2 = \omega + \alpha_1 \epsilon_{t-1}^2 + \beta_1 \sigma_{t-1}^2 + \lambda_1 x_{1,t-1}, \tag{12}$$

and returns the print

```
Date: Wed Apr 15 09:26:46 2020
Method: normal ML
Coefficient covariance: ordinary
Message (nlminb): relative convergence (4)
No. of observations: 1661
Sample: 2002-01-02 to 2008-08-29

             intercept      arch1     garch1     SPY_RK
Estimate:   0.01763853 0.00000000 0.71873142 0.28152520
Std. Error: 0.01161863 0.03427413 0.09246282 0.08558003

Log-likelihood: -1970.247
```

The estimates suggest the ARCH parameter α_1 is 0. In a t-test with $\alpha_1 = 0$ as the null hypothesis, the parameter lies on the boundary of the permissible parameter space under the null. Accordingly, inference is non-standard and below I illustrate how this can be carried out with the `ttest0()` function. Note that if α_1 is, indeed, 0, then the specification reduces to the non-exponential Realised GARCH of Hansen et al. (2012). Below I illustrate how it can be estimated by simply omitting the ARCH term, i.e., by imposing a zero-coefficient restriction via omission.

The `"ordinary"` coefficient-covariance is the default. To instead use the dependence robust coefficient-covariance, set the `vcov.type` argument to `"robust"`:

```
garchx(eps, xreg = xlagged, vcov.type = "robust")
```

The associated print

```
Date: Wed Apr 15 09:31:12 2020
Method: normal ML
Coefficient covariance: robust
Message (nlminb): relative convergence (4)
No. of observations: 1661
Sample: 2002-01-02 to 2008-08-29

             intercept      arch1    garch1    SPY_RK
Estimate:   0.01763853 0.00000000 0.7187314 0.2815252
Std. Error: 0.01864470 0.04569981 0.1507067 0.1136347

Log-likelihood: -1970.247
```

reveals the standard errors change, but not dramatically. If the estimation result had been stored in an object with, say, the command `mymod <-garchx(eps,xreg = xlagged)`, then the robust coefficient-covariance could instead have been extracted by the code `vcov(mymod,vcov.type = "robust")`.

Inference under nullity

If the value of a parameter is zero under the null hypothesis, then it lies on the boundary of the permissible parameter space. In these cases, the non-standard inference is required, see Francq and

Thieu (2018). The **garchx** package offers two functions to facilitate such non-standard tests, `ttest0()`, and `waldtest0()`.

Recall that $\boldsymbol{\vartheta}_0$ denotes the d-dimensional vector of true parameters. In a plain GARCH(1,1), for example, $d = 3$. Next, let $\boldsymbol{e}_k$ denote a $d \times 1$ vector whose elements are all 0 except element k, which is 1. The function `ttest0()` undertakes the following t-test of parameter $k \geq 2$:

$$H_0 : \boldsymbol{e}_k' \boldsymbol{\vartheta}_0 = 0 \quad \text{and} \quad \boldsymbol{e}_l' \boldsymbol{\vartheta}_0 > 0 \ \ \forall l \neq k \qquad \text{against} \qquad H_A : \boldsymbol{e}_k' \boldsymbol{\vartheta}_0 > 0.$$

Note that, in this test, all parameters – except parameter k – are assumed to be greater than 0 under the null. While the test-statistic is the usual one, the p-value is obtained by only considering the positive part of the normal distribution. To illustrate the usage of `ttest0`, let us revisit the GARCH(1,1)-X model in (12):

```
mymod <- garchx(eps, xreg = xlagged)
```

In this model, the non-exponential Realized GARCH of Hansen et al. (2012) is obtained when the ARCH(1)-parameter α_1 is 0. This is straightforwardly tested with `ttest0(mymod,k = 2)`, which yields

```
      coef  std.error t-stat p-value
arch1    0 0.03427413      0     0.5
```

In other words, at the most common significance levels, the result supports the claim that $\alpha_1 = 0$. Finally, note that if the user does not specify k, then the code `ttest0()` returns a t-test of all the coefficients except the intercept ω.

The function `waldtest0()` can be used to test whether one or more coefficients are zero. Let $\boldsymbol{r}$ denote the restriction vector of dimension $r_0 \times 1$, and let $\boldsymbol{R}$ denote the combination matrix of dimension $r_0 \times d$. Assuming that $\boldsymbol{R}$ has full row-rank, the null and alternative hypotheses in the Wald-test are given by

$$H_0 : \boldsymbol{R}\boldsymbol{\vartheta}_0 = \boldsymbol{r} \qquad \text{against} \qquad H_A : \boldsymbol{R}\boldsymbol{\vartheta}_0 \neq \boldsymbol{r}.$$

The associated Wald test-statistic has the usual form, but the distribution is non-standard (Francq and Thieu, 2018):

$$W_T = (\boldsymbol{R}\widehat{\boldsymbol{\vartheta}} - \boldsymbol{r})' \boldsymbol{R} (\widehat{\boldsymbol{\Sigma}}/T) \boldsymbol{R}' (\boldsymbol{R}\widehat{\boldsymbol{\vartheta}} - \boldsymbol{r}), \qquad W_T \xrightarrow{d} W = ||\boldsymbol{R}\boldsymbol{Z}||^2, \qquad \boldsymbol{Z} \sim N(\boldsymbol{0}, \boldsymbol{\Sigma}).$$

Critical values are obtained by parametric Bootstrap. First, the sequence

$$\left\{ ||\boldsymbol{R}\widehat{\boldsymbol{Z}}_i||^2, i = 1, \ldots, n \right\}$$

is simulated, where the $\widehat{\boldsymbol{Z}}_i$'s are independent and identically distributed $N(\boldsymbol{0}, \widehat{\boldsymbol{\Sigma}})$ vectors. In `waldtest0()`, the default is $n = 20000$. Next, the critical value associated with significance level $\alpha \in (0, 1)$ is obtained by computing the empirical (1-α)-quantile of the simulated values. To illustrate the usage of `waldtest0()`, let us reconsider the GARCH(1,1)-X model in (12). Specifically, let us test whether both the ARCH and GARCH coefficients are zero: $H_0 : \alpha_1 = 0$ and $\beta_1 = 0$. This means

```
r <- cbind(c(0,0))
R <- rbind(c(0,1,0,0),c(0,0,1,0))
```

Next, the command `waldtest0(mymod,r = r,R = R)` performs the test and returns a list with the statistic and critical values:

```
$statistic
[1] 72.95893

$critical.values
10%       5%       1%
41.79952 57.97182 97.15217
```

In other words, the Wald statistic is 72.96 and the critical values associated with the 10%, 5%, and 1% levels, respectively, are 41.80, 57.97, and 97.15. So H_0 is rejected at the 10% and 5% levels, but not at the 1% level. If the user wishes to do so, the significance levels can be changed via the `level` argument.

Zero-coefficient restrictions via omission

The ARCH, GARCH, and asymmetry orders can be specified in two ways. Either via the `order` argument as illustrated above or via the `arch`, `garch`, and `asym` arguments whose defaults are all `NULL`. If any of their values is not `NULL`, then it takes precedence over the corresponding component in `order`. For example, the code

```
garchx(eps, order = c(0,0), arch = 1, garch = 1)
```

estimates a GARCH(1,1) since the values of `arch` and `garch` override those of `order[2]` and `order[1]`, respectively. Similarly, `garchx(eps,asym = 1)` estimates a GARCH(1,1) with asymmetry, and `garchx(eps,garch = 0)` estimates a GARCH(1,0) model.

To estimate higher-order models with the `arch`, `garch`, and `asym` arguments, the lags must be provided explicitly. For example, to estimate the GARCH(2,2) model $\sigma_t^2 = \omega + \alpha_1\epsilon_{t-1}^2 + \alpha_2\epsilon_{t-2}^2 + \beta_1\sigma_{t-1}^2 + \beta_2\sigma_{t-2}^2$, use

```
garchx(eps, arch = c(1,2), garch = c(1,2))
```

Zero-coefficient constraints, therefore, can be imposed by simply omitting the lags in question. For example, to estimate the GARCH(2,2) model with $\alpha_1 = \beta_1 = 0$, use

```
garchx(eps, arch = 2, garch = 2)
```

This returns the print

```
Date: Wed Apr 15 09:34:04 2020
Method: normal ML
Coefficient covariance: ordinary
Message (nlminb): relative convergence (4)
No. of observations: 1661
Sample: 2002-01-02 to 2008-08-29

              intercept      arch2     garch2
Estimate:   0.009667606 0.07533534 0.91392791
Std. Error: 0.004494075 0.01636917 0.01899654

Log-likelihood: -2033.7251
```

To estimate the non-exponential Realized GARCH of Hansen et al. (2012), use

```
garchx(eps, arch = 0, xreg = xlagged)
```

The returned print shows that the ARCH(1) term has not been included during the estimation.

Finally, a caveat is in order. The flexibility provided by the `arch`, `garch`, and `asym` arguments is not always warranted by the underlying estimation theory. For example, if the ARCH-parameter α_1 in a plain GARCH(1,1) model is restricted to zero, then the normal ML estimator is invalid. The `garchx()` function, nevertheless, tries to estimate it if the user provides the code `garchx(eps,arch = 0)`. Currently, the function `garchx()` does not undertake any checks of whether the zero-coefficient restrictions are theoretically valid.

Numerical optimization

The two optimization algorithms in base R that work best for GARCH estimation are, in my experience, the `"Nelder-Mead"` method in the `optim()` function and the `nlminb()` function. The latter enables bounded optimization, so it is the preferred algorithm here since the parameters of the GARCH model must be non-negative. The `"L-BFGS-B"` method in `optim()` also enables bounded optimization, but it does not work as well in my experience. When using the `garchx()` function, the call to `nlminb()` can be controlled and tuned via the arguments `initial.values`, `lower`, `upper`, and `control`. In `nlminb()`, the first argument is named `start`, whereas the other three are equal.

Suitable initial parameter values are important for numerical robustness. In the `garchx()` function, the user can set these via the `initial.values` argument. If not, then they are automatically determined internally. In the case of a GARCH(1,1), the default initial values are $\omega = 0.1$, $\alpha_1 = 0.1$, and $\beta_1 = 0.7$. For numerical robustness, it is important that they are not too close to the lower boundary of 0 and that β_1 is not too close to instability, i.e., $\beta_1 \geq 1$. The choice `c(0.1,0.1,0.7)` works well across a range of problems. Indeed, the Monte Carlo simulations of the large sample properties of the packages (see Section 3) reveals that the numerical robustness of **tseries** improves when these initial values are used instead of the default initial values. In the `list` returned by `garchx()`, the item named `initial.values` contains the values used. For example, the following code extracts the initial values used in the estimation of a GARCH(1,1) with asymmetry:

```
mymod <- garchx(eps, asym = 1)
mymod$initial.values
```

In each iteration, `nlminb()` calls the function `garchxObjective()` to evaluate the objective function. For additional numerical robustness, checks of the parameters and fitted conditional variance are conducted within `garchxObjective()` at each iteration. The first check is for whether any of the parameter values at the current iteration are equal to `NA`. The second check is for whether any of the fitted conditional variances are `Inf`, 0, or negative. If either of these checks fails, then `garchxObjective()` returns the value of the `logl.penalty` argument in the `garchx()` function, whose default value is that produced by the initial values. To avoid that the term $\ln \sigma_t^2$ in the objective function explodes to minus infinity, the fitted values of σ_t^2 are restricted to be equal or greater than the value provided by the `sigma2.min` argument in the `garchx()` function.

A drawback with `nlminb()` is that it does not return an estimate of the Hessian at the optimum, which is needed to compute the coefficient-covariance. To obtain such an estimate, the `optimHess()` function is used. In `garchx()`, the call to `optimHess()` can be controlled and tuned via the `optim.control` argument. Next, the inverse of the estimated Hessian is computed with `solve()`, whose tolerance for detecting linear dependencies in the columns is determined by the `solve.tol` argument in the `garchx()` function.

Checking the large sample properties

The function `garchxAvar()` returns the asymptotic coefficient-covariance of a GARCH-X model. Currently (version 1.1), only non-normality robust versions are available. The aim of this section is to illustrate how it can be used to check whether the large sample properties of the packages correspond to those of the underlying asymptotic estimation theory. Specifically, the aim is to explore whether large sample estimates from Monte Carlo simulations are unbiased, whether the empirical standard errors correspond to the asymptotic ones, and whether the estimate of the non-normality robust coefficient-covariance is unbiased.

The `garchxAvar()` function

To recall, the non-normality robust asymptotic coefficient-covariance is given by

$$\boldsymbol{\Sigma} = \Big(E(\eta_t^4) - 1)\Big)\, \boldsymbol{J}^{-1}, \qquad \boldsymbol{J} = E\left(\frac{\partial^2 l_t(\boldsymbol{\vartheta}_0)}{\partial \boldsymbol{\vartheta} \partial \boldsymbol{\vartheta}'}\right)$$

when the η_t's are independent of the past. In general, the expression for $\boldsymbol{J}$ is not available in closed form. Accordingly, numerical methods are needed. The `garchxAvar()` function combines simulation and numerical differentiation to compute $\boldsymbol{\Sigma}$. In short, the function proceeds by first simulating n values of ϵ_t (the default is $n = 10$ million), and then the Hessian of the criterion function $n^{-1}\sum_{t=1}^{n} l_t(\boldsymbol{\vartheta})$ about the true value $\boldsymbol{\vartheta}_0$ is obtained by numerical differentiation to compute an estimate of $\boldsymbol{J}$. Internally, the `garchxAvar()` function conducts the simulation with `garchxSim()` and the differentiation with `optimHess()`. If we denote the numerically obtained Hessian as $\tilde{\boldsymbol{J}}$, then the corresponding finite-sample counterpart of the asymptotic coefficient-covariance associated with a sample of size T is given by

$$\frac{1}{T}\Big(E(\eta_t^4) - 1\Big)\, \tilde{\boldsymbol{J}}^{-1}. \tag{13}$$

In other words, the square root of the diagonal of this expression is the asymptotic standard error associated with sample size T.

To obtain an idea about the precision of `garchxAvar()`, a numerical comparison is made for the case where the DGP is an ARCH(1) with standard normal innovations:

$$\epsilon_t = \sigma_t \eta_t, \qquad \eta_t \overset{iid}{\sim} N(0,1), \qquad \sigma_t^2 = \omega + \alpha_1 \epsilon_{t-1}^2. \tag{14}$$

In this case, it can be shown that

$$\boldsymbol{J} = E\begin{bmatrix} \frac{1}{(\omega+\alpha_1\epsilon_{t-1}^2)^2} & \frac{\epsilon_{t-1}^2}{(\omega+\alpha_1\epsilon_{t-1}^2)^2} \\ \frac{\epsilon_{t-1}^2}{(\omega+\alpha_1\epsilon_{t-1}^2)^2} & \frac{\epsilon_{t-1}^4}{(\omega+\alpha_1\epsilon_{t-1}^2)^2} \end{bmatrix}; \tag{15}$$

see (Francq and Zakoïan, 2019, pp. 180-181). In other words, in this specific case, it is straightforward to obtain a numerical estimate of $\boldsymbol{J}$ without having to resort to numerical derivatives (as in `garchxAvar()`) by simply computing the means of the sample counterparts. For an ARCH(1) with $(\omega, \alpha_1) = (1, 0.1)$, the code :

```
n <- 10000000
```

```
omega <- 1; alpha1 <- 0.1
set.seed(123)
eta <- rnorm(n)
eps <- garchxSim(n, intercept = omega, arch = alpha1, garch = NULL,
        innovations = eta)

epslagged2 <- eps[-length(eps)]^2
epslagged4 <- epslagged2^2
J <- matrix(NA, 2, 2)
J[1,1] <- mean( 1/( (omega+alpha1*epslagged2)^2 ) )
J[2,1] <- mean( epslagged2/( (omega+alpha1*epslagged2)^2 ) )
J[1,2] <- J[1,2]
J[2,2] <- mean( epslagged4/( (omega+alpha1*epslagged2)^2 ) )
Eeta4 <- 3
Avar1 <- (Eeta4-1)*solve(J)
```

computes the asymptotic coefficient-covariance, and stores it in an object named `Avar1`:

```
Avar1
          [,1]      [,2]
[1,]  3.475501 -1.368191
[2,] -1.368191  1.686703
```

With `garchxAvar()`, using the same simulated series for η_t, we obtain

```
Avar2 <- garchxAvar(c(omega,alpha1), arch=1, Eeta4=3, n=n, innovations=eta)
Avar2
          intercept     arch1
intercept  3.474903 -1.367301
arch1     -1.367301  1.685338
```

These are quite similar in relative terms since the ratio `Avar2/Avar1` shows each entry in `Avar2` is less than 0.1% away from those of `Avar1`.

Bias and standard errors of estimates

To illustrate how `garchxAvar()` can be used to study the large sample properties of the packages, a Monte Carlo study is undertaken. The DGP in the study is a plain GARCH(1,1) with either $\eta_t \sim N(0,1)$ or $\eta_t \sim$ standardized $t(5)$, and the sample size is $T = 10000$:

$$\epsilon_t = \sigma_t \eta_t, \qquad \eta_t \overset{iid}{\sim} N(0,1) \quad \text{or} \quad \eta_t \overset{iid}{\sim} \text{standardized } t(5), \qquad t = 1, \ldots, T = 10000,$$
$$\sigma_t^2 = \omega + \alpha_1 \epsilon_{t-1}^2 + \beta_1 \sigma_{t-1}^2, \qquad (\omega, \alpha_1, \beta_1) = (0.2, 0.1, 0.8).$$

The values of $(\omega, \alpha_1, \beta_1)$ are similar to those that are usually found in empirical studies of financial returns. The code

```
n <- 10000000
pars <- c(0.2, 0.1, 0.8)
set.seed(123)
AvarNormal <- garchxAvar(pars, arch=1, garch=1, Eeta4=3, n=n)
eta <- rt(n, df=5)/sqrt(5/3)
Avart5 <- garchxAvar(pars, arch=1, garch=1, Eeta4=9, n=n,
        innovations=eta)
```

computes and stores the asymptotic coefficient-covariances in objects named `AvarNormal` and `Avart5`, respectively. They are:

```
AvarNormal
          intercept      arch1    garch1
intercept  7.043653  1.1819890 -4.693843
arch1      1.181989  0.7784797 -1.278153
garch1    -4.693843 -1.2781529  3.616365

Avart5
           intercept     arch1     garch1
intercept  16.234885  3.216076 -11.313749
```

	$m(\widehat{\omega})$	$se(\widehat{\omega})$	$ase(\widehat{\omega})$	$m(\widehat{\alpha}_1)$	$se(\widehat{\alpha}_1)$	$ase(\widehat{\alpha}_1)$	$m(\widehat{\beta}_1)$	$se(\widehat{\beta}_1)$	$ase(\widehat{\beta}_1)$	n(NA)
$N(0,1)$:										
tseries	0.218	0.160	0.027	0.100	0.010	0.009	0.791	0.082	0.019	0
fGarch	0.203	0.027	0.027	0.100	0.009	0.009	0.799	0.019	0.019	0
rugarch	0.204	0.027	0.027	0.100	0.009	0.009	0.797	0.019	0.019	0
garchx	0.203	0.027	0.027	0.100	0.009	0.009	0.798	0.019	0.019	0
$t(5)$:										
tseries	0.218	0.158	0.040	0.101	0.015	0.016	0.791	0.077	0.030	0
fGarch	0.204	0.039	0.040	0.101	0.016	0.016	0.797	0.030	0.030	0
rugarch	0.201	0.037	0.040	0.100	0.014	0.016	0.799	0.027	0.030	2
garchx	0.201	0.037	0.040	0.100	0.015	0.016	0.799	0.028	0.030	0

Table 2: Comparison of the large sample properties of **tseries** version 0.10-47 (Trapletti and Hornik, 2019), **fGarch** version R 3.0.1 (Wuertz et al., 2020), **rugarch** version 1.4-2 (Ghalanos, 2020), and **garchx** version 1.1 (Sucarrat, 2020). $m(\cdot)$: sample average of estimates. $se(\cdot)$: sample standard deviation of estimates. $ase(\cdot)$: asymptotic standard error. n(NA): the number of times estimation failed due to numerical issues.

```
arch1       3.216076  2.483018  -3.647237
garch1    -11.313749 -3.647237   9.239820
```

Next, the asymptotic standard errors associated with sample size $T = 10000$ are obtained with

```
sqrt( diag(AvarNormal/10000) )
sqrt( diag(Avart5/10000) )
```

These values are contained in the columns labelled $ase(\cdot)$ in Table 2.

Table 2 contains the estimation results of the Monte Carlo study (1000 replications). For each package, normal ML estimation is undertaken with default options on initial parameter values, initial recursion values, and numerical control. The columns labelled $m(\cdot)$ contain the sample average across the replications, and $se(\cdot)$ contains the sample standard deviation. Apart from **tseries**, the simulations suggest the packages produce asymptotically unbiased estimates and empirical standard errors that correspond to the asymptotic ones. Closer examination suggests the biases and faulty empirical standard errors of **tseries** are due to outliers. Additional simulations, with non-default initial parameter values, produce results similar to those of the other packages.[3] This underlines the importance of suitable initial parameter values for numerical robustness. The package **rugarch** ran into numerical problems twice for $\eta_t \sim t(5)$, and thus, failed to returned estimates in these two cases. Additional simulations confirmed **rugarch** is less robust numerically than the other packages under its default options when $\eta_t \sim t(5)$, i.e., it always failed at least once. Changing the initial parameter values to those of **garchx** did not resolve the problem. Also, changing the optimizer to a non-default algorithm, `nlminb()`, which is the default algorithm in **fGarch** and the only option available in **garchx**, produced more failures and substantially biased results by **rugarch**.[4]

Coefficient-covariance estimate

In each of the 1000 replications of the Monte Carlo study, the estimate of the asymptotic coefficient-covariance is recorded. For **fGarch**, **rugarch**, and **garchx**, the estimate is of the non-normality robust type. For **tseries**, which does not offer the non-normality robust option, the estimate is under the assumption of normality. Note also that, for **tseries**, the results reported here are with the numerically more robust non-default initial parameter values alluded to above.

Let $\widehat{\mathbf{\Sigma}}_i$ denote the estimate produced by a package in replication $i = 1, \ldots, 1000$ of the simulations. The relative bias in replication i is given by the ratio $\widehat{\mathbf{\Sigma}}_i / \mathbf{\Sigma}$, a 3×3 matrix, which is obtained by dividing the row i column j component in $\widehat{\mathbf{\Sigma}}_i$ by the corresponding component in $\mathbf{\Sigma}$. The average relative bias,

[3] The additional simulations are not reported, but they are readily conducted by minor modifications to the replication files. Specifically, the code `garch(eps)` needs to be modified to `garch(eps, control = garch.control(start = c(0.1, 0.1, 0.7)))`.

[4] In the replication code, these results are reproduced by changing the estimation command from `ugarchfit(data=eps, spec=spec)` to `ugarchfit(data=eps, spec=spec, solver="nlminb")`.

$m(\widehat{\mathbf{\Sigma}}/\mathbf{\Sigma})$, is obtained by taking the average across the 1000 replications for each of the 9 entries. When $\eta_t \sim N(0,1)$, this produces the following averages:

```
##tseries:
          intercept  arch1 garch1
intercept    1.0702 1.0489 1.0656
arch1        1.0489 1.0256 1.0366
garch1       1.0656 1.0366 1.0566

##fGarch:
          intercept  arch1 garch1
intercept    1.0596 1.0335 1.0548
arch1        1.0335 1.0126 1.0229
garch1       1.0548 1.0229 1.0455

##rugarch:
          intercept  arch1 garch1
intercept    1.0869 1.0723 1.0848
arch1        1.0723 1.0280 1.0501
garch1       1.0848 1.0501 1.0748

##garchx:
          intercept  arch1 garch1
intercept    1.0630 1.0350 1.0576
arch1        1.0350 1.0142 1.0244
garch1       1.0576 1.0244 1.0479
```

Three general characteristics are clear. First, the ratios are all greater than 1. In other words, all packages tend to return estimated coefficient-covariances that are too large in absolute terms. In particular, standard errors tend to be too high. Second, the size of the biases is similar across packages. Those of **rugarch** are slightly higher than those of the other packages, but the difference may disappear if a larger number of replications is used. Third, the magnitude of the relative bias is fairly low since they all lie between 1.26% and 8.69%.

When $\eta_t \sim t(5)$, the simulations produce the following averages:

```
##tseries:
          intercept  arch1 garch1
intercept    0.1082 0.1038 0.1088
arch1        0.1038 0.0952 0.1002
garch1       0.1088 0.1002 0.1070

##fGarch:
          intercept  arch1 garch1
intercept    0.9088 1.0198 0.9098
arch1        1.0198 1.0721 0.9858
garch1       0.9098 0.9858 0.9062

##rugarch:
          intercept  arch1 garch1
intercept    0.8423 0.8596 0.8356
arch1        0.8596 0.8361 0.8349
garch1       0.8356 0.8349 0.8263

##garchx:
          intercept  arch1 garch1
intercept    0.9343 0.9017 0.9200
arch1        0.9017 0.8973 0.8903
garch1       0.9200 0.8903 0.9043
```

The downwards relative bias of about 90% produced by **tseries** simply reflects that a non-normality robust option is not available in that package. However, the size of the bias is larger than expected. If it were simply due to $E(\eta_t^4)$ in the expression for $\mathbf{\Sigma}$ being erroneous (3 instead of 9 in the simulations), then the ratios should have been in the vicinity of $(3-1)/(9-1) = 0.29$. Instead, they are substantially lower, since they all lie in the vicinity of 0.10. In other words, the way estimation is implemented by **tseries** induces a downward bias in the standard errors that can be substantially larger than expected when η_t is fat-tailed. The relative bias produced by **fGarch**, **rugarch**, and **garchx** is more moderate

since they all lie less than 18% away from the true values. While the relative bias of **rugarch** is slightly larger than those of **fGarch** and **garchx**, the general tendency of the three packages is that the bias is downwards.

Comparison of speed

In nominal terms, all four packages are fairly fast. On an average contemporary laptop, for example, estimation of a plain GARCH(1,1) usually takes less than a second if the number of observations is 10000 or less. The reason is that all four packages use compiled C/C++ or Fortran code in the recursion, i.e., the computationally most demanding part. While the nominal speed difference is almost unnoticeable in simple models with small T, the relative difference among the packages is significant. In other words, when T is large or when a large number of models are estimated (as in Monte Carlo simulations), then the choice of a package can be important.

The comparison is undertaken with the **microbenchmark** (Mersmann, 2019) package version 1.4-7, and the average estimation-time of four GARCH models are compared:

$$\epsilon_t = \sigma_t \eta_t, \qquad \eta_t \overset{iid}{\sim} N(0,1),$$

1	GARCH(1,1):	$\sigma_t^2 = \omega + \alpha_1 \epsilon_{t-1}^2 + \beta_1 \sigma_{t-1}^2$
2	GARCH(2,2):	$\sigma_t^2 = \omega + \sum_{i=1}^2 \alpha_i \epsilon_{t-i}^2 + \sum_{j=1}^2 \beta_j \sigma_{t-j}^2$
3	GARCH(1,1) w/asymmetry:	$\sigma_t^2 = \omega + \alpha_1 \epsilon_{t-1}^2 + \beta_1 \sigma_{t-1}^2 + \gamma_1 I_{\{\epsilon_{t-1}<0\}} \epsilon_{t-1}^2$
4	GARCH(1,1)-X:	$\sigma_t^2 = \omega + \alpha_1 \epsilon_{t-1}^2 + \beta_1 \sigma_{t-1}^2 + \lambda_1 x_{t-1}$

The parameters of the Data Generating Processes (DGPs) are

$$(\omega, \alpha_1, \beta_1, \alpha_2, \beta_2, \gamma_1, \lambda_1) = (0.2, 0.1, 0.8, 0.00, 0.00, 0.05, 0.3),$$

and x_t in DGP number 4 is governed by the AR(1) process

$$x_t = 0.5 x_{t-1} + 0.1 u_t, \qquad u_t \overset{iid}{\sim} N(0,1).$$

The comparison is made for sample sizes $T = 1000$ and $T = 2000$.

Table 3 contains the results of the comparison in relative terms. A value of 1.0 means the package is the fastest on average for the experiment in question. A value of 7.15 means the average estimation time of the package is 7.15 times larger than the average of the fastest, and so on. The entry is empty if the GARCH specification cannot be estimated by the package.The overall pattern of the results is clear: **tseries** is the fastest among the models it can estimate, **garchx** is the second fastest, **fGarch** is the third fastest, and **rugarch** is the slowest. Another salient feature is how much faster **tseries** is relative to the other packages. This is particularly striking for the GARCH(2,2), where the second-fastest package – **garchx** – is about 5 to 6 times slower, and the slowest package – **rugarch** – is about 28 to 30 times slower. A third notable characteristic is that the relative differences tend to fall as the sample size T increases. For example, **rugarch** is about 17 times slower than **tseries** when $T = 1000$, but only about 13 times slower when $T = 2000$. As for the specifications that **tseries** are not capable of estimating, **garchx** is the fastest. Notably so compared with **fGarch** and even more so compared with **rugarch**.

Conclusions

This paper provides an overview of the package **garchx** and compares it with three prominent CRAN-packages that offer GARCH estimation routines: **tseries**, **fGarch**, and **rugarch**. While **garchx** does not offer all the GARCH-specifications available in **rugarch**, it is much more flexible than **tseries**, and it also offers the important possibility of including covariates. This feature is not available in **fGarch**. The package **garchx** also offers additional features that are not available in the other packages: i) A dependence-robust coefficient covariance, ii) functions that facilitate hypothesis testing under nullity, iii) zero-coefficient restrictions by omission, and iv) a function that computes the asymptotic coefficient-covariance of a GARCH model.

In a Monte Carlo study of the packages, the large sample properties of the normal Quasi ML (QML) estimator were studied. There, it was revealed that **fGarch** and **garchx** are numerically more robust (under default options) than **tseries** and **rugarch**. However, in the case of **tseries**, the study also

DGP	T	**tseries**	**fGarch**	**rugarch**	**garchx**
1 GARCH(1, 1):	1000	1.00	7.15	17.42	2.69
	2000	1.00	6.28	12.89	1.85
2 GARCH(2, 2):	1000	1.00	10.14	29.78	5.27
	2000	1.00	14.72	27.79	6.27
3 GARCH(1, 1, 1):	1000		2.26	14.72	1.00
	2000		2.97	9.91	1.00
4 GARCH(1, 1)-X:	1000			5.90	1.00
	2000			6.36	1.00

Table 3: Relative speed comparison of **tseries** version 0.10-47 (Trapletti and Hornik, 2019), **fGarch** version R 3.0.1 (Wuertz et al., 2020), **rugarch** version 1.4-2 (Ghalanos, 2020), and **garchx** version 1.1 (Sucarrat, 2020). A value of 1.00 means the package is the fastest on average for the experiment in question. A value of 7.15 means the average estimation time of the package is 7.15 times larger than the average of the fastest, and so on. The entry is empty if the GARCH specification cannot be estimated by the package.

revealed how its numerical robustness could be improved straightforwardly by simply changing the initial parameter values. In the case of **rugarch**, it is less clear how the numerical robustness can be improved. The study also revealed that the standard errors of **tseries** could be substantially biased downwards when η_t is non-normal. A bias is expected since **tseries** does not offer a non-normality robust coefficient-covariance. However, the bias is larger than suggested by the underlying estimation theory.

In a relative speed comparison of the packages, it emerged that the least flexible package – **tseries** – is notably faster than the other packages. Next, **garchx** is the second fastest (1.85 to 6.27 times slower in the experiments), **fGarch** is the third fastest, and **rugarch** is the slowest. The experiments also revealed that the difference could be larger in higher-order models. For example, in the estimation of a GARCH(2,2), **rugarch** was about 28 times slower than **tseries**. In estimating a plain GARCH(1,1), by contrast, it was only 13 to 17 times slower. Another finding was that the difference seems to fall in sample size: The larger the sample size, the smaller the difference in speed.

Bibliography

T. Bollerslev. Generalized autoregressive conditional heteroscedasticity. *Journal of Econometrics*, 31: 307–327, 1986. [p276]

T. Bollerslev and J. Wooldridge. Quasi-Maximum Likelihood Estimation and Inference in Dynamic Models with Time Varying Covariances. *Econometric Reviews*, 11:143–172, 1992. [p277, 278, 279]

C. Brownlees, F. Cipollini, and G. Gallo. Multiplicative Error Models. In L. Bauwens, C. Hafner, and S. Laurent, editors, *Handbook of Volatility Models and Their Applications*, pages 223–247. Wiley, New Jersey, 2012. [p276]

F. C. Drost and T. E. Nijman. Temporal Aggregation of Garch Processes. *Econometrica*, 61:909–927, 1993. [p279]

R. Engle. Autoregressive Conditional Heteroscedasticity with Estimates of the Variance of United Kingdom Inflations. *Econometrica*, 50:987–1008, 1982. [p276]

R. F. Engle and J. R. Russell. Autoregressive Conditional Duration: A New Model of Irregularly Spaced Transaction Data. *Econometrica*, 66:1127–1162, 1998. [p276]

J. C. Escanciano. Quasi-maximum likelihood estimation of semi-strong GARCH models. *Econometric Theory*, 25:561–570, 2009. [p278]

C. Francq and L. Q. Thieu. Qml inference for volatility models with covariates. *Econometric Theory*, 2018. doi: https://doi.org/10.1017/S0266466617000512. https://doi.org/10.1017/S0266466617000512. [p278, 279, 281, 282, 290]

C. Francq and J.-M. Zakoïan. *GARCH Models*. Wiley, New York, 2019. 2nd. Edition. [p284]

J. Geweke. Modelling the Persistence of Conditional Variance: A Comment. *Econometric Reviews*, 5: 57–61, 1986. [p276]

A. Ghalanos. *rugarch: Univariate GARCH Models*, 2020. URL https://CRAN.R-project.org/package=rugarch. R package version 1.4-2. [p276, 279, 286, 289]

B. E. Hansen. Autoregressive Conditional Density Estimation. *International Economic Review*, 35: 705–730, 1994. [p279]

P. R. Hansen, Z. Huan, and H. H. Shek. Realized GARCH: A Joint Model for Returns and Realized Measures of Volatility. *Journal of Applied Econometrics*, 27:877–906, 2012. [p278, 280, 281, 282, 283]

O. Mersmann. *microbenchmark: Accurate Timing Functions*, 2019. URL https://CRAN.R-project.org/package=microbenchmark. R package version 1.4-7. [p288]

A. Milhøj. A Multiplicative Parametrization of ARCH Models. Research Report 101, University of Copenhagen: Institute of Statistics, 1987. [p276]

S. Pantula. Modelling the Persistence of Conditional Variance: A Comment. *Econometric Reviews*, 5: 71–73, 1986. [p276]

J. A. Ryan and J. M. Ulrich. *xts: eXtensible Time Series*, 2014. URL https://CRAN.R-project.org/package=xts. R package version 0.12-0. [p280]

G. Sucarrat. *garchx: Flexible and Robust GARCH-X Modelling*, 2020. URL https://CRAN.R-project.org/package=garchx. R package version 1.1. [p279, 286, 289]

G. Sucarrat and S. Grønneberg. Risk Estimation with a Time Varying Probability of Zero Returns. *Journal of Financial Econometrics*, 2020. Forthcoming. DOI: https://doi.org/10.1093/jjfinec/nbaa014. [p279]

A. Trapletti and K. Hornik. *tseries: Time Series Analysis and Computational Finance*, 2019. URL https://CRAN.R-project.org/package=tseries. R package version 0.10-47. [p276, 279, 286, 289]

D. Wuertz, T. Setz, Y. Chalabi, C. Boudt, P. Chausse, and M. Miklovac. *fGarch: Rmetrics - Autoregressive Conditional Heteroskedastic Modelling*, 2020. URL https://CRAN.R-project.org/package=fGarch. R package version 3042.83.2. [p276, 279, 286, 289]

A. Zeileis and G. Grothendieck. zoo: S3 infrastructure for regular and irregular time series. *Journal of Statistical Software*, 14(6):1–27, 2005. URL http://www.jstatsoft.org/v14/i06/. [p280]

Genaro Sucarrat
BI Norwegian Business School
Nydalsveien 37
0484 Oslo
Norway
genaro.sucarrat@bi.no

Appendix: Estimation of the dependence robust coefficient-covariance

Francq and Thieu (2018) show that

$$\begin{aligned} \boldsymbol{J} &= E\left(\frac{\partial^2 l_t(\boldsymbol{\vartheta}_0)}{\partial\boldsymbol{\vartheta}\partial\boldsymbol{\vartheta}'}\right) = E\left(\frac{1}{\sigma_t^4(\boldsymbol{\vartheta}_0)}\frac{\partial\sigma_t^2(\boldsymbol{\vartheta}_0)}{\partial\boldsymbol{\vartheta}}\frac{\partial\sigma_t^2(\boldsymbol{\vartheta}_0)}{\partial\boldsymbol{\vartheta}'}\right) \\ \boldsymbol{I} &= E\left[\left\{E\left(\eta_t^4|\mathcal{F}_{t-1}\right)-1\right\}\frac{1}{\sigma_t^4(\boldsymbol{\vartheta}_0)}\frac{\partial\sigma_t^2(\boldsymbol{\vartheta}_0)}{\partial\boldsymbol{\vartheta}}\frac{\partial\sigma_t^2(\boldsymbol{\vartheta}_0)}{\partial\boldsymbol{\vartheta}'}\right] \end{aligned}$$

This means

$$\begin{aligned} \boldsymbol{I} &= E\left[\left\{E\left(\eta_t^4|\mathcal{F}_{t-1}\right)\right\}\frac{1}{\sigma_t^4(\boldsymbol{\vartheta}_0)}\frac{\partial\sigma_t^2(\boldsymbol{\vartheta}_0)}{\partial\boldsymbol{\vartheta}}\frac{\partial\sigma_t^2(\boldsymbol{\vartheta}_0)}{\partial\boldsymbol{\vartheta}'}\right]-\boldsymbol{J} \\ &= E\left[\left(\frac{\epsilon_t^2}{\sigma_t^4(\boldsymbol{\vartheta}_0)}\frac{\partial\sigma_t^2(\boldsymbol{\vartheta}_0)}{\partial\boldsymbol{\vartheta}}\right)\left(\frac{\epsilon_t^2}{\sigma_t^4(\boldsymbol{\vartheta}_0)}\frac{\partial\sigma_t^2(\boldsymbol{\vartheta}_0)}{\partial\boldsymbol{\vartheta}}\right)'\right]-\boldsymbol{J} \end{aligned}$$

The computationally challenging part to estimate in $\boldsymbol{I}$ is $\partial\sigma_t^2(\boldsymbol{\vartheta}_0)/\partial\boldsymbol{\vartheta}$ since it entails the computation of a numerically differentiated gradient of a recursion at each t. In **garchx**, this is implemented with `numericDeriv()` in the `vcov.garchx()` function.

ROBustness In Network (robin): an R Package for Comparison and Validation of Communities

by Valeria Policastro, Dario Righelli, Annamaria Carissimo, Luisa Cutillo and Italia De Feis

Abstract In network analysis, many community detection algorithms have been developed. However, their implementation leaves unaddressed the question of the statistical validation of the results. Here, we present **robin** (ROBustness In Network), an R package to assess the robustness of the community structure of a network found by one or more methods to give indications about their reliability. The procedure initially detects if the community structure found by a set of algorithms is statistically significant and then compares two selected detection algorithms on the same graph to choose the one that better fits the network of interest. We demonstrate the use of our package on the American College Football benchmark dataset.

Introduction

Over the last twenty years, network science has become a strategic field of research thanks to the strong development of high-performance computing technologies. The activity and interaction of thousands of elements can now be measured simultaneously, allowing us to model cellular networks, social networks, communication networks, power grids, and trade networks, to cite a few examples. Different types of data will produce different types of networks in terms of structure, connectivity, and complexity. In the study of complex networks, a network is said to have a community structure if the nodes are densely connected within groups but sparsely connected between them (Girvan and Newman, 2002). The inference of the community structure of a network is an important task. Communities allow us to create a large-scale map of a network since individual communities act like meta-nodes in the network, which makes its study easier. Moreover, community detection can predict missing links and identify false links in the network. Despite its difficulty, a huge number of methods for community detection have been developed to deal with different size complexity and made available to the scientific community by open-source software packages. In this paper, we will address a specific question: are the detected communities significant, or are they a result of chance only due to the positions of edges in the network?

An important answer to this question is the Order Statistics Local Optimisation Method (*OSLOM*, `http://www.oslom.org/`) presented in Lancichinetti et al. (2011). *OSLOM* introduces an iterative technique based on the local optimization of a fitness function, the C-score (Lancichinetti et al., 2010), expressing the statistical significance of a cluster with respect to random fluctuations. The significance is evaluated by fixing a threshold parameter P a priori.

Another interesting approach is the Extraction of Statistically Significant Communities (*ESSC*, `https://github.com/jdwilson4/ESSC`) technique proposed in Wilson et al. (2014). The algorithm is iterative and identifies statistically stable communities measuring the significance of connections between a single vertex and a set of vertices in undirected networks under the configuration model (Bender and Canfield, 1978) used as the null hypothesis. The method employs multiple testing and false discovery rate control to update the candidate community.

Kojaku and Masuda (2018) introduced the QStest (`https://github.com/skojaku/qstest/`), a method to statistically test the significance of individual communities in a given network. Their algorithm works with different detection algorithms using a quality function that is consistent with the one used in community detection and takes into account the dependence of the quality function value on the community size. QStest assesses the statistical significance under the configuration model too.

Very recently, He et al. (2020) suggested the Detecting statistically Significant Communities (DSC) method, a significance-based community detection algorithm that uses a tight upper bound on the p-value under the configuration model coupled with an iterative local search method.

OSLOM, ESSC, and DSC assess the statistical significance of every single community analytically while QStest adopts the sampling method to calculate the p-value of a given community. Moreover, all of them detect statistically significant communities under the configuration model, and only QStest is independent of the detection algorithm.

We present **robin** (ROBustness In Network), an R/CRAN package whose purpose is to give clear indications about the reliability of one or more community detection algorithms understudy, analyzing their robustness with respect to random perturbations. The idea behind **robin** is that if a partition

is significant, it will be recovered even if the structure of the graph is modified. Alternatively, if the partition is not significant, minimal modifications of the graph will be sufficient to change it. **robin** is inspired by the concept presented by Carissimo et al. (2018), who studied the stability of the recovered partition against random perturbations of the original graph structure using tools from Functional Data Analysis (FDA).

robin provides the best choice among the variety of the existing methods for the network of interest. It is based on a procedure that gives the opportunity to use the community detection techniques implemented in the igraph package Csardi and Nepusz (2019) while providing the user with the possibility to include other community detection algorithms. **robin** initially detects if the community structure found by some algorithms is statistically significant, then it compares the different selected detection algorithms on the same network. **robin** assumes undirected graphs without loops and multiple edges.

robin looks at the global stability of the detected partition and not of single communities but accepts any detection algorithm and any random model, and these aspects differentiate it from OSLOM, ESSC, DSC, and QStest. Unlike other studies that treat the comparison between algorithms in a theoretical way, such as Yang et al. (2016), **robin** aims to give a practical answer to such a comparison that can vary with the network of interest.

The model

robin implements a methodology that examines the stability of the recovered partition by one or more algorithms. The methodology is useful for two purposes: to detect if the community structure found is statistically significant or is a result of chance; to choose the detection algorithm that better fits the network under study. These are implemented following two different workflows.

The ***first workflow*** tests the stability of the partitions found by a single community detection algorithm against random perturbations of the original graph structure. To address this issue, we specify a perturbation strategy (see subsection **Perturbation strategy**) and a null model to build some procedures based on a prefixed stability measure (see subsection **Stability measure**). Given:

- a network of interest $g1$
- its corresponding null random model $g2$
- a Detection Algorithm (DA)
- a stability measure (M)

Our process builds two curves as functions of the perturbation level p, as shown in Figure 1, and tests their similarity by two types of functional statistical tests (see subsection **Statistical tests**).

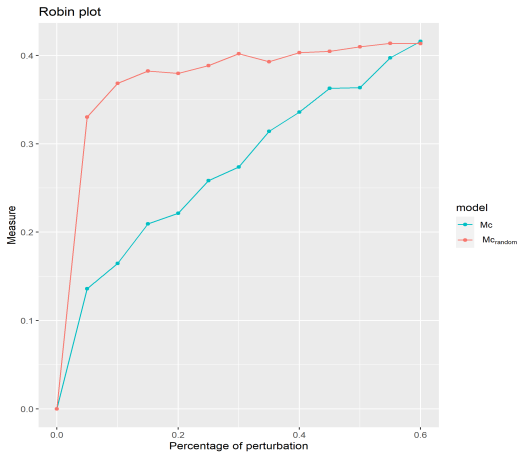


Figure 1: Example of Mc_{random} and Mc curves generated by an M stability measure

The first curve Mc is obtained by computing M between the partition of the original network $g1$ and the partition of a different perturbed version of $g1$. The second curve Mc_{random} is obtained by computing M between the partition of a null random network $g2$ and the partition of a different perturbed version of $g2$.

The comparison between the two M curves enables us to reconsider the problem regarding the significance of the retrieved community structure in the context of stability/robustness of the

recovered partition against perturbations. The basic idea is that if small changes in the network cause a completely different grouping of the data, the detected communities are not reliable. For a better understanding of this point, we refer the reader to the paper Carissimo et al. (2018) where the methodology was developed.

The choice of the null model plays a key role because we would expect it to reproduce the same structure of the real network but with completely random edges. For this reason, **robin** offers two possibilities: a degree preserving randomization by using the `rewire` function of the **igraph** package or a model chosen by the user.

The degree preserving randomization, i.e., Configuration Model (CM), is a model able to capture and preserve strongly heterogeneous degree distributions often encountered in real network data sets and is the standard null model for empirical patterns. Nevertheless, it can happen that it is not sufficient to preserve only the degree of the graph understudy, so **robin** allows the user to include their own null model.

In section **Example test: the American College football network**, we explore the *dk* null random model provided in Orsini et al. (2015), whose code is available at `https://github.com/polcolomer/RandNetGen` as a possible alternative to CM. The *dk*-series model generates a random graph preserving the global organization of the original network at various increasing levels of details chosen by the user via the setting of the parameter d. More precisely, the *dk*-series is a converging series of properties that characterize the local network structure at an increasing level of detail and define a corresponding series of null models or random graph ensembles. Increasing values of d capture progressively more properties of the network: *dk* 1 is equivalent to randomizing the network fixing only the degree sequence, *dk* 2 fixes additionally the degree correlations, *dk* 2.1 fixes also the clustering coefficient, and *dk* 2.5 the full clustering spectrum.

The first workflow is summarised as follows (see Figure 2):

1. find a partition C_1 for the real network and a partition C_2 for the null network,
2. perturb both networks,
3. retrieve two new partitions $C_{1(p)}$ and $C_{2(p)}$,
4. calculate two clustering distances (for the real network and the null network) between the original partitions and the ones obtained from the perturbed network as:

$$M\left(C_{1(p)}, C_1\right) \quad \text{and} \quad M\left(C_{2(p)}, C_2\right) \tag{1}$$

Steps 2) - 4) are computed at different perturbation levels $p \in [0 : 0.05 : 0.6]$ to create two curves, one for the real network and one for the null model, then their similarity is tested by two functional statistical tests described in subsection **Statistical tests**.

This procedure allows the filtering of the detection algorithms according to their performance. Moreover, the selected ones can be compared using the second workflow.

The ***second workflow*** helps to choose among different community detection algorithms the one that best fits the network of interest, comparing their robustness two at a time. More precisely, the technique (see Figure 3):

1. find two partitions C_1 and C_2 inferred by two different algorithms on the network under study,
2. perturb the network creating a new one,
3. retrieve two new partitions $C_{1(p)}$ and $C_{2(p)}$,
4. evaluate $M\left(C_{1(p)}, C_1\right)$ and $M\left(C_{2(p)}, C_2\right)$.

Steps 2) - 4) are repeated at different perturbation levels $p \in [0 : 0.05 : 0.6]$ to create two curves and then their similarity is tested.

Perturbation strategy

The perturbed network has been restricted to have the same number of vertices and edges as the original unperturbed network. Therefore, only the positions of the edges are changed. It is expected that if a community structure is robust, it should be stable under small perturbations of the edges. This is because perturbing the network edges by a small amount will imply just a small percentage of nodes to be moved in different communities; on the other hand, perturbing a high percentage of the edges in the network will produce random clusters. Note that zero perturbation $p = 0$ corresponds to the original graph while a maximal perturbation level $p = 1$ will correspond to the random graph.

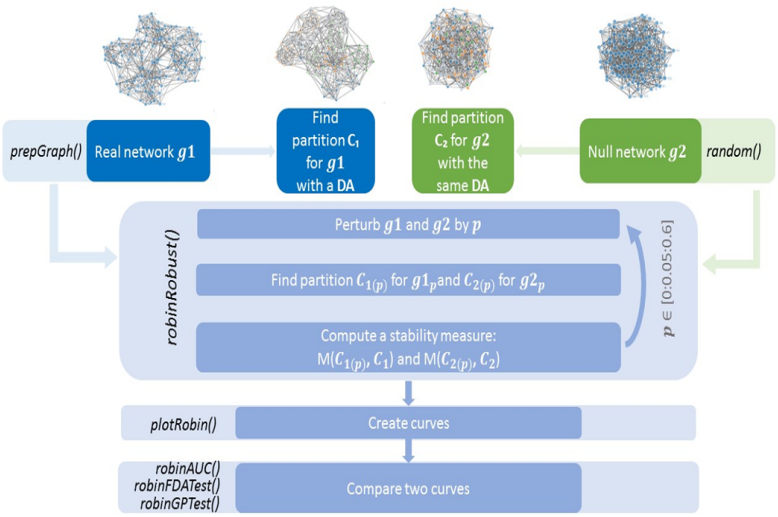


Figure 2: Workflow to test the goodness of a community detection algorithm.

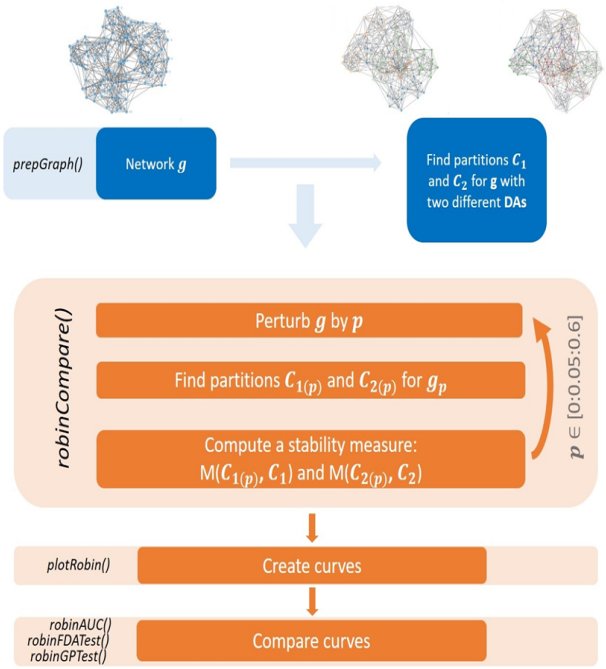


Figure 3: Workflow to compare two different community detection algorithms.

Therefore, in **robin**, the perturbation of a network preserves the degree distribution of the original network.

Two different procedures for the perturbation strategy are implemented, namely independent and dependent types. The independent strategy introduces a percentage p of perturbation in the original graph at each iteration, for $p = 0, \ldots, p_{max}$. Whereas the dependent procedure introduces 5% of perturbation at each iteration on the previous perturbed graph, starting from the original network, until p_{max} of the graph's edges are perturbed. In the implementation of the perturbation strategy, we set up $p_{max} = 0.6$, because the structure of the network becomes random if we perturb more than 50% of the edges.

In particular, we noticed that the greatest modification of the network structure happens for

a perturbation level between 30% and 40% if a network is robust, while it happens at very low perturbation levels if the network is not robust.

We stress again that the M curve for a network with a strong structure grows rapidly (perturbation level between 0% and 40%) then levels off when $50\% < p < 100\%$.

Moreover, the choice $p_{max} = 0.6$ reduces computational time and shows more clearly the differences between the curves.

Varying the percentage of perturbation, many graphs are generated and compared by means of the stability measure chosen. For each perturbation level, we generated 10 perturbed graphs and calculated the stability measure. From each of these graphs, we generated 9 more by rewiring an additional 1% of the edges. Therefore, the procedure generates 100 graphs with the respective stability measures for each level of p and gives as output the mean of the stability measure for every 10 graphs generated.

Stability measure

The procedure we implemented is based on four different stability measures:

- the Variation of Information (VI) proposed by Meilă (2007),
- the Normalized Mutual Information (NMI) measure proposed by Danon et al. (2005),
- the split-join distance of van Dongen (2000),
- the Adjusted Rand Index (ARI) by Hubert and Arabie (1985).

VI measures the amount of information lost and gained in changing from one cluster to another, while split-join distance calculates the number of nodes that have to be exchanged to transform any of the two clusterings into the other; but for both of them, low values represent more similar clusters, and high values represent more different clusters. On the contrary, NMI and ARI are similarity measures, and therefore, lower values identify more different clusters and higher values more similar ones. To make all the measures comparable, we considered the 1-1 transformation for the NMI and the ARI since they vary between $[0, 1]$ as:

$$f(X) = 1 - X \tag{2}$$

Only two of the four proposed stability measures, i.e., split-join and VI, are distances. They differ in their dependency on the number of clusters K: while the VI distance grows logarithmically with K, the split-join metric grows with K toward the upper bound of 1. To make the four different stability measures comparable, we normalized VI and split-join between 0 and 1 (i.e., we divided the VI and the split-join by their maximum, respectively $\log(n)$ and $2n$, where n indicates the number of vertices in the graph).

Statistical tests

robin allows different multiple statistical tests to check the differences between the real and the random curve or between the curves built from two different detection algorithms. The variation of p from 0 to 0.6 induces an intrinsic order to the data structure as in temporal data. This lets p assume, the same role as a time point in a temporal process, and as a consequence, we can use any suitable time series approach to compare our curves. In the following, we describe the use of two such approaches.

The first is a test based on the Gaussian Process regression (GP) described in Kalaitzis and Lawrence (2011b). In this paper, the authors use GP to compare treatment and control profiles in biological time-course experiments. The main idea is to test if two time series represent the same or two different temporal processes. A Gaussian process is a collection of random variables, any finite number of which have a joint Gaussian distribution and is completely specified by its mean function and its covariance function, see e.g., Rasmussen and Williams (2006). Given the mean function $m(x)$ and the covariance function $k(x, x')$ of a real process $f(x)$, we can write the GP as:

$$f(x) \sim \mathcal{GP}\left(m(x), k(x, x')\right). \tag{3}$$

The random variables $\mathbf{f} = (f(X_1), \ldots, f(X_n))^T$ represent the value of the function $f(x)$ at time locations $(X_i)_{i=1,\ldots,n}$, being $f(x)$ the true trajectory/profile of the gene. Assuming $f(x) = \Phi(x)^T \mathbf{w}$, where $\Phi(x)$ are projection basis functions, with prior $\mathbf{w} \sim N(\mathbf{0}, \sigma_{\mathbf{w}}^2 \mathbf{I})$, we have

$$m(x) = \Phi(x)^T E[\mathbf{w}] = 0, \quad k(x, x') = \sigma_{\mathbf{w}}^2 \Phi(x)^T \Phi(x) \tag{4}$$

$$f(x) \sim \mathcal{GP}\left(0, \sigma_{\mathbf{w}}^2 \Phi(x)^T \Phi(x)\right). \tag{5}$$

Since observations are noisy, i.e., $\mathbf{y} = \mathbf{\Phi w} + \boldsymbol{\varepsilon}$, with $\mathbf{\Phi} = (\Phi(X_1)^T, \ldots, \Phi(X_n)^T)$, assuming that the noise $\boldsymbol{\varepsilon} \sim N(\mathbf{0}, \sigma_n^2 \mathbf{I})$ and using Eq. (4), the marginal likelihood becomes:

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{(2\pi)^{n/2} \left|\mathbf{K_y}\right|^{1/2}} \exp\left(-\frac{1}{2}\mathbf{y}^t \mathbf{K_y}^{-1} \mathbf{y}\right), \tag{6}$$

with $\mathbf{K_y} = \sigma_{\mathbf{w}}^2 \mathbf{\Phi\Phi}^T + \sigma_n^2 \mathbf{I}$.

In this framework, the hypothesis testing problem over the perturbation interval $[0, p_{max}]$ can be reformulated as:

$$H_0 : \log_2 \frac{M_1(x)}{M_2(x)} \sim GP\left(0, k\left(x, x'\right)\right) \quad \text{against} \quad H_1 : \log_2 \frac{M_1(x)}{M_2(x)} \sim GP\left(m(x), k\left(x, x'\right)\right), \tag{7}$$

where x represents the perturbation level. To compare the two curves, **robin** uses the Bayes Factor (BF), which is approximated with a log-ratio of marginal likelihoods of two GPs, each one representing the hypothesis of differential (the profile has a significant underlying signal) and non-differential expression (there is no underlying signal in the profile, just random noise).

The second test implemented is based on the Interval Testing Procedure (ITP) described in Pini and Vantini (2016). The ITP provides an interval-wise nonparametric functional testing and is not only able to assess the equality in distribution between functions, but also to underline specific differences. Indeed, users can see where are localized the differences between the two curves. The Interval Testing Procedure is based on:

1. Basis Expansion: functional data are projected on a functional basis (i.e. Fourier or B-splines expansion);
2. Interval-Wise Testing: statistical tests are performed on each interval of basis coefficients;
3. Multiple Correction: for each component of the basis expansion, an adjusted ***p***-value is computed from the ***p***-values of the tests performed in the previous step.

In summary, GP provides a global answer on the dissimilarity of the two M curves, while ITP points out local changes between such curves. As a rule of thumb, we suggest initially using GP to flag a difference and then ITP to understand at which level of perturbation such a difference is locally significant.

We also provide a global method to quantify the differences between the curves when they are very close. This is based on the calculation of the area under the curves with a spline approach.

Package structure

Installation

Once in the R environment, it is possible to install and load the **robin** package with its dependencies, as follows:

```
install.packages("robin")
```

The **robin** package includes as dependencies **igraph** (Csardi and Nepusz, 2019), **networkD3** (Gandrud et al., 2017), **ggplot2** (Wickham, 2019), **gridExtra** (Auguie, 2017), **fdatest** (Pini and Vantini, 2015) , **gprege** (Kalaitzis and Lawrence, 2011a), and **DescTools** (Signorell and mult. al., 2019) packages. All, except **gprege** which is a Bioconductor package, are automatically loaded with the command:

```
library(robin).
```

To install the **gprege** package, start R and enter:

```
if (!requireNamespace("BiocManager", quietly = TRUE))
      install.packages("BiocManager")
BiocManager::install("gprege")
```

Data import and visualization

robin is a user-friendly software providing some additional functions for data import and visualization, such as `prepGraph`, `plotGraph`, and `plotComm`. The function `prepGraph`, required by the procedure, reads, and simplifies undirected graphs removing loops and multiple edges. The available input

graphs formats are: "edgelist", "pajek", "ncol", "lgl", "graphml", "dimacs", "graphdb", "gml", "dl", and an igraph object. The function `plotGraph`, with the aid of the **network3D** package, starting from an igraph object loaded with `prepGraph`, shows an interactive 3D graphical representation of the network, useful to visualize the network of interest before the analysis. Furthermore, the function `plotComm` helps to plot a graph with colorful nodes that simplifies the visualization of the detected communities, given the membership of the communities.

Procedures

robin embeds all the community detection algorithms present in **igraph**. They can be classified as in (Fortunato, 2009)

modularity based methods:

- `cluster_fast_greedy` (Clauset et al., 2005)
- `cluster_leading_eigen` (Newman, 2006)
- `cluster_louvain` (Blondel et al., 2008)

divisive algorithms:

- `cluster_edge_betweenness` (Newman and Girvan, 2004)

methods based on statistical inference:

- `cluster_infomap` (Rosvall and Bergstrom, 2008)

dynamic algorithms:

- `cluster_spinglass` (Reichardt and Bornholdt, 2006)
- `cluster_walktrap` (Pons and Latapy, 2005)

alternative methods:

- `cluster_label_prop` (Raghavan et al., 2007).

robin gives the possibility to input a custom external function to detect the communities. The user can provide his own function as a value of the parameter `FUN` in both analyses, implemented into the functions `robinRobust` and `robinCompare`. These two functions create the internal process for perturbation and measurement of communities stability. In particular `robinRobust` tests the robustness of a chosen detection algorithm and `robinCompare` compares two different detection algorithms. The option `measure` in the `robinRobust` and `robinCompare` functions provides the flexibility to choose between the four different measures listed in the subsection **Stability measure**.

robin offers two choices for the null model to set up for `robinRobust`:

- external building according to users' preferences, then the null graph is passed as a variable,
- generation by using the function `random`.

The function `random` creates a random graph with the same degree distribution as the original graph, but with completely random edges, by using the `rewire` function of the **igraph** package with the `keeping_degseq` option that preserves the degree distribution of the original network. The function `rewire` randomly assigns a number of edges between vertices with the given degree distribution. Note that **robin** assumes undirected graphs without loops and multiple edges which are directly created, from any input graph, by the function `prepGraph`.

Construction of curves

The `plotRobin` function allows the user to generate two curves based on the computation of the chosen stability measures.

When `plotRobin` is used on the output of `robinRobust`, i.e., the first step of the overall procedure, the first curve represents the measure between the partition of the original unperturbed graph and the partition of each perturbed graph (blue curve in Figure 4-Left panel), and the second curve is obtained in the same way but considering as the original graph the random graph (red curve in Figure 4-Left panel). The comparison between the two curves turns the question about the significance of the retrieved community structure into the study of the robustness of the recovered partition against perturbation.

When `plotRobin` is used on the output of `robinCompare`, i.e. the second step of the overall procedure, it generates a plot that depicts two curves, one for each clustering algorithm. In the right panel of Figure 4, each curve is obtained by computing the measure between the partition of the original unperturbed graph with the partition of each perturbed graph, where the partition method is either Louvain (blue curve) or Fast Greedy (red curve).

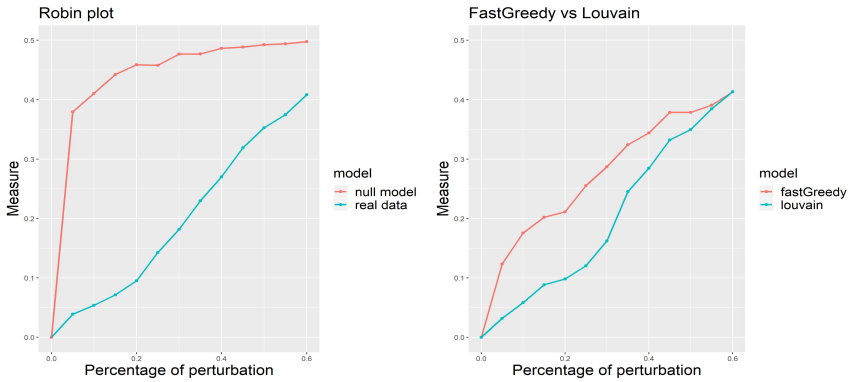


Figure 4: Curves of the null model and the real data generated by the VI stability measure and the Louvain detection algorithm on the American College Football network (Left panel). Curves of the Louvain and Fast greedy algorithm generated by the VI stability measure on the American College Football network (Right panel) (Girvan and Newman, 2002).

Testing

The GP test is implemented in `robinGPTest` and uses the R package **gprege** (Kalaitzis and Lawrence, 2011a). The ITP test is implemented in `robinFDATest` and uses the R package **fdatest** (Pini and Vantini, 2015). The area under the curves is calculated by the function `robinAUC` and relies on the **DescTools** package. Figure 5 shows the curves for the comparison of Louvain and Fast greedy algorithms' performance generated by the VI stability measure using the Interval Testing Procedure on the American College Football network (left panel) (Girvan and Newman, 2002) and corresponding adjusted p-values (right panel).

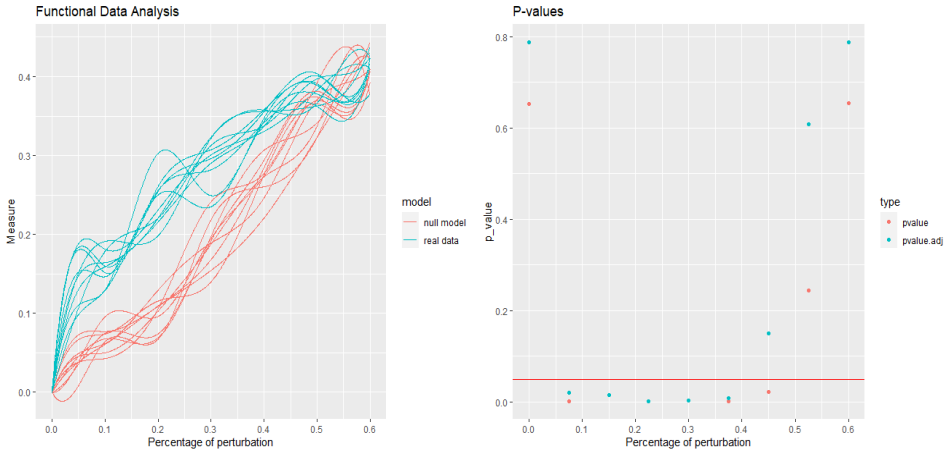


Figure 5: Curves of the Louvain and Fast greedy algorithm generated by the VI stability measure using the Interval Testing Procedure on the American College Football network (Left panel). Corresponding p-values and adjusted p-values for all the intervals with the horizontal red line on to the critical value 0.05 (Right panel).

All the functions implemented in **robin** are summarized in Table 1.

Computational time

The time complexity of the proposed strategy, more precisely of the `robinRobust` function, is evaluated as follows. Generating a rewired network with N nodes and M edges consumes $O(N + M)$ time, for

Table 1: Summary of the functions implemented in **robin**.

FUNCTION	DESCRIPTION
Import/Manipulation	
`prepGraph`	Management and preprocessing of input graph
`random`	Building of null model
Analysis	
`robinRobust`	Comparison of a community detection method versus random perturbations of the original graph
`robinCompare`	Comparison of two different community detection methods
Visualization	
`methodCommunity`	Detection of the community structure
`membershipCommunities`	Detection of the membership vector of the community structure
`plotGraph`	Graphical interactive representation of the network
`plotComm`	Graphical interactive representation of the network and its communities
`plotRobin`	Plots of the two curves
Test	
`robinGPTest`	GP test and evaluation of the Bayes factor
`robinFDATest`	ITP test and evaluation of the adjusted p-values
`robinAUC`	Evaluation of the area under the curve

both the real and the null model. For each network, we detect the communities, using any community detection algorithm present in **igraph** or any custom external algorithm inserted by the user, and calculate a stability measure. Let D be the time complexity associated with the community detection algorithm chosen. The overall procedure is iterated $k = 100$ times for each percentage p of the $n_p = 12$ perturbation levels ($p \in [0, p_{max}]$, $p_{max} = 0.6$). In total, the proposed procedure requires $O(D + (((N + M + D) * k) * n_p))$ time both for the real and the null model.

In Table 2, we show the computational time evaluated on a computer with an Intel 4 GHz i7-4790K processor and 24GB of memory. In this example, we used *Louvain* as a detection algorithm on the *LFR* benchmark graphs (Lancichinetti et al., 2008). The time complexity could be mitigated using parallel computing, but this is not yet implemented.

Example test: the American College football network

robin includes the *American College football* benchmark dataset as an analysis example that is freely available at `http://www-personal.umich.edu/~mejn/netdata/`. The dataset contains the network of United States football games between Division I colleges during the 2000 season (Girvan and Newman, 2002). It is a network of 115 vertices that represent teams (identified by their college names) and 613

Table 2: Computational time

NODES	EDGES	TIME(SECS)
100	500	2.1
1000	9267	36.1
10000	100024	361.8
100000	994053	9411.6
1000000	8105913	110981.5

edges that represent regular-season games between the two teams they connect. The graph has the ground truth, where each node has a value that indicates to which of the 12 conferences it belongs, and this offers a good opportunity to test the ability of **robin** to validate the community robustness. It is known that each conference contains around 8-12 teams. The games are more frequent between members of the same conference than between members of different conferences. They are on average seven between teams of the same conference and four between different ones. We applied all the methods listed in subsection **Procedures** to this network, choosing as measure the VI metric.

```
library(robin)

my_network <- system.file("example/football.gml", package="robin")
graph <- prepGraph(file=my_network, file.format="gml")

attributes <- vertex_attr(graph, index = V(graph))
real <- attributes$value
real <- as_membership(real)

set.seed(10)
members_In <- membershipCommunities(graph=graph, method=DA)
VI_In <- compare(real, members_In, method="vi")
```

Note that the variable *DA* refers to the detection algorithms present in **igraph** and can assume the following values: *fastGreedy*, *infomap*, *walktrap*, *edgeBetweenness*, *spinglass*, *leadingEigen*, *labelProp*, *louvain*. The function `compare` is contained in the package **igraph** and permits the assessment of the distance between two community structures according to the chosen method.

Table 3 summarizes the VI results calculated between the real communities and the ones that the detection algorithms created.

Table 3: VI measure between different methods and ground-truth.

METHODS	NORMALIZED VI
cluster_infomap	0.054
cluster_spinglass	0.063
cluster_louvain	0.076
cluster_label_prop	0.076
cluster_walktrap	0.078
cluster_edge_betweenness	0.083
cluster_fast_greedy	0.185
cluster_leading_eigen	0.196

It is possible to observe that the best performance is offered by Infomap, having the lowest VI value, followed by Spinglass. Louvain, Propagating Labels, Walktrap, and Edge betweenness have a similar intermediate VI value, while the worst performance is given by Fast greedy and Leading eigenvector. Then, we used **robin** to check if the results are confirmed by looking at the VI curves and the results of the testing procedure for the second workflow, i.e., the one comparing two detection algorithms, considering Infomap versus all the others.

```
comp <- robinCompare(graph=graph, method1=DA1,
                     method2=DA2, measure="vi", type="independent")

plotRobin(graph=graph, model1=comp$Mean1, model2=comp$Mean2,
```

```
measure="vi")
```

Figure 6 shows the results we obtained. If we focus on the perturbation interval $[0, 0.3]$, it is possible to note the similar behavior between the curves representing Infomap/Spinglass, Infomap/Louvain, Infomap/Propagating Labels, Infomap/Walktrap, and Infomap/Edge betweenness, with a closer distance between Infomap/Spinglass. On the contrary, the curves Infomap/Fast greedy and Infomap/Leading eigenvector have an opposite behavior, building almost an ellipse. This confirms what is displayed in Table 3.

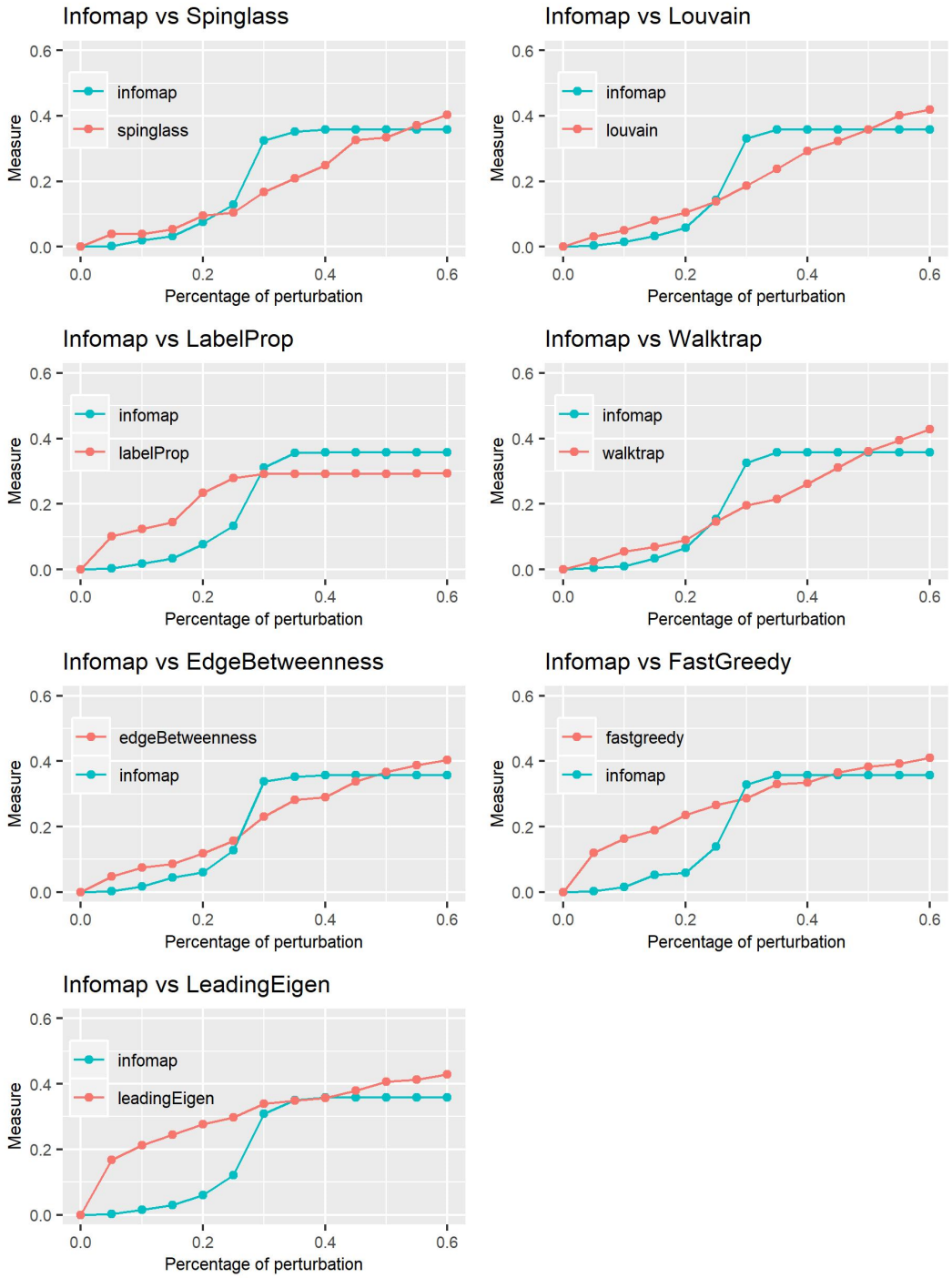


Figure 6: Plot of the VI curves of Infomap against all other methods.

In our overall procedure, we explored two different ways of generating a null model, namely the

Configuration Model (CM) and the *dk*-series model.

```
graphRandomCM <- random(graph=graph)
graphRandomDK <- prepGraph(file="dk2.1_footballEdgelist.txt",
                           file.format = "edgelist")

plotGraph(graph)
plotGraph(graphRandomCM)
plotGraph(graphRandomDK)
```

The different structures provided by the real data network, CM, and *dk*-series with $d = 2.1$ are shown in Figure 7.

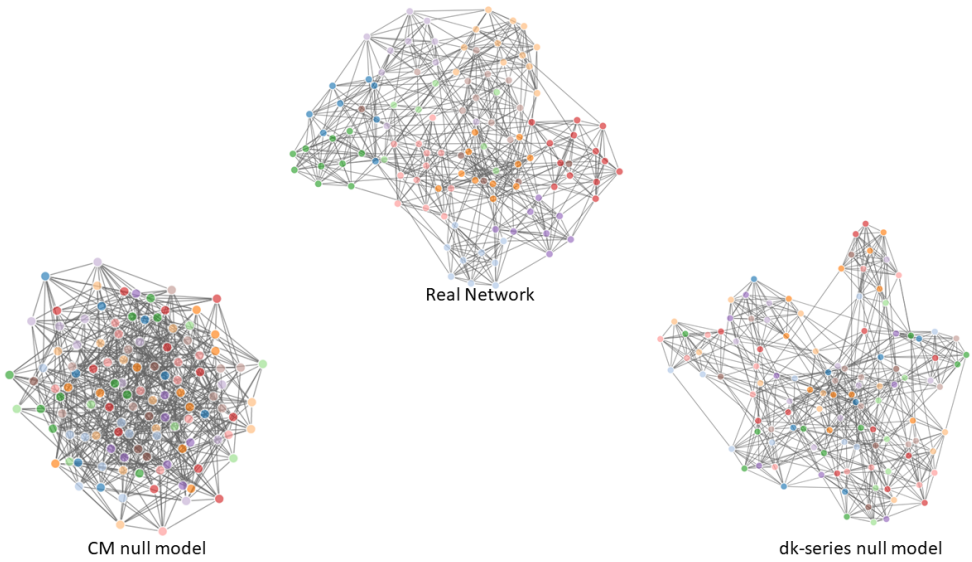


Figure 7: Graph of the real data (Upper panel); graph of the CM null model (Left - lower panel); graph of the *dk*-series null model with $d = 2.1$ (Right - lower panel).

The CM generates a random graph with the same degree sequence as the original one but with a randomized group structure. Our experiments show that CM is not a good null model when using Propagating Labels and Infomap as community extraction methods (Figure 8). In fact, when the modularity is low, these two algorithms tend to assign all the nodes to the same community, hence resulting in a flat stability measure curve. We launched the function `robinRobust` to assess the robustness of each detection algorithm.

```
proc_CM <- robinRobust(graph=graph, graphRandom=graphRandomCM,
                       measure="vi", method=DA, type="independent")

plotRobin(graph=graph, model1=proc_CM$Mean,model2=proc_CM$MeanRandom,
          measure="vi")
```

The *dk*-series model generates a random graph preserving the global organization of the original network at various increasing levels of details chosen by the user via the setting of the parameter d. In particular, we chose the *dk* random graph with d=2.1.

```
proc_DK <- robinRobust(graph=graph, graphRandom=graphRandomDK,
                       measure="vi", method="fastGreedy", type="independent")

plotRobin(graph=graph, model1=proc_DK$Mean, model2=proc_DK$MeanRandom,
          measure="vi")
```

Figure 9 shows the stability measure curves of each detection algorithm compared to dk 2.1 null model. For all the methods, the two curves are very close due to the capability of the null model to preserve a structure similar to the real network and visually confirm the results in Table 3.

Moreover, for the *dk*-series model, we tested the differences between the two curves using the GP methodology implemented in the function in `robinGPTest`.

```
BFdk <- robinGPTest(model1=proc_DK$Mean, model2=proc_DK$MeanRandom)
```

The results are shown in Table 4 and agree with those shown in Table 3.

Table 4: Bayes Factor and AUC ratio for *dk* -series with $d = 2.1$

METHODS	BAYES FACTOR	AUC
`cluster_infomap`	53.67	1.133
`cluster_spinglass`	40.55	1.249
`cluster_louvain`	31.52	1.208
`cluster_label_prop`	102.2	0.852
`cluster_walktrap`	30.74	1.204
`cluster_edge_betweenness`	31.10	1.194
`cluster_fast_greedy`	0.001	1.017
`cluster_leading_eigen`	8.474	1.042

Fastgreedy clearly fails in recovering the communities. LeadingEigen has stronger evidence but too weak when compared to the other methods. Louvain, Walktrap, and EdgeBetweenness have the same strong evidence followed by Spinglass and Infomap. LabelProp shows the strongest evidence, but the result is obviously influenced by the swap between the two curves when the perturbation is greater than 20%, underlying a worse performance of the algorithm. The same swap can be noted for Infomap at 35% perturbation level, but with less difference between the two curves. This is confirmed by the fact that the ratios between the AUC of the real null model curve and the AUC of the real network are close to 1.

```
AUC <- robinAUC(graph=graph, model1=proc_DK$Mean,
                model2=proc_DK$MeanRandom, measure="vi")
AUCdkratio <- AUC$area2/AUC$area1
```

Also, note that LabelProp originates the paradox that the AUC of the real model curve exceeds the AUC of the null network, despite the hypothesis testing result is positive. Hence, it is always the case to look at the plots and AUC ratios.

Conclusion

In this paper, we presented **robin**, an R/CRAN package, to assess the robustness of the community structure of a network found by one or more detection methods, providing an indication of their reliability. The procedure implemented is useful for comparing different community detection algorithms and choosing the one that best fits the network of interest. More precisely, **robin** initially detects if the community structure found by some algorithms is statistically significant, then it compares the different selected detection algorithms on the same network. **robin** uses analysis tools set up for functional data analysis, such as *GP* regression and *ITP*. The core functions of the package are `robinRobust` and `robinCompare`, which build the stability measure curves for the null model and the network understudy for a fixed detection algorithm and the stability measure for the network understudy for two detection algorithms, respectively. Moreover, `robinGPTest` and `robinFDATest` implement the GP test and the ITP test. We illustrated the usage of the package on a benchmark dataset. The package is available on CRAN at https://CRAN.R-project.org/package=robin.

Computational details

The results in this paper were obtained using R 3.6.1 with the packages **igraph** version 1.2.4.2, **networkD3** version 0.4, **ggplot2** version 3.2.1, **gridExtra** version 2.3, **fdatest** version 2.1, **gprege** version 1.30.0, and **DescTools** version 0.99.31. R itself and all packages used are available from the Comprehensive R Archive Network (CRAN) at https://CRAN.R-project.org/.

Acknowledgements

This work was supported by the project Piattaforma Tecnologica ADVISE - Regione Campania, by the project "TAILOR" (H2020-ICT-48 GA: 952215) and by DiSTABiF at the University of Campania Luigi Vanvitelli that is coordinating V.P. PhD program.

Contributions

V.P. implemented the software and analyzed its properties. D.R. supported V.P. in R package implementation. A.C., L.C., and I.D.F. conceived the work, equally contributed to the development and implementation of the concept, discussed and analyzed the results. A.C., L.C., I.D.F., and V.P. wrote the manuscript.

Bibliography

B. Auguie. **gridExtra**: *Miscellaneous Functions for "Grid" Graphics*, 2017. URL https://CRAN.R-project.org/package=gridExtra. R package version 2.3. [p297]

E. A. Bender and E. R. Canfield. The asymptotic number of labeled graphs with given degree sequences. *Journal of Combinatorial Theory A*, 24:296–307, 1978. [p292]

V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, 2008. doi: 10.1088/1742-5468/2008/10/p10008. [p298]

A. Carissimo, L. Cutillo, and I. De Feis. Validation of community robustness. *Computational Statistics and Data Analysis*, 120:1–24, 2018. doi: 10.1016/j.csda.2017.10.006. [p293, 294]

A. Clauset, M. Newman, and C. Moore. Finding community structure in very large networks. *Physical review. E, Statistical, nonlinear, and soft matter physics*, 70:066111, 2005. doi: 10.1103/PhysRevE.70.066111. [p298]

G. Csardi and T. Nepusz. **igraph**: *Network Analysis and Visualization*, 2019. URL https://CRAN.R-project.org/package=igraph. R package version 1.2.4.2. [p293, 297]

L. Danon, A. Díaz-Guilera, J. Duch, and A. Arenas. Comparing community structure identification. *Journal of Statistical Mechanics: Theory and Experiment*, 9:219–228, 2005. doi: 10.1088/1742-5468/2005/09/P09008. [p296]

S. Fortunato. Community detection in graphs. *Physics reports*, 486(3–5):75–174, 2009. [p298]

C. Gandrud, J. J. Allaire, K. Russell, and C. J. Yetman. **networkD3**: *D3 JavaScript Network Graphs from R*, 2017. URL https://CRAN.R-project.org/package=networkD3. R package version 0.4. [p297]

M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences of the United States of America*, 99(12):7821–7826, 2002. doi: 10.1073/pnas.122653799. [p292, 299, 300]

Z. He, H. Liang, Z. Chen, C. Zhao, and Y. Liu. Detecting statistically significant communities. *IEEE Transactions on Knowledge and Data Engineering,*, 2020. [p292]

L. Hubert and P. Arabie. Comparing partitions. *Journal of Classification*, 2(1):193–218, 1985. doi: 10.1007/BF01908075. [p296]

A. A. Kalaitzis and N. D. Lawrence. **gprege**: *Gaussian Process Ranking and Estimation of Gene Expression time-series*, 2011a. URL https://www.bioconductor.org/packages/release/bioc/html/gprege.html. R package version 1.30.0; Bioconductor version 3.10. [p297, 299]

A. A. Kalaitzis and N. D. Lawrence. A simple approach to ranking differentially expressed gene expression time courses through gaussian process regression. *BMC Bioinformatics*, 12(180), 2011b. doi: 10.1186/1471-2105-12-180. [p296]

S. Kojaku and N. Masuda. A generalised significance test for individual communities in networks. *Scientific Reports*, 8:7351, 2018. [p292]

A. Lancichinetti, S. Fortunato, and F. Radicchi. Benchmark graphs for testing community detection algorithms. *Physical Review E*, 78(4), Oct 2008. ISSN 1550-2376. doi: 10.1103/physreve.78.046110. URL http://dx.doi.org/10.1103/PhysRevE.78.046110. [p300]

A. Lancichinetti, F. Radicchi, and J. J. Ramasco. Statistical significance of communities in networks. *Phys. Rev. E*, 81:046110, 2010. [p292]

A. Lancichinetti, F. Radicchi, J. J. Ramasco, and S. F. S. Finding statistically significant communities in networks. *PloS One*, 6(4):e18961, 2011. [p292]

Meilă. Comparing clusterings - an information based distance. *Journal of Multivariate Analysis*, 98(5): 873–895, 2007. doi: 10.1016/j.jmva.2006.11.013. [p296]

M. E. J. Newman. Finding community structure in networks using the eigenvectors of matrices. *Phys. Rev. E*, 74(3):036104, 2006. doi: 10.1103/PhysRevE.74.036104. [p298]

M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Phys. Rev. E*, 69(2):026113, 2004. doi: 10.1103/PhysRevE.69.026113. [p298]

C. Orsini, M. Dankulov, P. C. de Simón, and et al. Quantifying randomness in real networks. *Nature Communications*, 6(8627), 2015. doi: 10.1038/ncomms9627. [p294]

A. Pini and S. Vantini. **fdatest***: Interval Testing Procedure for Functional Data*, 2015. URL https://CRAN.R-project.org/package=fdatest. R package version 2.1. [p297, 299]

A. Pini and S. Vantini. The interval testing procedure: A general framework for inference in functional data analysis. *Biometrics*, 72(3):835–845, 2016. doi: 10.1111/biom.12476. [p297]

P. Pons and M. Latapy. Computing communities in large networks using random walks. In *Computer and Information Sciences - ISCIS 2005*, pages 284–293. Springer Berlin Heidelberg, 2005. [p298]

U. N. Raghavan, R. Albert, and S. Kumara. Near linear time algorithm to detect community structures in large-scale networks. *Phys. Rev. E*, 76:036106, 2007. doi: 10.1103/PhysRevE.76.036106. [p298]

C. E. Rasmussen and C. K. I. Williams. *Gaussian Process for Machine Learning*. The MIT Press, 2006. [p296]

J. Reichardt and S. Bornholdt. Statistical mechanics of community detection. *Phys. Rev. E*, 74:016110, 2006. doi: 10.1103/PhysRevE.74.016110. [p298]

M. Rosvall and C. T. Bergstrom. Maps of random walks on complex networks reveal community structure. *Proceedings of the National Academy of Sciences*, 105(4):1118–1123, 2008. doi: 10.1073/pnas.0706851105. [p298]

A. Signorell and mult. al. **DescTools***: Tools for Descriptive Statistics*, 2019. URL https://CRAN.R-project.org/package=DescTools. R package version 0.99.31. [p297]

S. van Dongen. Performance criteria for graph clustering and markov cluster experiments. Technical report, National Research Institute for Mathematics and Computer Science in the Netherlands, P. O. Box 94079 NL-1090 GB Amsterdam, Netherlands, 2000. [p296]

H. Wickham. **ggplot2***: Create Elegant Data Visualisations Using the Grammar of Graphics*, 2019. URL https://CRAN.R-project.org/package=ggplo2. R package version 3.2.1. [p297]

J. Wilson, S. Wang, P. Mucha, S. Bhamidi, and A. Nobel. A testing based extraction algorithm for identifying significant communities in networks. *The Annals of Applied Statistics*, 8(3):1853–1891, 2014. [p292]

Z. Yang, R. Algesheimer, and C. J. Tessone. A comparative analysis of community detection algorithms on artificial networks. *Scientific Reports*, 6(30750), 2016. doi: 10.1038/srep30750. [p293]

Valeria Policastro (package author and creator)
Dipartimento Scienze e Tecnologie Ambientali, Biologiche e Farmaceutiche
Universitá degli Studi della Campania "Luigi Vanvitelli"
Via Vivaldi, 43 81100 Caserta, Italia
and
Istituto per le Applicazioni del Calcolo "M. Picone" - sede di Napoli
Consiglio Nazionale delle Ricerche
via Pietro Castellino 111

80131 Napoli, Italy
valeria.policastro@unicampania.it

Dario Righelli
Dipartimento di Statistica
Universitá di Padova
Via Cesare Battisti, 241 35121 Padova, Italia
and
Istituto per le Applicazioni del Calcolo "M. Picone" - sede di Napoli
Consiglio Nazionale delle Ricerche
via Pietro Castellino 111
80131 Napoli, Italy
d.righelli@na.iac.cnr.it

Annamaria Carissimo
Istituto per le Applicazioni del Calcolo "M. Picone" - sede di Napoli
Consiglio Nazionale delle Ricerche
via Pietro Castellino 111
80131 Napoli, Italy
Co-last and corresponding author
a.carissimo@iac.cnr.it

Luisa Cutillo
School of Mathematics
University of Leeds
Leeds
LS2 9JT, United Kingdom
and
Dipartimento di Studi Aziendali e Quantitativi
Universitá degli Studi di Napoli "Parthenope'
Via Generale Parisi, 13
80132, Napoli, Italia
Co-last and corresponding author
l.cutillo@leeds.ac.uk

Italia De Feis
Istituto per le Applicazioni del Calcolo "M. Picone" - sede di Napoli
Consiglio Nazionale delle Ricerche
via Pietro Castellino 111
80131 Napoli, Italy
Co-last and corresponding author
i.defeis@iac.cnr.it

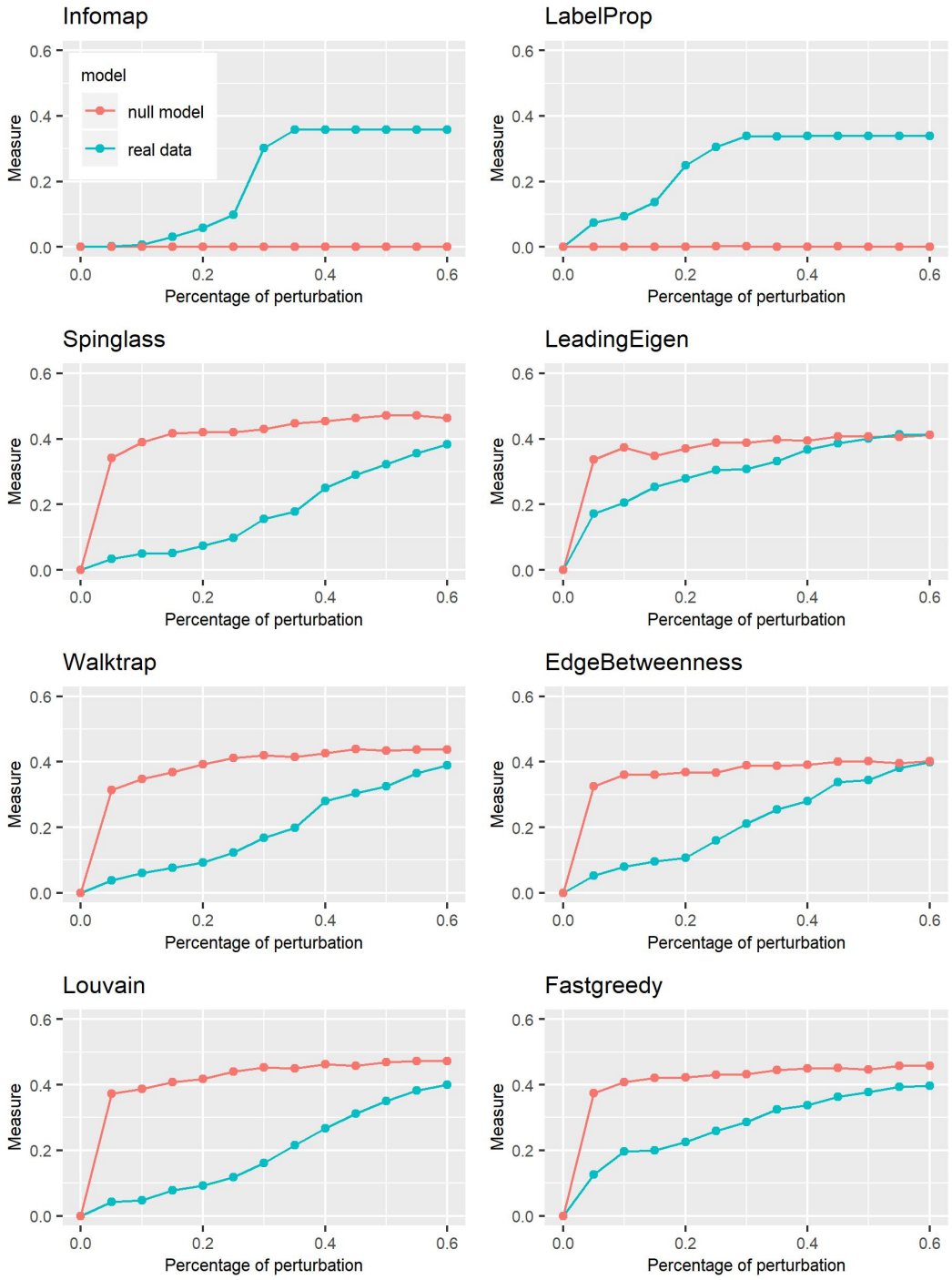


Figure 8: Plot of the VI curves of the CM null model and all the algorithms implemented.

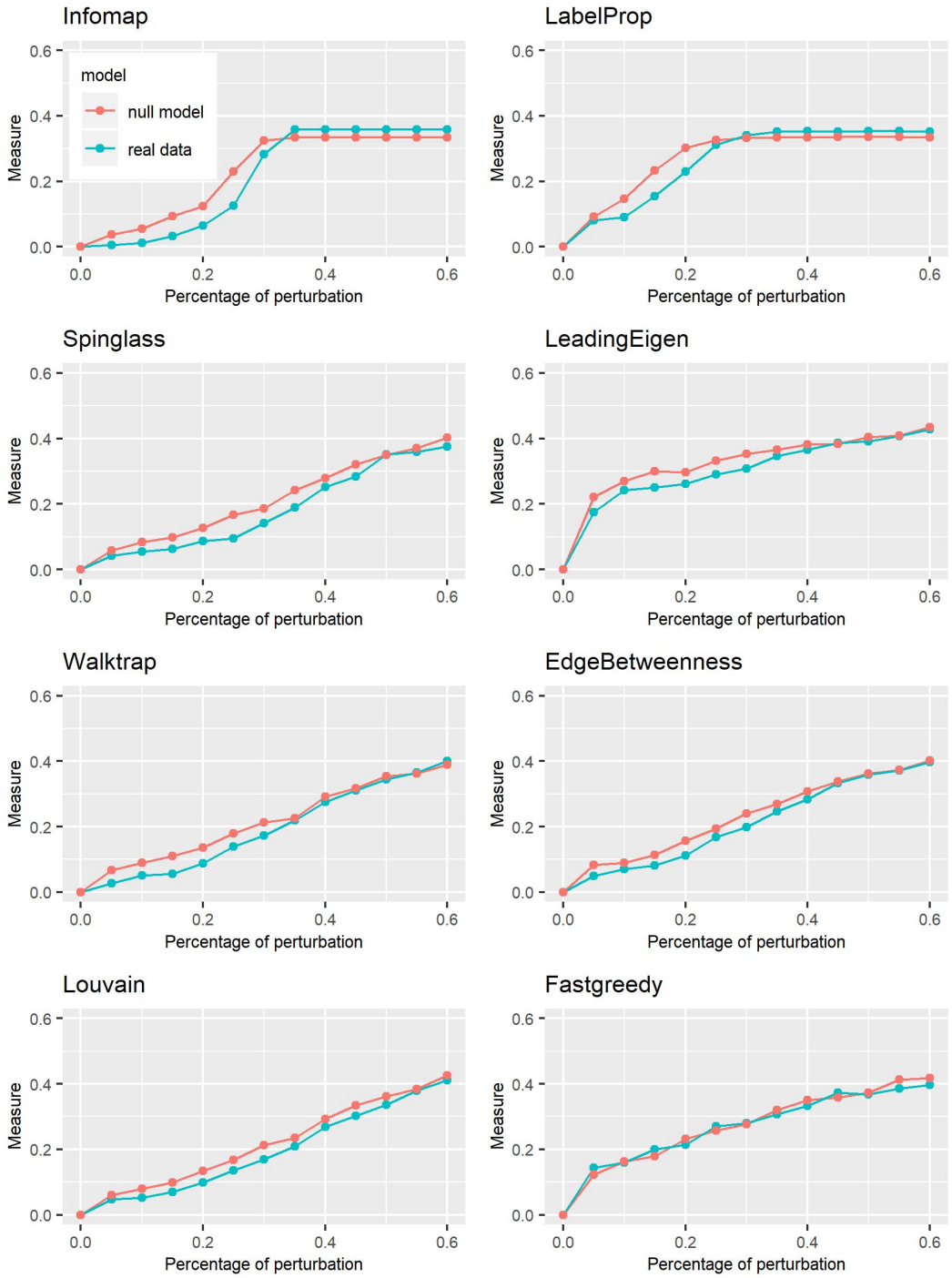


Figure 9: Plot of the VI curves of the *dk*-series null model with $d = 2.1$ and all the algorithms implemented.

Finding Optimal Normalizing Transformations via bestNormalize

by Ryan A. Peterson

Abstract The **bestNormalize** R package was designed to help users find a transformation that can effectively normalize a vector regardless of its actual distribution. Each of the many normalization techniques that have been developed has its own strengths and weaknesses, and deciding which to use until data are fully observed is difficult or impossible. This package facilitates choosing between a range of possible transformations and will automatically return the best one, i.e., the one that makes data look the *most* normal. To evaluate and compare the normalization efficacy across a suite of possible transformations, we developed a statistic based on a goodness of fit test divided by its degrees of freedom. Transformations can be seamlessly trained and applied to newly observed data and can be implemented in conjunction with **caret** and **recipes** for data preprocessing in machine learning workflows. Custom transformations and normalization statistics are supported.

Introduction

The **bestNormalize** package contains a suite of transformation-estimating functions that can be used to normalize data. The function of the same name attempts to find and execute the best of all of these potential normalizing transformations. In this package, we define "normalize" as in "to render data Gaussian", rather than transform them to a specific scale.

There are many instances where researchers may want to normalize a variable. First, there is the (often problematic) assumption of normality of the outcome (conditional on the covariates) in the classical linear regression problem. Over the years, many methods have been used to relax this assumption: generalized linear models, quantile regression, survival models, etc. One technique that is still somewhat popular in this context is to "beat the data" to look normal via some kind of normalizing transformation. This could be something as simple as a log transformation or something as complex as a Yeo-Johnson transformation (Yeo and Johnson, 2000). In fact, many complex normalization methods were designed expressly to find a transformation that could render regression residuals Gaussian. While perhaps not the most elegant solution to the problem, often, this technique works well as a quick solution. Another increasingly popular application of normalization occurs in applied regression settings with highly skewed distributions of the covariates (Kuhn and Johnson, 2013). In these settings, there exists the tendency to have high leverage points (and highly influential points), even when one centers and scales the covariates. When examining interactions, these influential points can become especially problematic since the leverage of that point gets amplified for every interaction in which it is involved. Normalization of such covariates can mitigate their leverage and influence, thereby allowing for easier model selection and more robust downstream predictor manipulations (such as principal components analysis), which can otherwise be sensitive to skew or outliers. As a result, popular model selection packages such as **caret** (Kuhn, 2017) and **recipes** (Kuhn and Wickham, 2018) have built-in mechanisms to normalize the predictor variables (they call this "preprocessing"). This concept is unique in that it forgoes the assumption of linearity between the outcome (Y) and the covariate, opting instead for a linear relationship between Y and the transformed value of the covariate (which in many cases may be more plausible).

This package is designed to make normalization effortless and consistent. We have also introduced Ordered Quantile (ORQ) normalization via the `orderNorm` function, which uses a rank mapping of the observed data to the normal distribution in order to guarantee normally distributed transformed data (if ties are not present). We have shown how ORQ normalization performs very consistently across different distributions, successfully normalizing left- or right-skewed data, multi-modal data, and even data generated from a Cauchy distribution (Peterson and Cavanaugh, 2019).

In this paper, we describe our R package **bestNormalize**, which is available via the Comprehensive R Archive Network (CRAN). First, we describe normalization methods that have been developed and that we implement in the package. Second, we describe the novel cross-validation-based estimation procedure, which we utilize to judge the normalization efficacy of our suite of normalization transformations. Third, we go through some basic examples of **bestNormalize** functionality and a simple implementation of our methods within the **recipes** package. We illustrate a more in-depth use-case in a car pricing application, performing a transform-both-sides regression as well as comparing the performance of several predictive models fit via **caret**. Finally, we conclude by discussing the pros and cons of normalization in general and future directions for the package.

Normalization methods

Many normalization transformation functions exist, and though some can be implemented well in existing R packages, **bestNormalize** puts them all under the same umbrella syntax. This section describes each transformation contained in the **bestNormalize** suite.

The Box-Cox transformation

The Box-Cox transformation was famously proposed in Box and Cox (1964) and can be implemented with differing syntax and methods in many existing packages in R (e.g., **caret**, **MASS** (Venables and Ripley, 2002), and more). It is a straightforward transformation that typically only involves one parameter, λ:

$$g(x;\lambda) = \mathbf{1}_{(\lambda\neq 0)}\frac{x^\lambda - 1}{\lambda} + \mathbf{1}_{(\lambda=0)}\log x \ ,$$

where x refers to the datum in its original unit (pre-transformation). Given multiple observations, the λ parameter can be estimated via maximum likelihood, and x must be greater than zero.

The Yeo-Johnson transformation

The Yeo-Johnson transformation (Yeo and Johnson, 2000) attempts to find the value of λ in the following equation that minimizes the Kullback-Leibler distance between the normal distribution and the transformed distribution.

$$\begin{aligned} g(x;\lambda) = &\mathbf{1}_{(\lambda\neq 0, x\geq 0)}\frac{(x+1)^\lambda - 1}{\lambda} \\ &+ \mathbf{1}_{(\lambda=0, x\geq 0)}\log(x+1) \\ &+ \mathbf{1}_{(\lambda\neq 2, x<0)}\frac{(1-x)^{2-\lambda} - 1}{\lambda - 2} \\ &+ \mathbf{1}_{(\lambda=2, x<0)} - \log(1-x) \end{aligned}$$

This method has the advantage of working without having to worry about the domain of x. As with the Box-Cox λ, this λ parameter can be estimated via maximum likelihood.

The Lambert W x F transformation

The Lambert W x F transformation, proposed in Goerg (2011) and implemented in the **LambertW** package, is essentially a mechanism that de-skews a random variable X using moments. The method is motivated by a system theory and is alleged to be able to transform any random variable into any other kind of random variable, thus being applicable to a large number of contexts. One of the package's main functions is `Gaussianize`, which is similar in spirit to the purpose of this package. However, this method may not perform as well on certain shapes of distributions as other candidate transformations; see Peterson and Cavanaugh (2019) for some examples.

The `Gaussianize` transformation can handle three types of transformations: skewed, heavy-tailed, and skewed heavy-tailed. For more details on this transformation, consult the **LambertW** documentation.[1] While the transformations contained and implemented by `bestNormalize` are reversible (i.e., 1-1), in rare circumstances, we have observed that the `lambert` function can yield non-reversible transformations.

The Ordered Quantile technique

The ORQ normalization technique (`orderNorm`) is based on the following transformation (originally discussed, as far as we can find, in Bartlett (1947) and further developed in Van der Waerden (1952)):

Let $\underline{x}$ refer to the original data. Then the transformation is:

[1] As of version 1.2.0 of **bestNormalize**, `lambert` methods are not performed by default in `bestNormalize`, but they are still available via the `allow_lambert` arguments.

$$g(\underline{x}) = \Phi^{-1}\left(\frac{\text{rank}(\underline{x}) - 1/2}{\text{length}(\underline{x})}\right)$$

This nonparametric transformation as defined works well on the observed data, but it is not trivial to implement in modern settings where the transformation needs to be applied on new data; we discussed this issue and our solution to it in Peterson and Cavanaugh (2019). Basically, on new data *within* the range of the original data, ORQ normalization will linearly interpolate between two of the original data points. On new data *outside* the range of the original data, the transformation extrapolates using a shifted logit approximation of the ranks to the original data. This is visualized below via the `iris` data set on the `Petal.Width` variable.

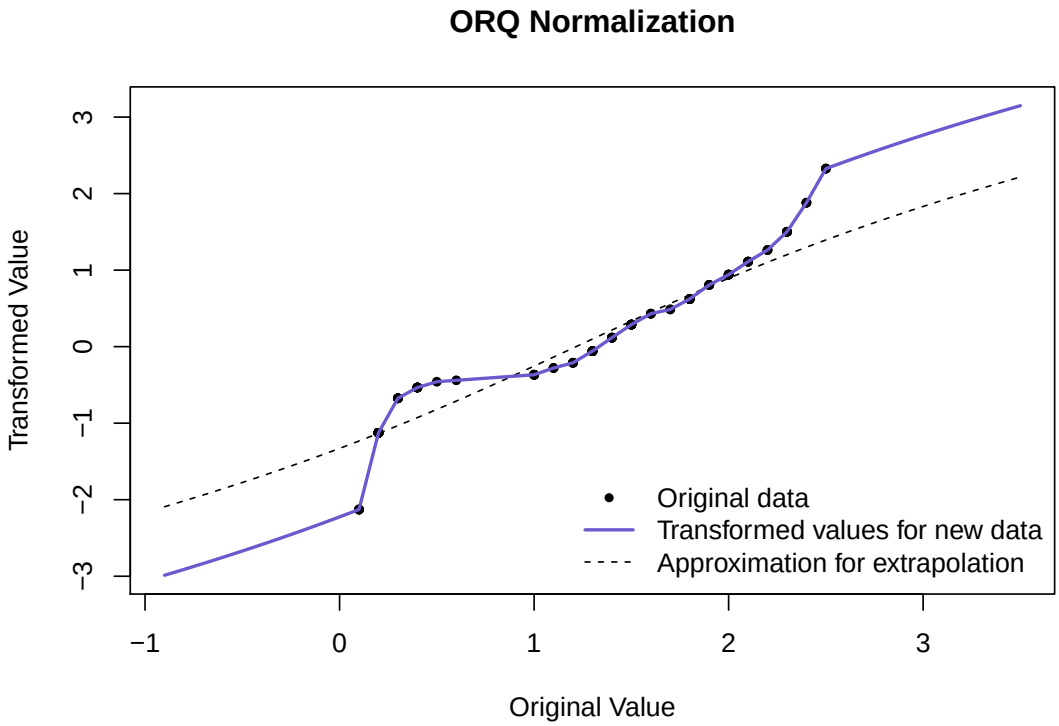


Figure 1: ORQ normalization visualization on Fisher's iris data.

The shifted logit extrapolation ensures that the function is 1-1 and can handle data outside the original (observed) domain. The effects of the approximation will usually be relatively minimal since we should not expect to see many observations outside the observed range if the training set sample size is large relative to the test set. The ORQ technique will not guarantee a normal distribution in the presence of ties, but it still could yield the best normalizing transformation when compared to the other possible approaches. More information on ORQ normalization can be found in Peterson and Cavanaugh (2019) or in the **bestNormalize** documentation.

Other included transformations

In addition to the techniques above, the **bestNormalize** package performs and evaluates:

- $\log_b(x + a)$ where $a = \max(0, -\min(x) + \epsilon)$ and $b = 10$ by default
- $\sqrt{x + a}$ where $a = \max(0, -\min(x))$ by default
- $\exp(x)$
- $\text{arcsinh}(x) = log(x + \sqrt{x^2 + 1})$

Other not-included transformations

A range of other normalization techniques has been proposed that are not included in this package (at the time of writing). These include (but are not limited to): Modified Box-Cox (Box and Cox, 1964), Manly's Exponential (Manly, 1976), John/Draper's Modulus (John and Draper, 1980), and Bickel/Doksum's Modified Box-Cox (Bickel and Doksum, 1981). However, it is straightforward to add new transformations into the same framework as other included transformations; each one is

treated as its own S3 class, so in order to add other transformations, all one must do is define a new S3 class and provide the requisite S3 methods. To this end, we encourage readers to submit a pull request to the package's GitHub page with new transformation techniques that could be then added as a default in `bestNormalize`. Otherwise, in a later section, we show how users can implement custom transformations alongside the default ones described above.

Which transformation "best normalizes" the data?

The `bestNormalize` function selects the best transformation according to an extra-sample estimate of the Pearson P statistic divided by its degrees of freedom (DF). This P statistic is defined as

$$P = \sum_{i=1}^{k} \frac{(O_i - E_i)^2}{E_i} \ ,$$

where O_i is the number observed, and E_i is the number of expected (under the hypothesis of normality) to fall into "bin" i. The bins (or "classes") are built such that observations will fall into each one with equal probability under the hypothesis of normality. A variety of alternative normality tests exist, but this particular one is relatively interpretable as a goodness of fit test, and the ratio P/DF can be compared between transformations as an absolute measure of departure from normality. Specifically, if the data in question follow a normal distribution, this ratio will be close to 1 or lower. The transformation which produces data with the lowest normality statistic is thus the most effective at normalizing the data, and gets selected by `bestNormalize`. The **bestNormalize** package utilizes **nortest** (Gross and Ligges, 2015) to compute this statistic; more information on its computation and degrees of freedom can be found in D'Agostino (1986) and Thode (2002).

Normality statistics for all candidate transformations can be estimated and compared with one simple call to `bestNormalize`, whose output makes it easy to see which transformations are viable and which are not. We have found that while complicated transformations are often *most* effective and therefore selected automatically, sometimes a simple transformation (e.g., the log or identity transforms) may be almost as effective, and ultimately the latter type will yield more interpretable results.

It is worth noting that when the normality statistic is estimated on in-sample data, the ORQ technique is predestined to be most effective since it is forcing its transformed data to follow a normal distribution exactly (Peterson and Cavanaugh, 2019). For this reason, by default, the `bestNormalize` function calculates an *out-of-sample* estimate for the P/DF statistic. Since this method necessitates cross-validation, it can be computationally frustrating for three reasons: (1) the results and the chosen transformation can depend on the seed, (2) it takes considerably longer to estimate than the in-sample statistic, and (3) it is unclear how to choose the number of folds and repeats.

In order to mediate these issues, we have built several features into **bestNormalize**. Issue (1) is only important for small sample sizes, and when it is a concern, the best transformations should look similar to one another. We address two solutions to (2) in the next section. In short, we have methods to parallelize or simplify the estimation of the statistic. For (3), we recommend 10-fold cross-validation with 5 repeats as the default, but if the sample is small, we suggest using 5 (or fewer) folds instead with more repeats; accurate estimation of P/DF requires a relatively large fold size (as a rule of thumb, 20 observations per fold seems to be enough for most cases, but this unfortunately depends on the distribution of the observed data).

Simple examples

In this section, we illustrate a simple use-case of the functions provided in **bestNormalize**.

Basic implementation

First, we will generate and plot some skewed data:

```
x <- rgamma(250, 1, 1)
```

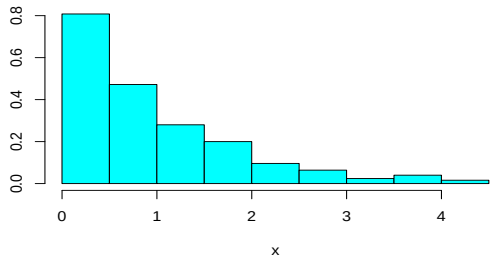


Figure 2: Simulated skewed data for simple example.

To perform a suite of potential transformations and see how effectively they normalized this vector, simply call `bestNormalize`:

```
(BNobject <- bestNormalize(x))
```

```
#> Best Normalizing transformation with 250 Observations
#>  Estimated Normality Statistics (Pearson P / df, lower => more normal):
#>  - arcsinh(x): 1.7917
#>  - Box-Cox: 1.0442
#>  - Center+scale: 3.0102
#>  - Exp(x): 9.5306
#>  - Log_b(x+a): 1.7072
#>  - orderNorm (ORQ): 1.1773
#>  - sqrt(x + a): 1.144
#>  - Yeo-Johnson: 1.1875
#> Estimation method: Out-of-sample via CV with 10 folds and 5 repeats
#>
#> Based off these, bestNormalize chose:
#> Standardized Box Cox Transformation with 250 nonmissing obs.:
#>  Estimated statistics:
#>  - lambda = 0.3254863
#>  - mean (before standardization) = -0.3659267
#>  - sd (before standardization) = 0.9807881
```

Evidently, the Box-Cox transformation performed the best, though many other transformations performed similarly. We can visualize the suite of transformations using the built-in `plot` method:

```
plot(BNobject, leg_loc = "topleft")
```

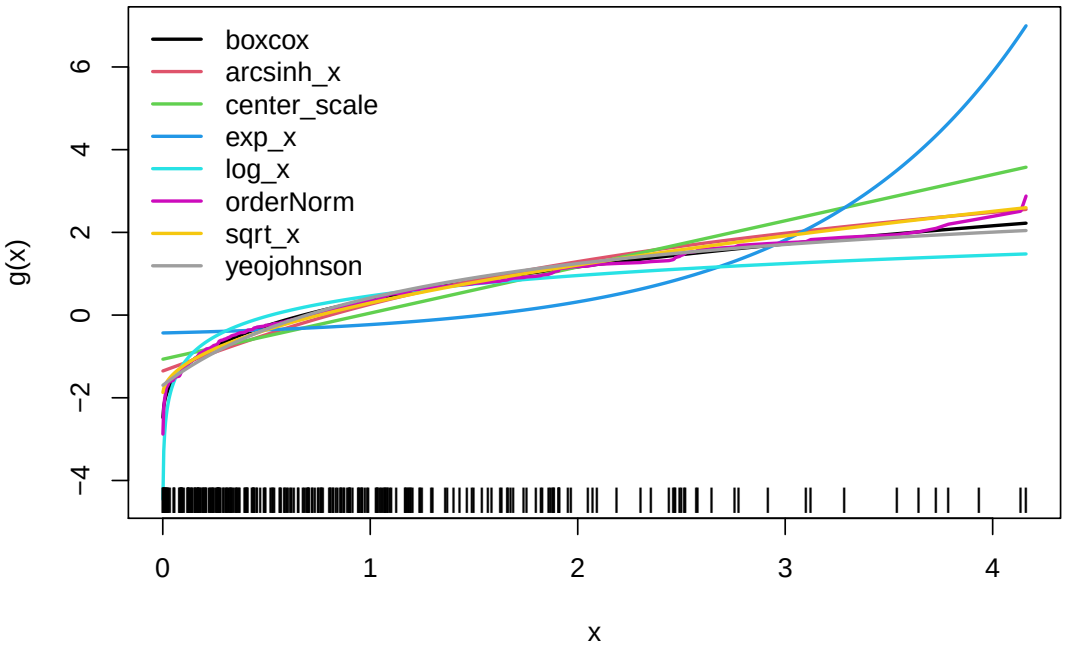


Figure 3: The suite of transformations estimated by default in **bestNormalize** (trained on simulated right-skewed data).

Finally, we can execute the best performing normalization on new data with `predict(BNobject, new_x)` or reverse the transformation with `predict(BNobject, new_x_t,inverse = TRUE)`. Note that normalized values can either be obtained using `predict` or by extracting `x.t` from the object. The best transformation, in this case, is plotted in Figure 4.

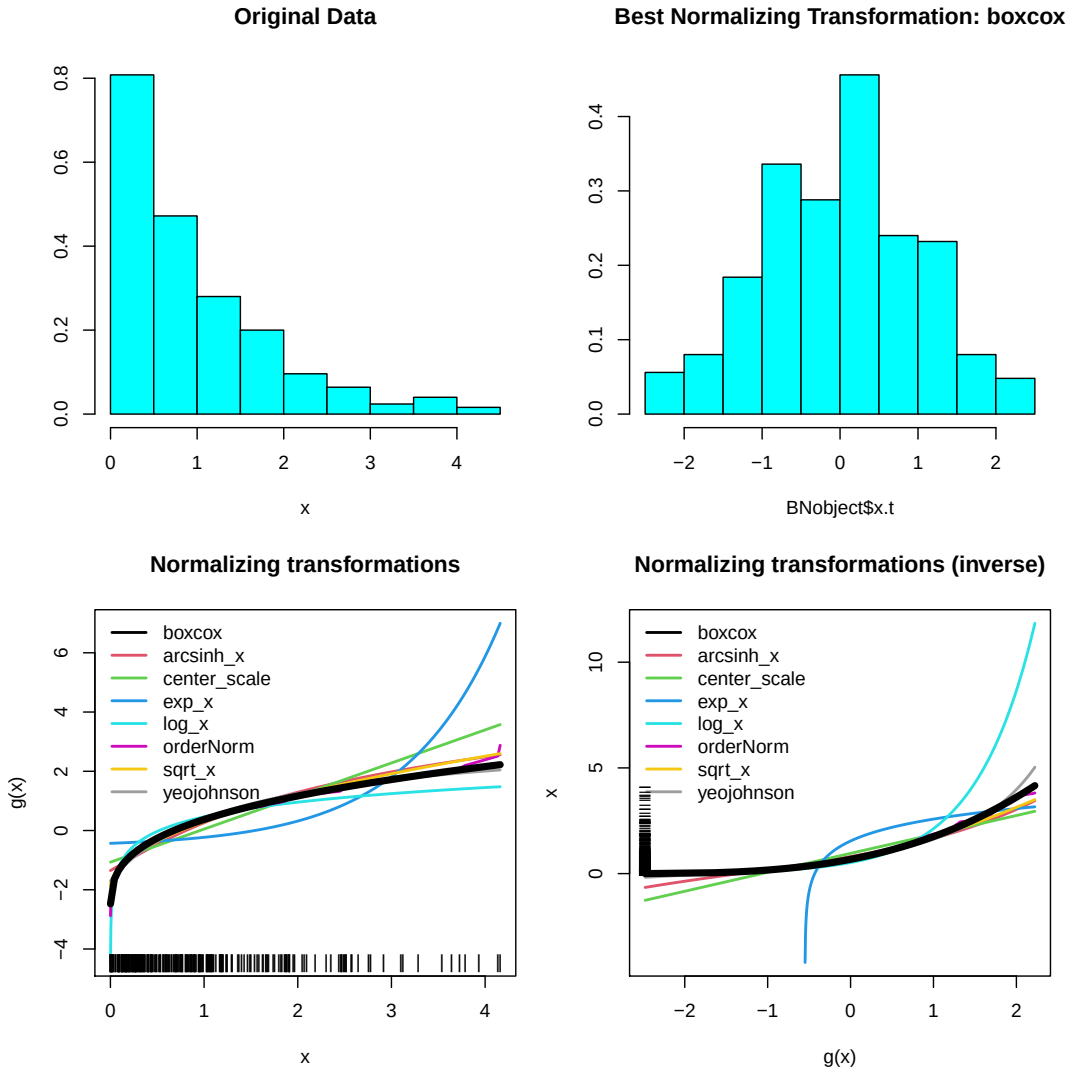


Figure 4: Summary of transformations performed on simulated right-skewed data.

Performing transformations individually

Each method can be performed (and stored) individually:

```
(arcsinh_obj <- arcsinh_x(x))

#> Standardized asinh(x) Transformation with 250 nonmissing obs.:
#>  Relevant statistics:
#>  - mean (before standardization) = 0.7383146
#>  - sd (before standardization) = 0.5458515

(boxcox_obj <- boxcox(x))

#> Standardized Box Cox Transformation with 250 nonmissing obs.:
#>  Estimated statistics:
#>  - lambda = 0.3254863
#>  - mean (before standardization) = -0.3659267
#>  - sd (before standardization) = 0.9807881

(yeojohnson_obj <- yeojohnson(x))
```

```
#> Standardized Yeo-Johnson Transformation with 250 nonmissing obs.:
#>  Estimated statistics:
#>  - lambda = -0.7080476
#>  - mean (before standardization) = 0.4405464
#>  - sd (before standardization) = 0.2592004

(lambert_obj <- lambert(x, type = "s"))

#> Standardized Lambert WxF Transformation of type s with 250 nonmissing obs.:
#>  Estimated statistics:
#>  - gamma = 0.3729
#>  - mean (before standardization) = 0.6781864
#>  - sd (before standardization) = 0.7123011

(orderNorm_obj <- orderNorm(x))

#> orderNorm Transformation with 250 nonmissing obs and no ties
#>  - Original quantiles:
#>    0%   25%   50%   75%  100%
#> 0.001 0.268 0.721 1.299 4.161
```

All normalization techniques in `bestNormalize` have their own class with convenient S3 methods and documentation. For instance, we can use the `predict` method to perform the transformation on new values using the objects we have just created, visualizing them in a plot:

```
xx <- seq(min(x), max(x), length = 100)
plot(xx, predict(arcsinh_obj, newdata = xx), type = "l", col = 1)
lines(xx, predict(boxcox_obj, newdata = xx), col = 2)
lines(xx, predict(yeojohnson_obj, newdata = xx), col = 3)
lines(xx, predict(orderNorm_obj, newdata = xx), col = 4)
```

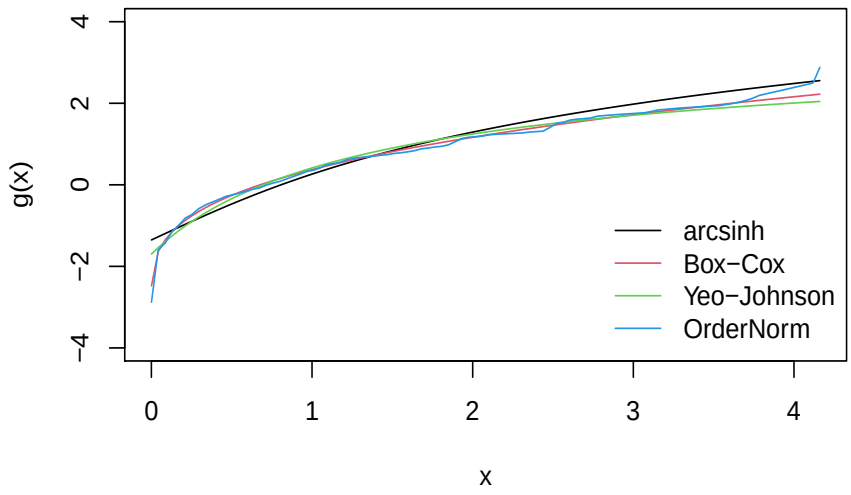


Figure 5: Manually plotting transformations trained on simulated right-skewed data.

In-sample normalization efficacy

To examine how each of the normalization methods performed (in-sample), we can visualize the transformed values in histograms (Figure 6), which plot the transformed data, `x.t`, stored in the transformation objects we created previously.

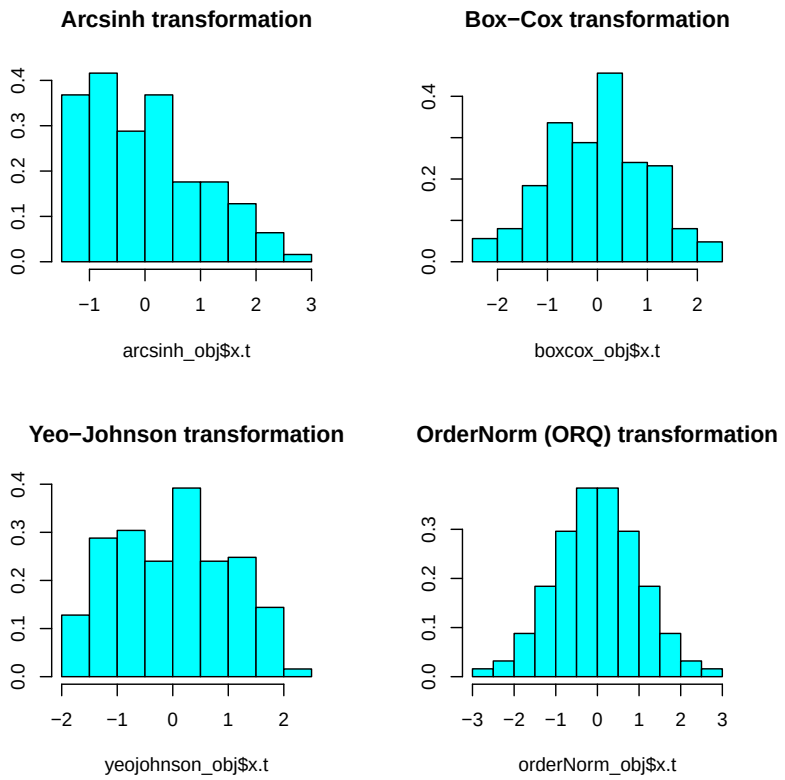


Figure 6: Normalized values for trained transformations on simulated right-skewed data.

Evidently, ORQ normalization appears to have worked perfectly to normalize the data (as expected), and the Box-Cox method seemed to do quite well too.

Out-of-sample normalization efficacy

The `bestNormalize` function performs repeated (r=5) 10-fold cross-validation (CV) by default and stores the estimated normality statistic for each left-out fold/repeat into `oos_preds`. Users can access and visualize these results via a boxplot (see below), which may give some insight into whether the transformation is truly preferred by the normality statistic or if another (possibly simpler) transformation can be applied that would achieve the approximately the same results. In this example, Box-Cox, square-root, Yeo-Johnson, and ORQ seem to do similarly well, whereas the identity transform[2], hyperbolic arc-sine, logging, and exponentiation are performing worse.

```
boxplot(BNobject$oos_preds, log = 'y')
abline(h = 1, col = "green3")
```

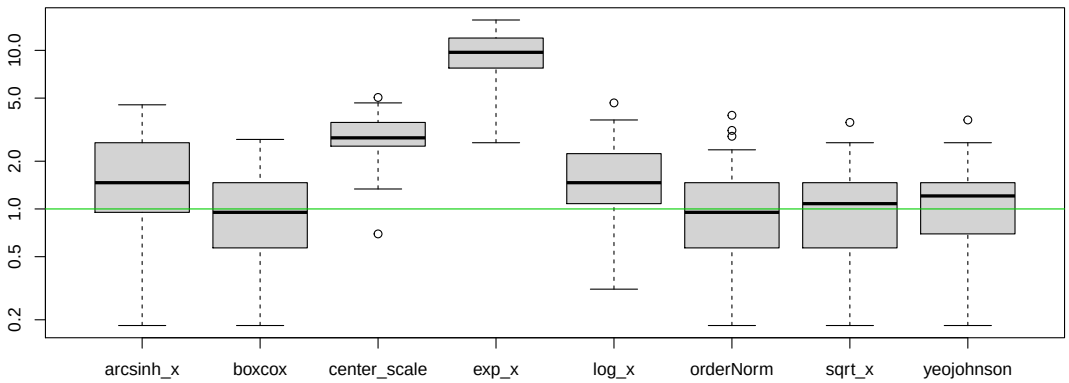


Figure 7: Cross-validation results for each normalization method, where our estimated normality statistic is plotted on the y-axis.

[2]Since `standardize=TRUE`, the identity transformation is represented in Figure 7 by `center_scale`, which yields the exact same normality statistic.

Leave-one-out CV can be optionally performed in `bestNormalize` via the `loo` argument, which, if set to `TRUE`, will compute the leave-one-out CV transformations for each observation and method. Specifically, `bestNormalize` will be run n separate times where each observation is individually left out of the fitting process and subsequently plugged back in to get a "leave-one-out transformed value". Instead of taking the mean across repeats and folds, in this case, we estimate normalization efficacy using the full distribution of leave-one-out transformed values. This option is computationally intensive. Note that as with the "in-sample" normality statistics, the leave-one-out CV approach tends to select the ORQ transformation since ORQ's performance improves as the number of points in the training set relative to the testing set increases.

```
bestNormalize(x, loo = TRUE)
```

```
#> Best Normalizing transformation with 250 Observations
#>  Estimated Normality Statistics (Pearson P / df, lower => more normal):
#>  - arcsinh(x): 4.42
#>  - Box-Cox: 0.7055
#>  - Center+scale: 8.258
#>  - Exp(x): 62.085
#>  - Log_b(x+a): 3.546
#>  - orderNorm (ORQ): 0.012
#>  - sqrt(x + a): 0.9145
#>  - Yeo-Johnson: 1.608
#> Estimation method: Out-of-sample via leave-one-out CV
#>
#> Based off these, bestNormalize chose:
#> orderNorm Transformation with 250 nonmissing obs and no ties
#>  - Original quantiles:
#>    0%   25%   50%   75%  100%
#> 0.001 0.268 0.721 1.299 4.161
```

Important features

Improving speed of estimation

Because `bestNormalize` uses repeated CV by default to estimate the out-of-sample normalization efficacy, it can be quite slow for larger objects. There are several means of speeding up the process. Each comes with some pros and cons. The first option is to specify `out_of_sample = FALSE`. This will highly speed up the process. However, for reasons previously discussed, ORQ normalization will always be chosen unless `allow_orderNorm = FALSE`. Therefore, a user might as well use the `orderNorm` function directly as opposed to only setting `out_of_sample = FALSE` since the end result will be the same (and `orderNorm` will run much faster). Note below that the in-sample normality results may differ slightly from the leave-one-out even when this may be unexpected (i.e., for the log transformation); this is due to slight differences in the standardization statistics.

```
bestNormalize(x, allow_orderNorm = FALSE, out_of_sample = FALSE)
```

```
#> Best Normalizing transformation with 250 Observations
#>  Estimated Normality Statistics (Pearson P / df, lower => more normal):
#>  - arcsinh(x): 4.401
#>  - Box-Cox: 0.7435
#>  - Center+scale: 8.087
#>  - Exp(x): 64.6975
#>  - Log_b(x+a): 3.47
#>  - sqrt(x + a): 0.9145
#>  - Yeo-Johnson: 1.7125
#> Estimation method: In-sample
#>
#> Based off these, bestNormalize chose:
#> Standardized Box Cox Transformation with 250 nonmissing obs.:
#>  Estimated statistics:
#>  - lambda = 0.3254863
#>  - mean (before standardization) = -0.3659267
#>  - sd (before standardization) = 0.9807881
```

Another option to improve estimation efficiency is to use the built-in parallelization functionality. The repeated CV process can be parallelized via the `cluster` argument and the **parallel** and **doRNG** (Gaujoux, 2020) packages. A cluster can be set up with `makeCluster` and passed to `bestNormalize` via the `cluster =` argument.

```
cl <- parallel::makeCluster(5)
b <- bestNormalize(x, cluster = cl, r = 10, quiet = TRUE)
parallel::stopCluster(cl)
```

The amount by which this parallelization will speed up the estimation of out-of-sample estimates depends (for the most part) on the number of repeats, the number of cores, and the sample size of the vector to be normalized. The plot below shows the estimation time for a run of `bestNormalize` with 15 repeats of 10-fold CV on a gamma-distributed random variable with various sample sizes and numbers of cores.

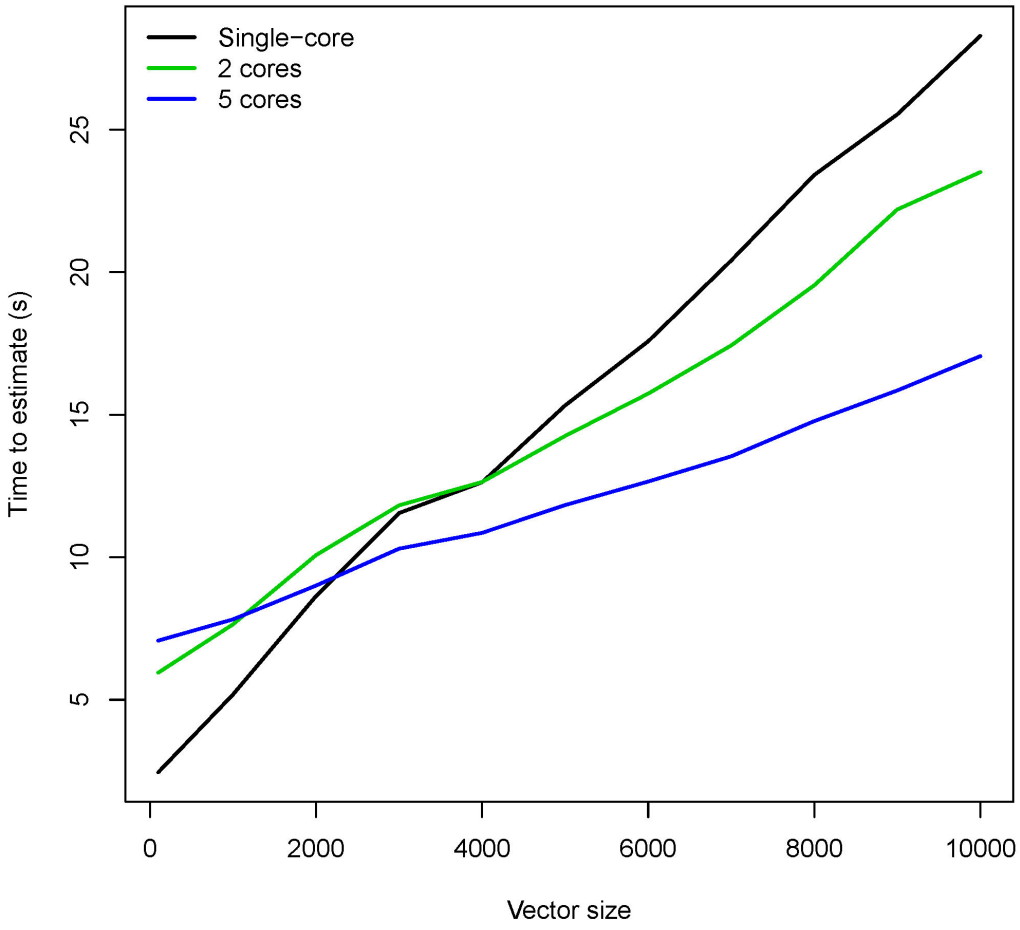


Figure 8: Potential speedup using parallelization functionality.

Implementation with caret, recipes

The `step_best_normalize` and the `step_orderNorm` functions can be utilized in conjunction with the **recipes** package to preprocess data in machine learning workflows with **tidymodels** (Kuhn and Wickham, 2020) or in combination with **caret**. The basic usage within **recipes** is shown below; for implementation with **caret**, refer to this paper's application.

```
rec <- recipe( ~ ., data = iris)  %>%                       # Initialize recipe
  step_best_normalize(all_predictors(), -all_nominal()) %>% # Transform predictors
  prep(iris) %>%                                            # Prep (train) recipe
  bake(iris)                                                # Bake (apply) recipe
```

Options can be supplied to `step_best_normalize` to speed up or alter performance via the `transform_options` argument, which passes a list of options to `bestNormalize`.

Additional customization

Two important means of customization are available: 1) users may add custom transformation functions to be assessed alongside the default suite of normalization methods, and 2) users may change the statistic used "under the hood" by `bestNormalize` to estimate the departure from normality of the transformed data. This section contains examples and guidance for both extensions.

1) Adding user-defined functions

Via the `new_transforms` argument, users can use `bestNormalize`'s machinery to compare custom, user-defined transformation functions to those included in the package. Below, I consider an example where a user may wish to compare the cube-root function with those provided in the package. `bestNormalize` requires two functions to implement this: the transformation function and an associated `predict` method. The custom cube-root transformation shown below is simple, but its skeleton can readily be made arbitrarily more complex.

```
## Define custom function
cuberoot_x <- function(x, ...) {
  x.t <- (x)^(1/3)

  # Get in-sample normality statistic results
  ptest <- nortest::pearson.test(x.t)

  val <- list(
    x.t = x.t,
    x = x,
    n = length(x.t) - sum(is.na(x)),
    norm_stat = unname(ptest$statistic / ptest$df)
  )

  # Assign class, return
  class(val) <- c('cuberoot_x')
  val
}

# S3 method that is used to apply the transformation to newly observed data
predict.cuberoot_x <- function(object, newdata = NULL, inverse = FALSE, ...) {

  # If no data supplied and not inverse
  if (is.null(newdata) & !inverse)
    newdata <- object$x

  # If no data supplied and inverse transformation is requested
  if (is.null(newdata) & inverse)
    newdata <- object$x.t

  # Perform inverse transformation
  if (inverse) {
    # Reverse-cube-root (cube)
    val <-  newdata^3

    # Otherwise, perform transformation as estimated
  } else if (!inverse) {
    val <- (newdata)^(1/3)
  }

  # Return transformed data
  unname(val)
}

## Optional: print S3 method
print.cuberoot_x <- function(x, ...) {
  cat('cuberoot(x) Transformation with', x$n, 'nonmissing obs.\n')
}
```

These functions can then be passed as a named list to `bestNormalize`:

```
custom_transform <- list(
  cuberoot_x = cuberoot_x,
  predict.cuberoot_x = predict.cuberoot_x,
  print.cuberoot_x = print.cuberoot_x
)

set.seed(123129)
x <- rgamma(100, 1, 1)
(b <- bestNormalize(x = x, new_transforms = custom_transform))
```

```
#> Best Normalizing transformation with 100 Observations
#>  Estimated Normality Statistics (Pearson P / df, lower => more normal):
#>  - arcsinh(x): 1.2347
#>  - Box-Cox: 1.0267
#>  - Center+scale: 2.0027
#>  - cuberoot_x: 0.9787
#>  - Exp(x): 4.7947
#>  - Log_b(x+a): 1.3547
#>  - orderNorm (ORQ): 1.1627
#>  - sqrt(x + a): 1.0907
#>  - Yeo-Johnson: 1.0987
#> Estimation method: Out-of-sample via CV with 10 folds and 5 repeats
#>
#> Based off these, bestNormalize chose:
#> cuberoot(x) Transformation with 100 nonmissing obs.
```

Evidently, the cube-root was the best normalizing transformation for this gamma-distributed random variable, performing comparably to the Box-Cox transformation.

2) Re-defining normality

The question "what is normal?" outside of a statistical discussion is quite loaded and subjective. Even in statistical discussions, many authors have contributed to the question of how to best detect departures from normality; these solutions are diverse, and several have been implemented well in **nortest** already. In order to accommodate those with varying opinions on the best definition of normality, we have included a feature that allows users to specify a custom definition of a normality statistic. This customization can be accomplished via the `norm_stat_fn` argument, which takes a function that will then be applied in lieu of the Pearson test statistic divided by its degree of freedom to assess normality.

The user-defined function must take an argument `x`, which indicates the data on which a user wants to evaluate the statistic.

Here is an example using the Lilliefors (Kolmogorov-Smirnov) normality test statistic:

```
bestNormalize(x, norm_stat_fn = function(x) nortest::lillie.test(x)$stat)
```

```
#> Best Normalizing transformation with 100 Observations
#>  Estimated Normality Statistics (using custom normalization statistic)
#>  - arcsinh(x): 0.1958
#>  - Box-Cox: 0.1785
#>  - Center+scale: 0.2219
#>  - Exp(x): 0.3299
#>  - Log_b(x+a): 0.1959
#>  - orderNorm (ORQ): 0.186
#>  - sqrt(x + a): 0.1829
#>  - Yeo-Johnson: 0.1872
#> Estimation method: Out-of-sample via CV with 10 folds and 5 repeats
#>
#> Based off these, bestNormalize chose:
#> Standardized Box Cox Transformation with 100 nonmissing obs.:
#>  Estimated statistics:
#>  - lambda = 0.3281193
#>  - mean (before standardization) = -0.1263882
#>  - sd (before standardization) = 0.9913552
```

Here is an example using the Lillifors (Kolmogorov-Smirnov) normality test's p-value:

```
(dont_do_this <- bestNormalize(x, norm_stat_fn = function(x) nortest::lillie.test(x)$p))
```

```
#> Best Normalizing transformation with 100 Observations
#>  Estimated Normality Statistics (using custom normalization statistic)
#>  - arcsinh(x): 0.4327
#>  - Box-Cox: 0.4831
#>  - Center+scale: 0.2958
#>  - Exp(x): 0.0675
#>  - Log_b(x+a): 0.3589
#>  - orderNorm (ORQ): 0.4492
#>  - sqrt(x + a): 0.4899
#>  - Yeo-Johnson: 0.4531
#> Estimation method: Out-of-sample via CV with 10 folds and 5 repeats
#>
#> Based off these, bestNormalize chose:
#> Standardized exp(x) Transformation with 100 nonmissing obs.:
#>  Relevant statistics:
#>  - mean (before standardization) = 6.885396
#>  - sd (before standardization) = 13.66084
```

Note: `bestNormalize` will attempt to minimize this statistic by default, which is definitely not what you want to do when calculating the p-value. This is seen in the example above, where the **worst** normalization transformation, exponentiation, is chosen. In this case, a user is advised to either manually select the best one or reverse their defined normalization statistic (in this case by subtracting it from 1):

```
best_transform <- names(which.max(dont_do_this$norm_stats))
do_this <- dont_do_this$other_transforms[[best_transform]]
or_this <- bestNormalize(x, norm_stat_fn = function(x) 1-nortest::lillie.test(x)$p)
```

A p-value for normality should not be routinely used as the sole selector of a normalizing transformation. A normality test's p-value, as a measure of the departure from normality, is confounded by the sample size (a high sample size may yield strong evidence of a practically insignificant departure from normality). Therefore, we suggest the statistic used should estimate the departure from normality rather the strength of evidence against normality (e.g., Royston, 1991).

Application to Autotrader data

Background

The `autotrader` data set was scraped from the autotrader website as part of this package (and because at the time of data collection in 2017, the package author needed to purchase a car). We apply the **bestNormalize** functionality to de-skew mileage, age, and price in a pricing model. See `?autotrader` for more information on this data set.

```
data("autotrader")
autotrader$yearsold <- 2017 - autotrader$Year
```

Table 1: Sample characteristics of 'autotrader' data.

	Overall (N=6,283)
Make	
- Acura	185 (2.9%)
- Buick	252 (4.0%)
- Chevrolet	1,257 (20.0%)
- GMC	492 (7.8%)
- Honda	1,029 (16.4%)
- Hyundai	381 (6.1%)
- Mazda	272 (4.3%)
- Nissan	735 (11.7%)
- Pontiac	63 (1.0%)
- Toyota	1,202 (19.1%)
- Volkswagen	415 (6.6%)
Price ($)	
- Mean (SD)	17,145 (8,346)
- Range	722 - 64,998
Mileage	
- Mean (SD)	63,638 (49,125)
- Range	2 - 325,556
Year	
- Mean (SD)	2011.9 (3.5)
- Range	2000.0 - 2016.0
Age (years old)	
- Mean (SD)	5.1 (3.5)
- Range	1.0 - 17.0

Transform-both-sides regression

Transform-both-sides (TBS) regression has several benefits that have been explored thoroughly elsewhere (see Harrell (2015) for an overview). Importantly, TBS regression can often (though not always) yield models that better satisfy assumptions of linear regression and mitigate the influence of outliers/skew. This approach has been shown to be useful in shrinking the size of prediction intervals while maintaining closer to nominal coverage in this data set (Peterson and Cavanaugh, 2019).

First, we will normalize the outcome (price).

```
(priceBN <- bestNormalize(autotrader$price))
```

```
#> Best Normalizing transformation with 6283 Observations
#>  Estimated Normality Statistics (Pearson P / df, lower => more normal):
#>  - arcsinh(x): 3.8573
#>  - Box-Cox: 2.2291
#>  - Center+scale: 3.5532
#>  - Log_b(x+a): 3.8573
#>  - orderNorm (ORQ): 1.1384
#>  - sqrt(x + a): 2.1977
#>  - Yeo-Johnson: 2.2291
#> Estimation method: Out-of-sample via CV with 10 folds and 5 repeats
#>
#> Based off these, bestNormalize chose:
#> orderNorm Transformation with 6283 nonmissing obs and ties
#>  - 2465 unique values
#>  - Original quantiles:
#>    0%   25%   50%   75%  100%
#>   722 11499 15998 21497 64998
```

We can see that the estimated normality statistic for the ORQ transformation is close to 1, so we know it is performing quite well despite the ties in the data. It is also performing considerably better than all of the other transformations.

```
(mileageBN <- bestNormalize(autotrader$mileage))

#> Best Normalizing transformation with 6283 Observations
#>  Estimated Normality Statistics (Pearson P / df, lower => more normal):
#>  - arcsinh(x): 3.4332
#>  - Box-Cox: 3.0903
#>  - Center+scale: 14.7488
#>  - Log_b(x+a): 3.4354
#>  - orderNorm (ORQ): 1.1514
#>  - sqrt(x + a): 5.1041
#>  - Yeo-Johnson: 3.0891
#> Estimation method: Out-of-sample via CV with 10 folds and 5 repeats
#>
#> Based off these, bestNormalize chose:
#> orderNorm Transformation with 6283 nonmissing obs and ties
#>  - 6077 unique values
#>  - Original quantiles:
#>     0%    25%    50%    75%   100%
#>      2  29099  44800  88950 325556
```

Similarly, the ORQ normalization performed best for mileage.

```
(yearsoldBN <- bestNormalize(autotrader$yearsold))

#> Best Normalizing transformation with 6283 Observations
#>  Estimated Normality Statistics (Pearson P / df, lower => more normal):
#>  - arcsinh(x): 83.2706
#>  - Box-Cox: 83.2909
#>  - Center+scale: 83.4324
#>  - Exp(x): 574.3318
#>  - Log_b(x+a): 83.0756
#>  - orderNorm (ORQ): 81.3615
#>  - sqrt(x + a): 83.4373
#>  - Yeo-Johnson: 84.0028
#> Estimation method: Out-of-sample via CV with 10 folds and 5 repeats
#>
#> Based off these, bestNormalize chose:
#> orderNorm Transformation with 6283 nonmissing obs and ties
#>  - 17 unique values
#>  - Original quantiles:
#>   0%  25%  50%  75% 100%
#>    1    3    4    7   17
```

For age, we see something peculiar; none of the normalizing transformations performed well according to the normality statistics. By plotting the data, it becomes evident that the frequency of ties in age makes it very difficult to find a normalizing transformation (see figure below). Even so, `orderNorm` is chosen as it has the lowest estimated P/DF statistic.

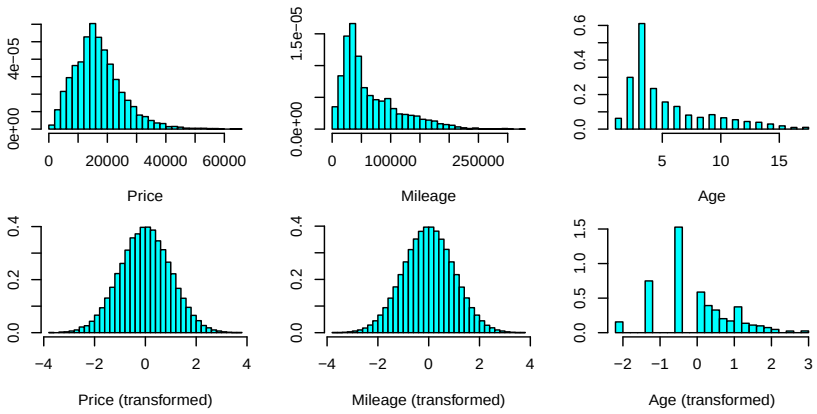


Figure 9: Distributions of car variables before and after normalization.

Next, we will fit a linear model on the transformed values of each variable for our TBS regression. The reverse-transformation functions will allow us to visualize how these variables affect model predictions in terms of their original units.

```
p.t <- priceBN$x.t; m.t <- mileageBN$x.t; yo.t <- yearsoldBN$x.t
fit <- lm(p.t ~ m.t + yo.t)
```

Table 2: TBS regression results for autotrader data.

Variable	Estimate	Std. Error	t value	Pr(>|t|)
Intercept	0.005	0.010	0.553	0.58
g(Mileage)	-0.234	0.016	-14.966	< 0.001
g(Age)	-0.441	0.016	-27.134	< 0.001

Unsurprisingly, we find that there are very significant relationships between transformed car price, mileage, and age. However, to interpret these values, we must resort to visualizations since there is no inherent meaning of a "one-unit increase" in the ORQ normalized measurements. We utilize the **visreg** package (Breheny and Burchett, 2017) to perform our visualizations, using `predict.bestNormalize` in conjunction with **visreg**'s `trans` and `xtrans` options to view the relationship in terms of the original unit for the response and covariate respectively (formatting omitted).[3] For the sake of illustration, we have also plotted the estimated effect of a generalized additive (spline) model fit with **mgcv** (Wood, 2011).

```
visreg(fit, "m.t")
visreg(fit, "m.t",
       partial = TRUE,
       trans = function(price.t)
         predict(priceBN, newdata = price.t, inverse = TRUE)/1000,
       xtrans = function(mileage.t)
         predict(mileageBN, newdata = mileage.t, inverse = TRUE)
       )
```

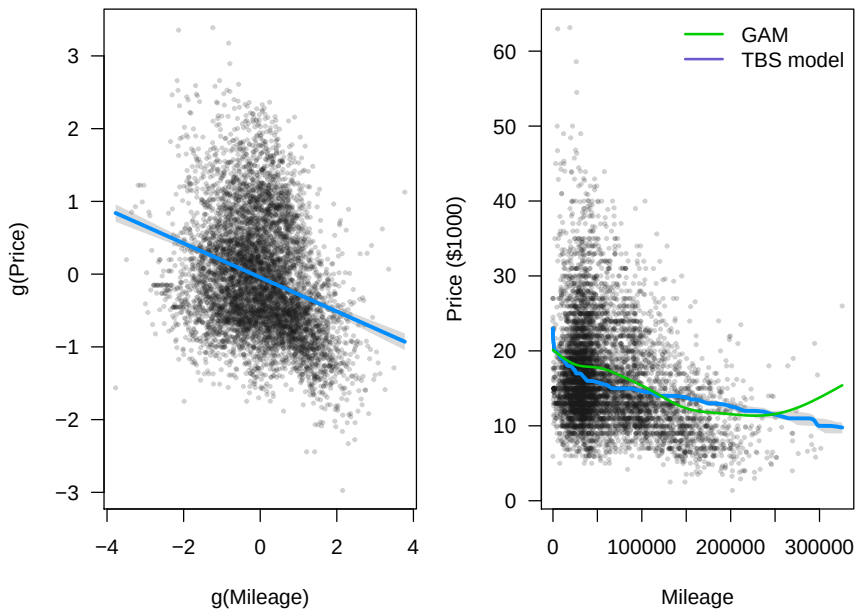


Figure 10: TBS regression visualized on transformed units (left) and original units (right).

Below, we visualize the age effect, demonstrating how one might visualize the effect outside of **visreg** (plot formatting is omitted).

[3] Alternatively, one can use **scales** (Wickham and Seidel, 2020) and **ggplot2** (Wickham, 2016) to visualize any transformation fit using **bestNormalize**; instructions are included in the package vignette.

```
# Set up data for plotting line
new_yo <- seq(min(autotrader$yearsold), max(autotrader$yearsold), len = 100)
newX <- data.frame(yearsold = new_yo, mileage = median(autotrader$mileage))
newXt <- data.frame(yo.t = predict(yearsoldBN, newX$yearsold),
                    m.t = predict(mileageBN, newX$mileage))

line_vals_t <- predict(fit, newdata = newXt) # Calculate line (transformed)
line_vals <- predict(priceBN, newdata = line_vals_t, inverse = TRUE)
plot(autotrader$yearsold, autotrader$price)
lines(new_yo, line_vals)
```

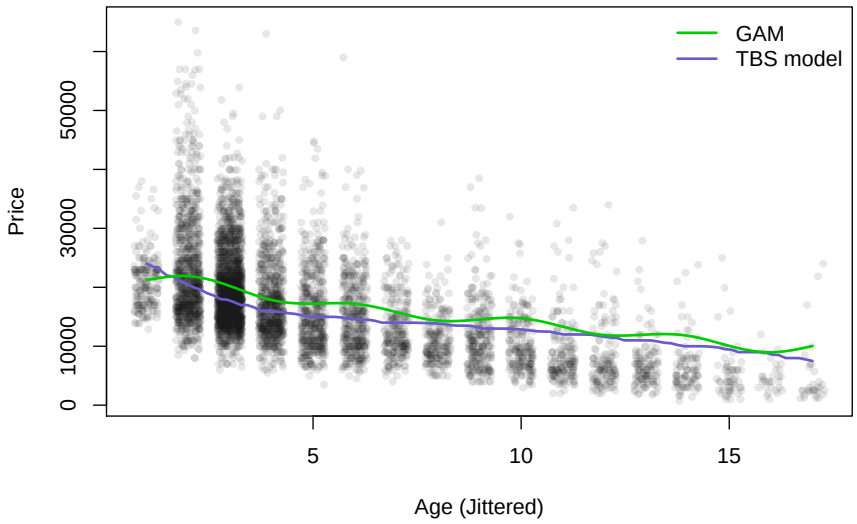


Figure 11: Age effect on car price (re-transformed to original unit).

Implementation with recipes

To build a predictive model for the price variable that uses each vehicle's model and make in addition to its mileage and age, we can utilize the **caret** and **recipes** functionality to do so. This section outlines how to use **bestNormalize** in conjunction with these other popular ML packages. Price is logged instead of ORQ transformed in order to facilitate the interpretation of measures for prediction accuracy.

```
library(tidymodels)
library(caret)
library(recipes)

set.seed(321)
df_split <- initial_split(autotrader, prop = .9)
df_train <- training(df_split)
df_test <- testing(df_split)

rec <- recipe(price ~ Make + model +  mileage + status + Year, df_train) %>%
  step_mutate(years_old = 2017 - Year) %>%
  step_rm(Year) %>%
  step_log(price) %>%
  step_best_normalize(all_predictors(), -all_nominal()) %>%
  step_other(all_nominal(), threshold = 10) %>%
  step_dummy(all_nominal()) %>%
  prep()

fit1 <- train(price ~ ., bake(rec, NULL), method = 'glmnet')
fit2 <- train(price ~ ., bake(rec, NULL), method = 'earth')
fit3 <- train(price ~ ., bake(rec, NULL), method = 'rf')

r <- resamples(fits <- list(glmnet = fit1, earth = fit2, rf = fit3))
summary(r) # Extra-sample CV results
```

Table 3: CV prediction accuracy of various ML methods.

	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	NA's
MAE							
glmnet	0.181	0.184	0.186	0.189	0.194	0.198	0
earth	0.147	0.151	0.154	0.155	0.158	0.163	0
rf	0.136	0.141	0.143	0.144	0.147	0.157	0
RMSE							
glmnet	0.242	0.247	0.252	0.256	0.264	0.276	0
earth	0.203	0.209	0.214	0.217	0.226	0.235	0
rf	0.193	0.208	0.213	0.210	0.215	0.217	0
RSQ							
glmnet	0.767	0.772	0.785	0.782	0.789	0.801	0
earth	0.807	0.833	0.845	0.842	0.855	0.864	0
rf	0.835	0.845	0.855	0.854	0.860	0.873	0

Evidently, the random forest generally performed better in cross-validated prediction metrics, achieving a higher R-squared (RSQ), lower root-mean-squared error (RMSE), and lower mean absolute error (MAE). Since price was logged, RMSE and MAE are on the log scale. For the test set, we calculate these quantities in price's original unit (2017 US dollars) using the **yardstick** package (Kuhn and Vaughan, 2020).

```
# Out of sample prediction accuracy
results <- lapply(fits, function(x) {
  p <- c(predict(x, newdata = bake(rec, df_test)))
  yardstick::metrics(data.frame(est = exp(p), truth = df_test$price),
                     truth = truth, estimate = est)
})
results
```

Table 4: Test data prediction accuracy of various ML methods. RMSE and MAE can be interpreted in terms of 2017 US dollars.

Method	RMSE	RSQ	MAE
glmnet	4076	0.772	2847
earth	3619	0.814	2500
rf	3257	0.853	2294

After normalization of mileage and age, a random forest had the optimal predictive performance on car price given a car's make, model, age, and mileage compared to other ML models, achieving out-of-sample R-squared 0.853 on a left-out test data set. We conjecture that the random forest performs best because it can better capture differential depreciation by make and model than the other methods.

Discussion

We have shown how the **bestNormalize** package can effectively and efficiently find the best normalizing transformation for a vector or set of vectors. However, normalization is by no means something that should be applied universally and without motivation. In situations where units have meaning, normalizing prior to analysis can contaminate the relationships suspected in the data and/or reduce predictive accuracy. Further, depending on the type of transformations used, interpreting regression coefficients post-transformation can be difficult or impossible without using a figure since the transformation function itself will look completely different for different distributions. So, while normalization transformations may well be able to increase the robustness of results and mitigate violations to the classical linear regression assumption of Gaussian residuals, it is by no means a universal solution.

On the other hand, when hypotheses are exploratory or when data is of poor quality with high amounts of skew/outliers, normalization can be an effective means of mitigating downstream

issues this can cause in the analyses. For example, in machine learning contexts, some predictor manipulations rely on second-order statistics (e.g., principal components analysis or partial least squares), for which the variance calculation can be sensitive to skew and outliers. Normalizing transformations can improve the quality and stability of these calculations. Similarly, predictor normalization reduces the tendency for high-leverage points to have their leverage propagated into engineered features such as interactions or polynomials. Ultimately, these benefits can often produce predictive models that are more robust and stable.

We focused on making this package useful in a variety of machine learning workflows. We are enthusiastic in our support of **bestNormalize**, and will continue to maintain the package while it is found to be useful by R users. We hope to continue to build up the repertoire of candidate transformations using the same infrastructure so that additional ones can be considered by default in the future.

Bibliography

M. S. Bartlett. The use of transformations. *Biometrics*, 3(1):39–52, 1947. ISSN 0006341X, 15410420. URL https://doi.org/10.2307/3001536. [p311]

P. J. Bickel and K. A. Doksum. An analysis of transformations revisited. *Journal of the American Statistical Association*, 76(374):296–311, 1981. URL https://doi.org/10.1080/01621459.1981.10477649. [p312]

G. E. P. Box and D. R. Cox. An analysis of transformations. *Journal of the Royal Statistical Society. Series B (Methodological)*, 26(2):211–252, 1964. ISSN 00359246. URL https://doi.org/10.2307/2984418. [p311, 312]

P. Breheny and W. Burchett. Visualization of regression models using visreg. *The R Journal*, 9(2):56–71, 2017. [p325]

R. B. D'Agostino. *Goodness-of-fit-techniques*, volume 68. CRC press, 1986. [p313]

R. Gaujoux. *doRNG: Generic Reproducible Parallel Backend for 'foreach' Loops*, 2020. URL https://CRAN.R-project.org/package=doRNG. R package version 1.8.2. [p319]

G. M. Goerg. Lambert w random variables-a new family of generalized skewed distributions with applications to risk estimation. *The Annals of Applied Statistics*, 5(3):2197–2230, 09 2011. URL https://doi.org/10.1214/11-AOAS457. [p311]

J. Gross and U. Ligges. *nortest: Tests for Normality*, 2015. URL https://CRAN.R-project.org/package=nortest. R package version 1.0-4. [p313]

F. E. Harrell. *Regression modeling strategies: with applications to linear models, logistic and ordinal regression, and survival analysis*. Springer, 2015. [p323]

J. A. John and N. R. Draper. An alternative family of transformations. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 29(2):190–197, 1980. URL https://doi.org/10.2307/2986305. [p312]

M. Kuhn. *caret: Classification and Regression Training*, 2017. URL https://CRAN.R-project.org/package=caret. R package version 6.0-78. [p310]

M. Kuhn and K. Johnson. *Applied Predictive Modeling*. Springer, 2013. URL https://doi.org/10.1007/978-1-4614-6849-3. [p310]

M. Kuhn and D. Vaughan. *yardstick: Tidy Characterizations of Model Performance*, 2020. URL https://CRAN.R-project.org/package=yardstick. R package version 0.0.7. [p327]

M. Kuhn and H. Wickham. *recipes: Preprocessing Tools to Create Design Matrices*, 2018. URL https://CRAN.R-project.org/package=recipes. R package version 0.1.2. [p310]

M. Kuhn and H. Wickham. *tidymodels: Easily Install and Load the 'Tidymodels' Packages*, 2020. URL https://CRAN.R-project.org/package=tidymodels. R package version 0.1.0. [p319]

B. F. J. Manly. Exponential data transformations. *Journal of the Royal Statistical Society. Series D (The Statistician)*, 25(1):37–42, 1976. URL https://doi.org/10.2307/2988129. [p312]

R. A. Peterson and J. E. Cavanaugh. Ordered quantile normalization: a semiparametric transformation built for the cross-validation era. *Journal of Applied Statistics*, pages 1–16, 2019. URL https://doi.org/10.1080/02664763.2019.1630372. [p310, 311, 312, 313, 323]

P. Royston. Estimating departure from normality. *Statistics in Medicine*, 10(8):1283–1293, 1991. doi: https://doi.org/10.1002/sim.4780100811. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/sim.4780100811. [p322]

H. C. Thode. *Testing for normality*, volume 164. CRC press, 2002. [p313]

B. Van der Waerden. Order tests for the two-sample problem and their power. In *Indagationes Mathematicae (Proceedings)*, volume 55, pages 453–458. Elsevier, 1952. [p311]

W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S*. Springer, New York, fourth edition, 2002. URL http://www.stats.ox.ac.uk/pub/MASS4. ISBN 0-387-95457-0. [p311]

H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. ISBN 978-3-319-24277-4. URL https://ggplot2.tidyverse.org. [p325]

H. Wickham and D. Seidel. *scales: Scale Functions for Visualization*, 2020. URL https://CRAN.R-project.org/package=scales. R package version 1.1.1. [p325]

S. N. Wood. Fast stable restricted maximum likelihood and marginal likelihood estimation of semiparametric generalized linear models. *Journal of the Royal Statistical Society (B)*, 73(1):3–36, 2011. [p325]

I. Yeo and R. A. Johnson. A new family of power transformations to improve normality or symmetry. *Biometrika*, 87(4):954–959, 2000. URL https://doi.org/10.1093/biomet/87.4.954. [p310, 311]

Ryan A. Peterson
Department of Biostatistics and Informatics
University of Colorado Anschutz Medical Campus 13001 East 17th Place Aurora, Colorado 80045
ORCID: 0000-0002-4650-5798
https://petersonr.github.io/
ryan.a.peterson@cuanschutz.edu

Package wsbackfit for Smooth Backfitting Estimation of Generalized Structured Models

by Javier Roca-Pardiñas, María Xosé Rodríguez-Álvarez and Stefan Sperlich

Abstract A package is introduced that provides the weighted smooth backfitting estimator for a large family of popular semiparametric regression models. This family is known as *generalized structured models*, comprising, for example, generalized varying coefficient model, generalized additive models, mixtures, potentially including parametric parts. The kernel-based weighted smooth backfitting belongs to the statistically most efficient procedures for this model class. Its asymptotic properties are well-understood thanks to the large body of literature about this estimator. The introduced weights allow for the inclusion of sampling weights, trimming, and efficient estimation under heteroscedasticity. Further options facilitate easy handling of aggregated data, prediction, and the presentation of estimation results. Cross-validation methods are provided which can be used for model and bandwidth selection.[1]

Introduction and brief review

The classes of generalized structured models (GSM) of Mammen and Nielsen (2003), structured additive regression models (Brezger et al., 2005), and semiparametric separable models (Rodríguez-Poó et al., 2003) are all devoted to harmonizing the fundamental aspects of flexibility, dimensionality and interpretability (c.f. also Stone, 1986) for multidimensional regression. In some cases, the particular structure is derived from pure theory, sometimes from empirical knowledge, or it is chosen data-adaptively. The epithet 'structured' underlines the explicit modeling of the structure of a regression in order to distinguish it from fully automatic black-box regression or prediction. Mammen and Nielsen (2003) define for response Y with covariate vectors $(\mathbf{Z}, \mathbf{X}, \mathbf{T}, \mathbf{U})$ the GSM class by

$$\Lambda(Y) = G\{\mathbf{Z}, \beta, g(\mathbf{X})\} + S\{\mathbf{T}, \delta, s(\mathbf{U})\}\,\epsilon = G\{\mathbf{Z}, \beta, g(\mathbf{X})\} + \varepsilon\ , \tag{1}$$

with Λ, G, S parametric known functions, β, δ unknown finite-dimensional parameter, $g(\cdot)$, $s(\cdot)$ unknown nonparametric functions, and ϵ, ε fulfilling $E[\epsilon|\mathbf{Z}, \mathbf{X}] = E[\varepsilon|\mathbf{Z}, \mathbf{X}] = 0$. While Λ is a transformation with potentially unknown parts which can be estimated along Linton et al. (2008), G and S are link functions that also determine further structures. For instance, for a partial linear varying coefficient model (Park et al., 2015) with $\mathbf{Z} = (Z_1, \cdots, Z_d, \mathbf{Z}_\kappa)$, $\mathbf{X} = (X_1, \cdot, X_d)$, where Z_1 to Z_d and X_1 to X_d are scalars, and $\mathbf{Z}_\kappa$ a vector of the length of β, function G defines

$$G\{\mathbf{Z}, \beta, g(\mathbf{X})\} = \underline{G}\{\eta(\mathbf{Z}, \beta, g(\mathbf{X}))\} = \underline{G}\left\{g_0 + \sum_{j=1}^{d} g_j(X_j)Z_j + \mathbf{Z}_\kappa^t\beta\right\}\ , \tag{2}$$

with index η and a known link function $\underline{G}$. You may also allow that some, or all of the X_j, $j = 1, \ldots, d$ are identical; the same holds for the Z_j, etc. Moreover, by setting $Z_j \equiv 1\ \forall j$ with all X_j being different, you obtain the generalized additive model (GAM). A detailed discussion on identifiability is provided in Lee et al. (2012).

Since Hastie and Tibshirani (1990) introduced their backfitting algorithm, additive models have become quite popular in statistics, particularly in biometrics, technometrics, and environmetrics. Opsomer and Ruppert (1997) and Opsomer (2000) derived asymptotic theory for that classical backfitting estimator with kernel smoothing. Mammen et al. (1999) developed asymptotic theory for a modified version, the smooth backfitting (SB) estimator, under weaker assumptions on the data like the allowance for strong correlation of the covariates. Mammen and Nielsen (2003) extended this method to the general GSM class (1), and Roca-Pardiñas and Sperlich (2010) proposed a common algorithm for it. Many extensions have been developed, procedures for bandwidth selection (e.g., Mammen and Park, 2005), quantile regression (Lee et al., 2010), and further asymptotic theory for particular cases (see e.g., Yu et al., 2008, for GAM). Most recent contributions extend SB to additive

[1]The second author acknowledges the support received by the Basque Government through the BERC 2018-2021 program and Elkartek project 3KIA (KK-2020/00049), by the Spanish Ministry of Science, Innovation and Universities through BCAM Severo Ochoa accreditation SEV-2017-0718 and through project MTM2017-82379-R funded by (AEI/FEDER, UE). The third author acknowledges financial support from the Swiss National Science Foundation, project 200021-192345.

inverse regression (Bissantz et al., 2016), proportional hazards (Hiabu et al., 2020), or regression with error-in-variables (Han and Park, 2018). All SB procedures and their theory are kernel-based.

The main advantage of SB is, apart from its excellent numerical performance proven by Nielsen and Sperlich (2005) and Roca-Pardiñas and Sperlich (2010), compared to the classical backfitting, that there exists a comprehensive literature that studies its statistical behavior and underlying assumptions. It provides the exact and complete asymptotic theory of SB, such that today this estimator is well understood. The only drawback has been that so far, there hardly existed an easily available software for this estimator, except the R-package **sBF** of Arcagni and Bagnato (2014) for the basic additive model. But due to its complexity, practitioners typically abstain from implementing it themselves. Therefore, the **wsbackfit** R-package has been developed which provides the weighted SB for all models listed in the next section, including a data-driven bandwidth selector. The package is freely available from the Comprehensive R Archive Network (CRAN) at `https://CRAN.R-project.org/package=wsbackfit` (R Core Team, 2016). Thus, this package closes the gap between the huge body of existing and still increasing literature about SB on the one side, and its potential use on the other, providing the necessary software. We hope it is soon extended by procedures for the various, partly above cited, extensions.

It is to be mentioned that there certainly exist R packages for alternative methods to estimate related models. Before briefly discussing some of the most advanced packages, let us mention the reviews of Binder and Tutz (2008), which reviewed spline-based methods, and Fahrmeier et al. (2004) which reviewed (spline-based) Bayesian methods.

Maybe the broadest set of models can be handled by the package **BayesX** (Umlauf et al., 2019). It embraces several well-known regression models such as GAM, generalized additive mixed models (GAMM), generalized geo-additive mixed models (GGAMM), dynamic models, varying coefficient models (VCM), and geographically weighted regression. Besides exponential family regression, **BayesX** also supports non-standard regression situations such as regression for categorical responses, hazard regression for continuous survival times, and continuous-time multi-state models; see also its support platform `http://www.uni-goettingen.de/de/bayesx`. It has been created by Brezger et al. (2005) and Kneib et al. (2008).

The R package **gam** (Hastie, 2019) presents considerable enhancements of the S-PLUS version going back to Hastie and Tibshirani (1990). It uses classical backfitting to combine different smoothing or fitting methods, particularly local regression and smoothing splines. Another powerful package is **mgcv** (Wood, 2017), which allows the fitting of generalized additive (mixed) models, with smoothing parameter estimation done by (restricted) marginal likelihood or generalized cross-validation, and uses iterated nested Laplace approximation for fully Bayesian inference. Another powerful and well-functioning package is **GAMLSS** of Stasinopoulos and Rigby (2007). It is based on penalized likelihood estimation combined with classical backfitting. While **mgcv** models the index function, **GAMLSS** models the location, scale, and shape functions by additive linear mixed models. It has been created to tackle many interesting distributions of Y. When speaking of likelihood based approaches, one should also mention a method introduced by Tutz and Binder (2006). Their R-package **GAMBoost** can be used to fit a GAM by likelihood based boosting, suited for a large number of predictors.

Regarding kernel-based methods that consider related or specific cases of (1), there are, for example, marginal integration (Linton and Nielsen, 1995) for additive interaction models (Sperlich et al., 2002), and local polynomials for smooth varying coefficients (Li and Racine, 2010). The latter is implemented in the **np** package (Hayfield and Racine, 2008), and turned out to be very competitive when compared to the before-mentioned spline-based packages (Sperlich and Theler, 2015).

The models that can be estimated by wsbackfit

The aim is to estimate a GSM as introduced in (1). In the moment of estimation, one has to be specific about Λ, G, and S. We concentrate on the popular cases, in particular on those that maintain additivity or a similar separability structure. This way, the estimates provide an easy interpretation, and overcome the curse of dimensionality, which else is inherited by more complex models. To the best of our knowledge, all existing smooth backfitting methods follow the suggestion of Mammen and Nielsen (2003) to estimate the mean and variance part subsequently, say, first $G\{\cdots\}$, then $S\{\cdots\}$. Our implementation follows the suggestion of Roca-Pardiñas and Sperlich (2010) to allow for a (re-)estimation of the mean part ($G\{\cdots\}$) including weights obtained from the estimation of the variance part ($S\{\cdots\}$) to potentially increase the efficiency. Note, however, that in nonparametrics, it is often not clear to what extent an efficiency gain can be achieved this way; see for example the discussion in Xiao et al. (2003). All proposed methods we are aware of,

are two- or three-step procedures similar to what we propose here. The estimation of the variance part is performed in the second step by regressing the squared residuals on $(\mathbf{T}, \mathbf{U})$, using the same procedure as for $G\{\cdots\}$. Therefore it is sufficient to concentrate in the following on the mean regression, which can equally well be applied to squared residuals for estimating $S\{\cdots\}$.

As said, the SB idea for GSM, together with some general results about its asymptotic behavior, was introduced by Mammen and Nielsen (2003); specific algorithms and their implementation were introduced and studied in Roca-Pardiñas and Sperlich (2010). Detailed information about the implementation is given in a technical report (Roca-Pardiñas and Sperlich, 2008). The implemented algorithms in **wsbackfit** are modified versions to speed up the procedure by binning techniques and a combination of parametric linear with local constant kernel regression; see below. The models considered in this package are semiparametric in the sense that they contain parametric as well as nonparametric components. Most of them could be seen as extensions of a generalized linear model (GLM) of type $G(\mathbf{Z}, \beta, g(\mathbf{X})) = \underline{G}\big(\eta(\mathbf{Z}, \beta, g(\mathbf{X}))\big) = \underline{G}(g_0 + \alpha^t\mathbf{X} + \beta^t\mathbf{Z})$, see for example McCullagh and Nelder (1989). So far, this package does not tackle random effects.

Regarding the choice of G, you have first to decide about the link $\underline{G}$. For each conditional distribution, there exists a canonical one: for the conditional Gaussian distribution, this is the identity. For a binary response, it is the Logit $(1 + 1/exp(\bullet))^{-1}$, and for a conditional Poisson it is $exp(\bullet)$. Note that the latter can also be used for Pseudo-Poisson estimation. The choice of $\underline{G}$ is certainly linked to the specification of Λ which is supposed to be known. Then, such transformation of Y can be performed a priori by the practitioner. Therefore, we henceforth suppress Λ, to simplify our notation. For Λ entailing unknown parameters, consult Linton et al. (2008).

Roca-Pardiñas and Sperlich (2010) showed that the estimation procedure for all these models can be summarized in one common feasible minimization problem, namely

$$minimize \quad \int \sum_{i=1}^{n} \left[\tilde{Y}_i - \eta\{\mathbf{Z}_i, \beta, g(\mathbf{x})\}\right]^2 W_i \cdot K_h(\mathbf{x} - \mathbf{X}_i)\, d\mathbf{x}\ , \tag{3}$$

where $\tilde{Y}_i$ is the transformed (e.g. by Λ) or linearized (in local scoring if the link is not the identity) response Y_i, and W_i is a weight. For example, in the generalized additive model with $\beta = 0$ we have covariates $Z_j \equiv 1$ for $j = 1, \ldots, d$, W_i contains the local scoring weights with $\tilde{Y}_i$ being the accordingly adjusted dependent variable. Further, $K_h(v) = h^{-1}K(v/h)$ with $K(\cdot)$ is the kernel function. It is well known that asymptotically, the choice of smoothing kernel does not have an important impact, as to a large part the kernel effect is compensated by an adequate bandwidth choice. We allow the user to choose between the Epanechnikov kernel which is asymptotically the most efficient one, and the Gaussian kernel which is popular as it helps to avoid some of the numerical problems that may arise in areas where data are sparse.

We call our procedure 'weighted smooth backfitting' to emphasize that the user has the option to include a vector of additional weights. As said, by putting the usual kernel weights apart, part of the weighting comes from local scoring in order to account for the link function $\underline{G}$.[2] However, independently from the link function, the practitioner might also want to include sampling weights, e.g., when using administrative data, or trimming weights, e.g., for excluding boundary points. A particular case is when additional weights are included to improve the efficiency of your estimators, e.g., to account for the (co-)variance structure. Roca-Pardiñas and Sperlich (2010) estimated in a first step the mean function, afterward the variance from the squared residuals, and used these in the third step as additional weights when re-estimating the conditional mean. The resulting average mean squared error was substantially smaller than the one of the original estimator, which ignored the (co-)variance structure; recall our discussion at the beginning of this section.

The models that package **wsbackfit** can presently estimate are: a partial linear GAM, a generalized partial linear varying coefficient model (GVCM), and combinations of them. The first one is a generalization of a GLM by replacing some linear components with additive nonparametric functions,

$$E\left[Y|\mathbf{X}, \mathbf{Z}\right] = \underline{G}\left(g_0 + \sum\nolimits_{j=1}^{d} g_j\left(X_j\right) + \beta^t\mathbf{Z}\right)\ ,$$

where X_1 to X_d are scalars, and $\mathbf{Z}$ is the vector of all covariates that are supposed to enter the index function η linearly. The g_j are nonparametric functions.[3] It is actually true that this is a special case of the partial linear GVCM of the form (2), obtained by setting $Z_1 = Z_2 = \cdots = Z_d = 1$. We list them nonetheless separately because, besides the slightly different implementation, we want the reader to recognize the difference in the modeling approach. First, the GVCM is a generalization of

[2] Yu et al. (2008) propose a somewhat different algorithm for GAMs replacing local scoring by an alternative that makes asymptotic theory simpler.

[3] For identification we follow Lee et al. (2012), and Roca-Pardiñas and Sperlich (2010) for implementation with modifications like the inclusion of a parametric linear slope for each g_j, see below.

a GLM in $\mathbf{Z}$, as could be seen in (2). And second, here we allow for all the flexibility suggested by Lee et al. (2012) regarding the alterations of covariates X_j and Z_j. For example, all $X_1, \ldots, X_{d_1}$ with $d_1 \leq d$ could be the same scalar variable X_1 such that

$$\sum\nolimits_{j=1}^{d_1} g_j\left(X_j\right) Z_j = \sum\nolimits_{j=1}^{d_1} \left\{ g_{j0} + g_{j1}\left(X_1\right) \right\} Z_j \ , \tag{4}$$

with g_{j0} unknown constants and g_{j1} unknown nonparametric functions; or, alternatively, all $Z_1, \ldots, Z_{d_1}$ with $d_1 \leq d$ could be the same scalar variable Z_1 such that (with g_{10} still a constant)

$$\sum\nolimits_{j=1}^{d_1} g_j\left(X_j\right) Z_j = \left\{ g_{10} + \sum\nolimits_{j=1}^{d_1} g_{j1}\left(X_j\right) \right\} Z_1 \ . \tag{5}$$

Certainly, you can have a mixture of both, as long as the identification conditions of Lee et al. (2012) are fulfilled to guarantee that the model does not suffer from concurvity. This includes the possibility that some variables appear in both sets, $\mathbf{X}$ and $\mathbf{Z}$. This could be of particular interest when defining different types of interactions.

Finally, one can include all together, i.e., nonparametric additive terms, nonparametric varying coefficients, and a parametric (linear) part like

$$E\left[Y|\mathbf{X},\mathbf{Z}\right] = \underline{G}\left\{ g_0 + \sum\nolimits_{j=1}^{d_1} g_j\left(X_j\right) Z_j + \sum\nolimits_{j=d_1+1}^{d} g_j\left(X_j\right) + \beta^t \mathbf{Z}_\kappa \right\} \ , \tag{6}$$

with $\mathbf{X}$ as before, $\mathbf{Z} = (Z_1, \ldots, Z_{d_1}, \mathbf{Z}_\kappa)$ a set of scalar variables $Z_1, \ldots, Z_{d_1}$, and a vector $\mathbf{Z}_\kappa$. Again, some of the X_j may represent the same variable; the same holds for the Z_j, $j = 1, ..., d$.

Cross-validation, bandwidths, and computational issues

Cross-validation (CV) can be used for model selection in general. However, for the sake of presentation we describe here our implementation in the context of bandwidth selection.

Cross-validation for bandwidth

All nonparametric estimates of the $g_j(X_j)$ in (6) depend on some bandwidths $h_1, \ldots, h_d$, which can be preset by the user. Alternatively, the package provides the option to choose the bandwidths data-adaptively via CV. Our implementation even allows for a mixture of both, i.e., users can fix some bandwidths and choose the others by CV. Albeit we use binning techniques, performing CV can render the program pretty slow, especially for high dimensions and huge data sets. Generally, our implementation follows the ideas of Nielsen and Sperlich (2005) and Roca-Pardiñas and Sperlich (2010). It is to be mentioned that several alternatives exist, in particular for the additive model. For instance, Mammen and Park (2005) proposed bandwidths selectors based on penalized least squares and plug-in approaches.

Given sample $\{\mathbf{X}_i, \mathbf{Z}_i, Y_i\}_{i=1}^n$, bandwidths $h_1, \ldots, h_d$ can be selected by minimizing some CV criterion in various ways. Allowing for limited dependent variables Y, the deviance is an appropriate measure of discrepancy between observed and fitted values. It is derived as a likelihood ratio test comparing the specified model with a so-called saturated one, when predicted values match the observed responses exactly. More specifically, denoting the fitted mean response given by $\hat{\mu}_i = \hat{E}\left[Y_i|\mathbf{X_i},\mathbf{Z_i}\right]$, the deviance is given by $\text{Dev} = \sum_{i=1}^n \text{Dev}_i(Y_i, \hat{\mu}_i)$. The definition of the individual deviance Dev_i depends on the link; namely

	$\text{Dev}_i(Y_i, \hat{\mu}_i)$
Gaussian	$\left(Y_i - \hat{\mu}_i\right)^2$
Binary	$-2\left(Y_i \log \hat{\mu}_i + (1 - Y_i) \log(1 - \hat{\mu}_i)\right)$
Poisson	$Y_i \log \frac{Y_i}{\hat{\mu}_i} - (Y_i - \hat{\mu}_i)$

Generally spoken, unless bandwidths are fixed by the user, they can be selected as

$$(h_1, \ldots, h_d) = \underset{(h_1^\bullet, \ldots, h_d^\bullet)}{\arg\min} \sum_{i=1}^{n} \text{Dev}_i \left[Y_i, \underline{G}\left(\hat{\eta}^{(-i)}_{\mathbf{X}_i,\mathbf{Z}_i} \right) \right], \tag{7}$$

with

$$\hat{\eta}^{(-i)}_{\mathbf{X}_i,\mathbf{Z}_i} = \hat{g}_0^{(-i)} + \sum\nolimits_{j=1}^{d_1} \hat{g}_j^{(-i)}\left(X_{ij}\right) Z_{ij} + \sum\nolimits_{j=d_1+1}^{d} \hat{g}_j^{(-i)}\left(X_{ij}\right) + \hat{\beta}^{t(-i)} \mathbf{Z}_{i\kappa} \ , \tag{8}$$

where $\hat{g}_j^{(-i)}(X_{ij})$ indicates the fit at X_{ij} leaving out the ith data point based on the smoothing parameter $h_j^\bullet$. One option to solve the minimization in (7) is to use a complete bandwidth selection that allows for all possible bandwidths combinations for the different covariates X_j. When the number of covariates X_j is large, the computational cost becomes very high or even prohibitive.

In order to simplify the problem, we provide the following three options to the user:

1. the user just prefixes all bandwidths that shall be used as final bandwidths;
2. the user prefixes starting values for the bandwidths, say $\tilde{h}_j$, and searches via CV for the optimal bandwidth vector $(h_1, \ldots, h_d)$ with a common bandwidth factor $c_h \in R$ such that $(h_1, \ldots, h_d) = c_h(\tilde{h}_1, \ldots, \tilde{h}_d)$;
3. the user only prefixes a bandwidth grid for a scalar h_c such that $(h_1, \ldots, h_d) = h_c(\sigma_1, \ldots, \sigma_d)$ with σ_j being the standard deviation of X_j, and h_c is chosen from the grid via CV.

The choice of prior $\tilde{h}_j$ typically follows some considerations of marginal distributions or marginal smoothing. For example, you could first perform a CV bandwidth choice for each nonparametric g_j by setting all other $g_{k\neq j}$ to zero, or by restricting them to be linear functions. The second method follows the ideas of Han and Park (2018), and the third follows a standard recommendation in the literature, see the review of Köhler et al. (2014). Combining options 1 and 3 is possible.

For the sake of presentation, we explain more details about the CV implementation only along option 3; as for option 2, it works analogously. Moreover, suppose the user chooses all bandwidths by option 3. As said, in option 3, $h_j = h_c\sigma_j$. While h_c might be different for each j if h_j is set by the user or chosen by option 2, in option 3, it is the same for all j. That is, we reduce the multidimensional search problem to a one-dimensional one. Specifically, if the user decides that all bandwidths are to be chosen by CV, $h_c := h_j/\sigma_j$ for all j, with

$$h_c = \arg\min_{h^\bullet} \sum_{i=1}^{n} \text{Dev}_i \left[Y_i, \underline{G}\left(\hat{\eta}^{(-i)}_{\mathbf{X}_i, \mathbf{Z}_i} \right) \right], \tag{9}$$

where $\eta^{(-i)}_{\mathbf{X}_i,\mathbf{Z}_i}$ indicates the fitted additive predictor at $\{\mathbf{X}_i, \mathbf{Z}_i\}$ (see (8)) leaving out the ith data point, and based on the smoothing parameters $h^\bullet\sigma_j$, $(j = 1, \ldots, d)$.

Unfortunately, a naive implementation of the leave-one-out CV technique would still imply a high computational cost as for each potential value of $h^\bullet$, it is necessary to repeat the estimation as many times as we have data points. To speed up the process, the **wsbackfit** package uses k-fold CV instead. In brief, k-fold CV consists of randomly splitting the available sample into k complementary subsamples of (approximately) the same size such that each data point only belongs to one of the k subsamples, say $\kappa(i)$. Then, the k-fold CV version of (9) is

$$h_c = \arg\min_{h^\bullet} \sum_{i=1}^{n} \text{Dev}_i \left[Y_i, \underline{G}\left(\hat{\eta}^{(-\kappa(i))}_{\mathbf{X}_i, \mathbf{Z}_i} \right) \right], \tag{10}$$

where $\eta^{(-\kappa(i))}_{\mathbf{X}_i,\mathbf{Z}_i}$ indicates the fitted additive predictor at $\{\mathbf{X}_i, \mathbf{Z}_i\}$ computed with the $\kappa(i)$ subsample removed. In contrast to leave-one-out CV, in k-fold CV, you repeat the estimations only k times, leaving-out one different subsample each time.

We conclude with two remarks. First, as said, the **wsbackfit** package also allows the user to specify the bandwidths h_j for some nonparametric functions, which are therefore treated as given in (10)), while letting the CV procedure select the others along option 3. A combination of option 2 with the others is not implemented. Examples can be found in Sections Package description and Examples and applications. Second, the minimum in (10) is determined by a grid search. The grid for the $h^\bullet$ (option 3) is by default `seq(0.01,0.99,length = 30)` but can optionally be set by the user via option `bw.grid`, see below. For option 2, the algorithm looks for the optimal c_h on an equispaced grid from 0.5 to 1.5.

Convergence Criteria

As explained above, smooth backfitting is solved by an iterative procedure to solve (3). When the link function is the identity, then there is only the loop running over the different additive components. If the link is not the identity, then there is also an outer loop carrying out the local scoring iteration. Then, within each of such outer iteration steps, the formerly mentioned loop of the smooth backfitting is conducted. Both loops are triggered by two factors, the tolerance *tol* in deviations between subsequent iterations and the maximum number *maxit* of iterations conducted. The defaults for the maximum number of iterations and the tolerance of deviation are $maxit = 10$ and $tol = 0.01$, respectively.

The inner loop stops when for iteration l, the updated $\hat{g}_j^l(\cdot)$ comply with criterion

$$\frac{\sum_{i=1}^n \left(\hat{g}_j^l(X_{ij}) - \hat{g}_j^{l-1}(X_{ij})\right)^2}{\sum_{i=1}^n \hat{g}_j^{l-1}(X_{ij})^2} \leq tol \quad \text{for} \quad j = 1, \ldots, d \ ,$$

where the g_j^{l-1} refer to the estimates of the previous iteration. Similarly, the outer loop stops when for iteration k, the convergence criterion

$$\text{Dev} = \frac{\sum_{i=1}^n \text{Dev}_i(Y_i, \hat{\mu}_i^{k-1}) - \text{Dev}_i(Y_i, \hat{\mu}_i^k)}{\sum_{i=1}^n \text{Dev}_i(Y_i, \hat{\mu}_i^{k-1})} \leq tol \ ,$$

is met, where $\hat{\mu}_i^k$ and $\hat{\mu}_i^{k-1}$ are the estimates of μ_i obtained in the present iteration and in the previous one $(k-1)$, respectively.

Binning and integration

Although we implemented some modifications and simplifications like the above described k-fold CV, or the combination of parametric linear with local constant estimation, for details see the Section Identification, SB in high dimensions might still imply a high computational cost. Therefore, as already indicated above, we implemented the kernel smoothing inside the **wsbackfit** package using binning-type procedures. These are used throughout, also for CV when selecting bandwidths. The key idea of this binning is to reduce the number of kernel evaluations (exploiting the fact that many of these are nearly identical) by replacing the original data set (composed of n data points) with a 'grouped' data set (with N groups as new data points with sampling weights, where $N << n$). The estimation is carried out on these N groups, including the sampling weights in W_i. For a detailed description of binning for kernel regression, see Fan and Gijbels (1996).

Note that for minimizing (3), we need to solve some univariate integrals over nonparametric estimates which is therefore done numerically. This is calculated by the Simpson rule with 51 equidistant grid points over the entire range of the respective covariate, i.e., from its smallest to the largest observation. Simulations showed that finer grids led to no improvement of the final estimates.

Identification

When we introduced the class of GSM in Section The models that can be estimated by wsbackfit, we restricted our presentation to models that can be estimated by the here introduced package. At this stage – note that the package design invites further contributions of SB-based methods – the package is able to estimate model (6) only for some link functions, and all g_j being one-dimensional nonparametric functions. Lee et al. (2012) discuss identification of model (6) in a very general way, also allowing all g_j to be multidimensional, and covariates X_j being overlapping vectors (i.e., containing, at least partly, the same elements), and possibly also containing some elements of the $\mathbf{Z}$ covariate vector. Essentially, they clarify which overlaps would render a model unidentifiable. As such discussion is quite technical and would go beyond the slope of this paper, we only refer to them.

Now recall models (4) and (5), which probably represent the most common cases in practice. For these models, consider the identification of the g_j with respect to location and scale. Each g_j in model (6) has been decomposed into a linear effect $\alpha_j \cdot X_j$ together with a purely nonparametric (beyond the linear) one, say $\tilde{g}_j$. In addition, for g_j being varying coefficients, we have included constants g_{j0}, $j = 1, \ldots, d_1$. Then, we can re-write model (6) as

$$\underline{G}\left\{g_0 + \sum\nolimits_{j=1}^{d_1} \left(g_{j0} + \alpha_j \cdot X_j + \tilde{g}_j\left(X_j\right)\right) Z_j + \sum\nolimits_{j=d_1+1}^{d} \left(\alpha_j \cdot X_j + \tilde{g}_j\left(X_j\right)\right) + \beta^t \mathbf{Z}_\kappa\right\}. \quad (11)$$

For this model, we set $E[\tilde{g}_j(X_j)] = 0$ $(j = 1, \ldots, d)$ and $E[X_j \cdot \tilde{g}_j(X_j)] = 0$ $(j = 1, \ldots, d_1)$. Apart from identification issues, this prevents us from biases in the linear direction, i.e., in the slope of the g_j, such that we can estimate the $\tilde{g}_j$ by local constant SB speeding up the algorithm significantly. Finally, this reparametrization helps us to see how to choose the starting values for the iterative backfitting procedure. For the first step, you can simply start with a parametric GLM estimator, setting $\tilde{g}_j \equiv 0$ for all j. Then, for $\hat{g}_0$, $\hat{\beta}$, $\hat{g}_{j,0}$, and $\hat{\alpha}_j$, $j = 1, \ldots, d$ obtained from that GLM estimation, you proceed estimating the $\tilde{g}_j$ as outlined in Roca-Pardiñas and Sperlich (2010), to afterward update all estimates via iteration.

Package description

The package is composed of several functions that enable users to fit the models with the methods described above. Table 1 provides a summary of the presently available functions.

wsbackfit functions	
`sback`	Main function for fitting generalized structures models using smooth backfitting.
`sb`	Function used to indicate the nonparametric terms and varying coefficient terms in the `sback()` formula.
`plot`	Function that provides plots of `sback` objects produced by `sback()`.
`print`	The default print method for an `sback()` object.
`summary`	Function that takes a fitted object produced by `sback()` and produces various useful summaries from it.
`predict`	Function that provides predictions (e.g., fitted values and nonparametric terms) based on an `sback` object for a new data set (`newdata`).
`residuals`	Returns residuals of `sback` objects produced by `sback()`. Deviance, Pearson, working and response residuals are available.
`summary`	Summary for objects of class `sback`.

Table 1: Summary of functions of **wsbackfit**.

The package has been designed similarly to other regression packages. The main function is `sback`. It fits GSM with SB, and creates an object of class `sback`. Numerical and graphical summaries of the fitted model can be obtained by using `print`, `summary`, and `plot`, implemented for `sback` objects. Moreover, function `predict` allows obtaining predictions (e.g., fitted values and nonparametric terms) for data different from those used for estimation, called, therefore, `newdata`. The main arguments of the `sback` function are listed in Table 2, and the list of outputs is given in Table 3.

`sback` arguments – input values	
`formula`	A `formula` object specifying the model to be fitted.
`data`	Data frame representing the data and containing all needed variables.
`offset`	An optional numerical vector containing a priori known components to be included in the linear predictor during fitting. Default is zero.
`weights`	An optional numeric vector of 'prior weights' to be used in the fitting process. By default, the weights are set to one.
`kernel`	A character specifying the kernel function. Implemented are: Gaussian and Epanechnikov. By default `"Gaussian"`.
`bw.grid`	Numeric vector; a grid for searching the bandwidth factor h_c. The bandwidth for dimension j is $h_c\sigma_j$. Default is `seq(0.01,0.99,length = 30)`
`c.bw.factor`	logical; indicates whether the common factor scheme for bandwidth selection proposed by Han and Park (2018) is performed. If `TRUE`, and provided the user has specified the (marginal) bandwidths for all nonparametric functions, say $\tilde{h}_j$, the functions searches for the common factor c_h that minimizes the deviance via (k-fold) cross-validation when the bandwidth used for dimension (covariate) j is $c_h\tilde{h}_j$. The search is done in an equispaced grid of length 15 between 0.5 and 1.5. The default is `FALSE`.
`KfoldCV`	Number of cross-validation folds to be used for either (1) automatically selecting the optimal bandwidth (in the sequence given in argument `bw.grid`) for each nonparametric function; or (2) automatically selecting the optimal common bandwidth factor (see argument `c.bw.factor`). Default is 5.
`kbin`	An integer value specifying the number of binning knots. Default is 30.
`family`	A character specifying the distribution family. Implemented are: Gaussian, Binomial and Poisson. In all cases, the link function is the canonical one. By default `"gaussian"`.

Table 2: Summary of the arguments of the main function `sback`.

We call function `sback` by

sback output values	
call	The matched call.
formula	The original supplied formula argument.
data	The original supplied data argument.
weights	The original supplied weights argument.
offset	The original supplied offset argument.
kernel	The original supplied kernel argument.
kbin	The original supplied kbin argument.
family	The original supplied family argument.
effects	Matrix with the estimated nonparametric functions (only the nonlinear component) for each covariate value in the original supplied data.
fitted.values	A numeric vector with the fitted values for the supplied data.
residuals	A numeric vector with the deviance residuals for the supplied data.
h	A numeric vector of the same length as the number of nonparametric functions, with the bandwidths used to fit the model.
coeff	A numeric vector with the estimated regression coefficients. This vector contains the estimates of the regression coefficients associated with the parametric part of the model (if present) as well as the linear components of the nonparametric functions.
err.CV	Matrix with the cross-validated error (deviance) associated with the sequence of tested bandwidths (those provided in argument bw.grid in function sback).

Table 3: Summary of output values of the sback function.

```
sback(formula, data, offset = NULL, weights = NULL,
  kernel = c("Gaussian", "Epanechnikov"),
  bw.grid = seq(0.01, 0.99, length = 30), c.bw.factor = FALSE,
  KfoldCV = 5, kbin = 30,
  family = c("gaussian", "binomial", "poisson"))
```

The argument formula corresponds to the model for the conditional mean function (6). This formula is similar to that used for glm, except that nonparametric functions can be added to the additive predictor by means of function sb (for details, see Table 4). For instance, specification y ~ x1 + sb(x2,h = -1) assumes a parametric effect of x1 (with x1 either numerical or categorical), and a nonparametric effect of x2. Varying coefficient terms get incorporated similarly. For example, y ~ sb(x1,by = x2) indicates that the coefficient(s) of x2 depend nonparametrically on x1. In this case, both, x1 and x2 should be numerical predictors. More examples are provided further below.

sb arguments	
x1	The univariate predictor.
by	Numeric predictor of the same dimension as x1. If present, the coefficients of this predictor depend nonparametrically on x1, i.e., a varying coefficient term.
h	Bandwidth for this term. If h = -1, the bandwidth is automatically selected using k-fold cross-validation. A value of 0 would indicate a linear fit. By default -1.

Table 4: Summary of the arguments of the function sb.

The bandwidths associated with the nonparametric functions are specified inside sb through argument h (see Table 4), either as final bandwidth (by setting h = h_j; option 1), as starting value to be multiplied by an optimal constant c_h found via CV from an equispaced grid of length 15 between 0.5 and 1.5 (by setting h = $\tilde{h}_j$ and c.bw.factor = TRUE; option 2), or by demanding a CV-bandwidth which is the product of a common factor h_c (chosen from bw.grid) times the scale σ_j of covariate X_j (by setting h = -1; option 3); recall Section Cross-validation, bandwidths, and computational issues. Actually, the user has even four options. These are specified through argument h of function sb, where option 4 is a mixture of options 1 and 3 by setting h = h_j inside sb for some nonparametric functions, and h = -1 for the others. The number k of CV folds is specified through KfoldCV, with 5 being the default.

In Section The models that can be estimated by wsbackfit, recall expression (3) with subsequent

discussion, we saw that the SB algorithm is implemented with potentially adjusted dependent variable $\tilde{Y}$ and weights W. The arguments `offset` and `weights` allow the user to modify them on top of what is done automatically (e.g., due to the local scoring). More specifically, the vector `offset` is subtracted from $\tilde{Y}$ when estimating. A standard application is the log of exposure in Poisson regression; see also the example in Section Poisson regression with offset. Regarding the weight W, recall that parts of it are automatically calculated (and updated in each backfitting iteration) by the local scoring procedure to account for the given link function, which is 1 if the link is the identity. This is multiplied by the optional vector `weights` provided by the user. For an example in which this option is used to improve the efficiency of the estimators, see Section Gaussian simulated data.

The desired smoothing kernel, either Epanechnikov or Gaussian, is specified through the argument `kernel` (by default Gaussian). With `kbin`, the user indicates the number of binning knots (denoted as N in Section Cross-validation, bandwidths, and computational issues). The argument `family` specifies the conditional distribution of the response variable. So far, the user can select among Gaussian, Poisson, and binary. In all cases, the canonical link function is considered. Finally, recall that predictions for data different from those being used for estimation can be obtained by means of function `predict`, specifying the new dataset in argument `newdata`.

Simulation study

This section reports the results of a small simulation. Two different scenarios are considered, namely

Scenario I. Additive model. Covariates X_1 and X_2 are simulated independently from the uniform distributions on the intervals $[0, 1]$ and $[-10, 1.5]$, respectively, and

$$\eta = g_1(X_1) + g_2(X_2) = 2 + 3X_1^2 + 0.01X_2^3.$$

Here, Y is generated under two different distributions

- $Y = \eta + \varepsilon$, where $\varepsilon \sim N\left(0, 0.5^2\right)$.
- $Y \sim \text{Bernoulli}(p)$, with $p = \exp(\widetilde{\eta}) / \exp(1 + \widetilde{\eta})$, where $\widetilde{\eta} = \eta/4$,

where the scaling factor in the Bernoulli case is used to control the signal-to-noise ratio.

Scenario II. Varying coefficient model. Here, covariates X_1, X_2, Z_1 and Z_2 are simulated independently from a uniform distribution on the interval $[0, 1]$, and

$$\eta = g_1(X_1) Z_1 + g_2(X_2) Z_2 = 5\sin(2\pi X_1) Z_1 + X_2 Z_2.$$

As for Scenario I, Y is generated according to

- $Y = \eta + \varepsilon$, where $\varepsilon \sim N\left(0, 0.5^2\right)$.
- $Y \sim \text{Bernoulli}(p)$, with $p = \exp(\eta) / \exp(1 + \eta)$.

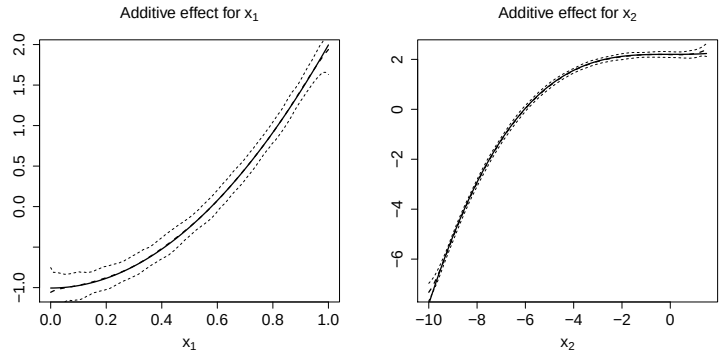
Results for Scenarios I and II are shown in Figure 1 and Figure 2, respectively. In both cases, the true and estimated functions are centered before plotting to make results comparable. All results are based on a sample size of $n = 500$ with $R = 500$ replicates. Also, we use the Gaussian kernel, 30 binning knots, and all bandwidths being selected using 5-fold cross-validation (option 3). We obtained essentially the same figures when we repeated these simulations with the Epanechnikov kernel. Not surprisingly, simulation results with options 1 and 2 for the bandwidth choice depended quite a bit on the setting of our (prior) bandwidths and are therefore not shown.

Examples and applications

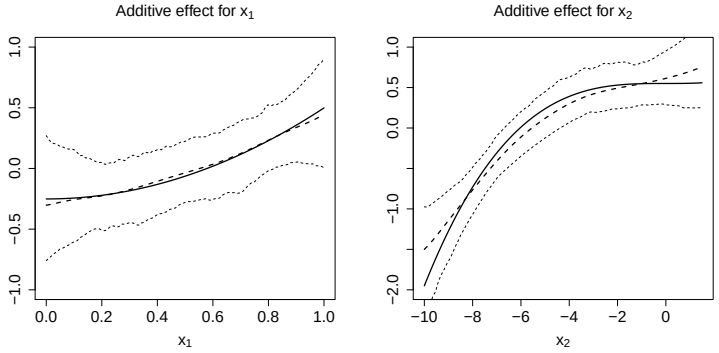
The last section is dedicated to examples with generalized additive and/or varying coefficient, partial linear models with and without heteroscedasticity, given different link functions. In fact, we have examples for each of the three link functions presently available. Among other things, it is shown how the optional weighting can be used to improve efficiency. While some examples are simulated, others illustrate applications from biometrics and health. Finally, we also show how to create useful graphics for interpreting the estimates.

Gaussian simulated data

We start with the presentation of a simulation example for estimating an additive model under heteroscedasticity. Consider the situation where the variance is a function of a dummy variable,



(a) Gaussian



(b) Bernoulli

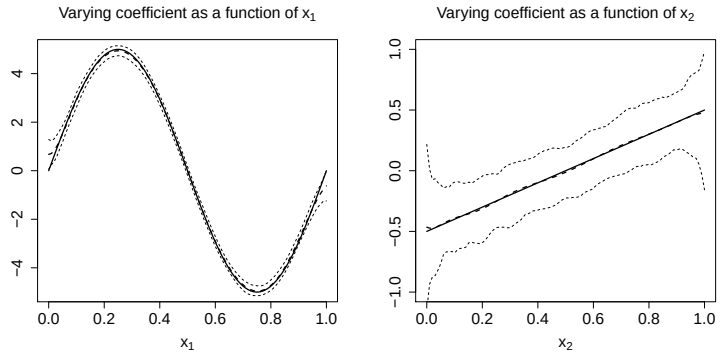
Figure 1: The simulation study and Scenario I. From left to right: true curve g_1 (solid line) and average estimate $\hat{g}_1$ (dashed line); true curve g_2 (solid line) and average estimate $\hat{g}_2$ (dashed line). In both cases, 2.5 and 97.5 simulation quantiles are plotted. Top row: Gaussian distribution. Bottom row: Bernoulli distribution.

i.e., one faces two noise levels. This is estimated and afterward used to improve the efficiency of the mean regression. As explained in the above sections, this requires a three-step procedure: first estimate the mean model, then the variance function, and finally re-estimate the mean model but including the inverse of the variance as an additional weight. Consider model

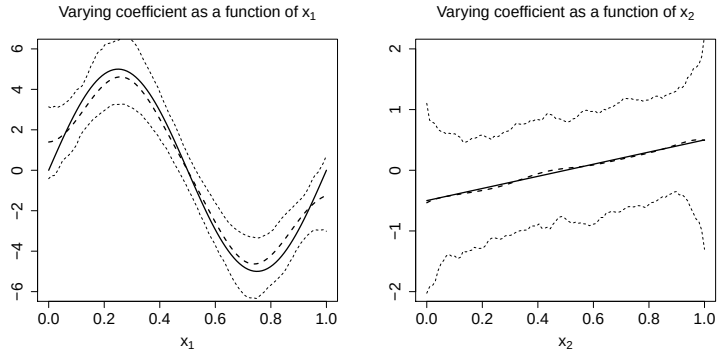
$$Y = \sum_{j=1}^{4} g_j(X_j) + \beta \mathbf{1}_{\{X_5=1\}} + \varepsilon(X_5), \tag{12}$$

with $g_1(x) = 2\sin(2x)$, $g_2(x) = x^2$, $g_3(x) = 0$, $g_4(x) = x$, $\beta = 1.5$, and $\mathbf{1}_A$ denoting the indicator function of event A. The covariates X_1 to X_4 are independent random variables, uniformly distributed on $[-2, 2]$, and $X_5 \in Bernoulli(0.4)$. The error term is given by $\epsilon(X_5) \in N(0, \sigma^2(X_5))$ with $\sigma(0) = 4$ and $\sigma(1) = 2$. Data are generated and fitted by

```
R> library(wsbackfit)
R> set.seed(123)
R> # Define the data generating process
R> n <- 1000
R> x1 <- runif(n)*4-2
R> x2 <- runif(n)*4-2
R> x3 <- runif(n)*4-2
R> x4 <- runif(n)*4-2
R> x5 <- as.numeric(runif(n)>0.6)

R> g1 <- 2*sin(2*x1)
R> g2 <- x2^2
R> g3 <- 0
R> g4 <- x4

R> mu <- g1 + g2 + g3 + g4 + 1.5*x5
```



(a) Gaussian



(b) Bernoulli

Figure 2: The simulation study and Scenario II. From left to right: true curve g_1 (solid line) and average estimate $\hat{g}_1$ (dashed line); true curve g_2 (solid line) and average estimate $\hat{g}_2$ (dashed line). In both cases, 2.5 and 97.5 simulation quantiles are plotted. Top row: Gaussian distribution. Bottom row: Bernoulli distribution.

```
R> err <- (0.5 + 0.5*x5)*rnorm(n)
R> y <- mu + err

R>  df_gauss <- data.frame(x1 = x1, x2 = x2, x3 = x3, x4 = x4, x5 = as.factor(x5), y = y)

R> # Fit the model with a fixed bandwidth for each covariate
R> m0 <- sback(formula = y ~ x5 + sb(x1, h = 0.1) + sb(x2, h = 0.1) +
+    sb(x3, h = 0.1) + sb(x4, h = 0.1), kbin = 30, data = df_gauss)
```

A numerical summary of the fitted model can be obtained by calling `print.sback()` or `summary.sback()` with shortcuts `print()` and `summary()`.

```
R> summary(m0)

Generalized Smooth Backfitting/wsbackfit:

Call: sback(formula = y ~ x5 + sb(x1, h = 0.1) + sb(x2, h = 0.1) +
    sb(x3, h = 0.1) + sb(x4, h = 0.1), data = df_gauss, kbin = 30)

Sample size: 1000

Bandwidths used in model:
 Effect          h
 sb(x1, h = 0.1) 0.1
 sb(x2, h = 0.1) 0.1
 sb(x3, h = 0.1) 0.1
 sb(x4, h = 0.1) 0.1

Linear/Parametric components:
```

```
   Intercept           x1           x2           x3           x4          x51
1.342678178  0.346506090 -0.040989607 -0.005250654  1.010634908  1.327833794
```

The output obtained from `summary` (corresponding to the prior `sback` call) includes the bandwidth of each nonparametric function, the parameters of the parametric part (here the intercept and β), and the linear slopes (i.e., the α_j in (11) in Section Identification) of the nonparametric functions g_j. Recall (Section Identification) that the algorithm decomposes each nonparametric function in a linear and a nonparametric local constant one. For a varying coefficient g_j, you also get constant g_{0j}.

To complement the numerical results, the **wsbackfit** package also provides graphical outputs by the use of `plot`. In particular, it provides the plots of the estimated nonparametric functions. Figure 3 shows the figures that appear as a result of the following code. We note that through argument `select`, the user can specify the model term to be plotted and use `ylim` to indicate the range for the y-axis. This, however, is optional. Alternatively, the program provides plots that automatically explore the variation of the estimates.

```
R> op <- par(mfrow = c(2,2))
R> plot(m0, select = 1, ylim = c(-2.5,2.5))
R> plot(m0, select = 2, ylim = c(-2.5,2.5))
R> plot(m0, select = 3, ylim = c(-2.5,2.5))
R> plot(m0, select = 4, ylim = c(-2.5,2.5))
R> par(op)
```

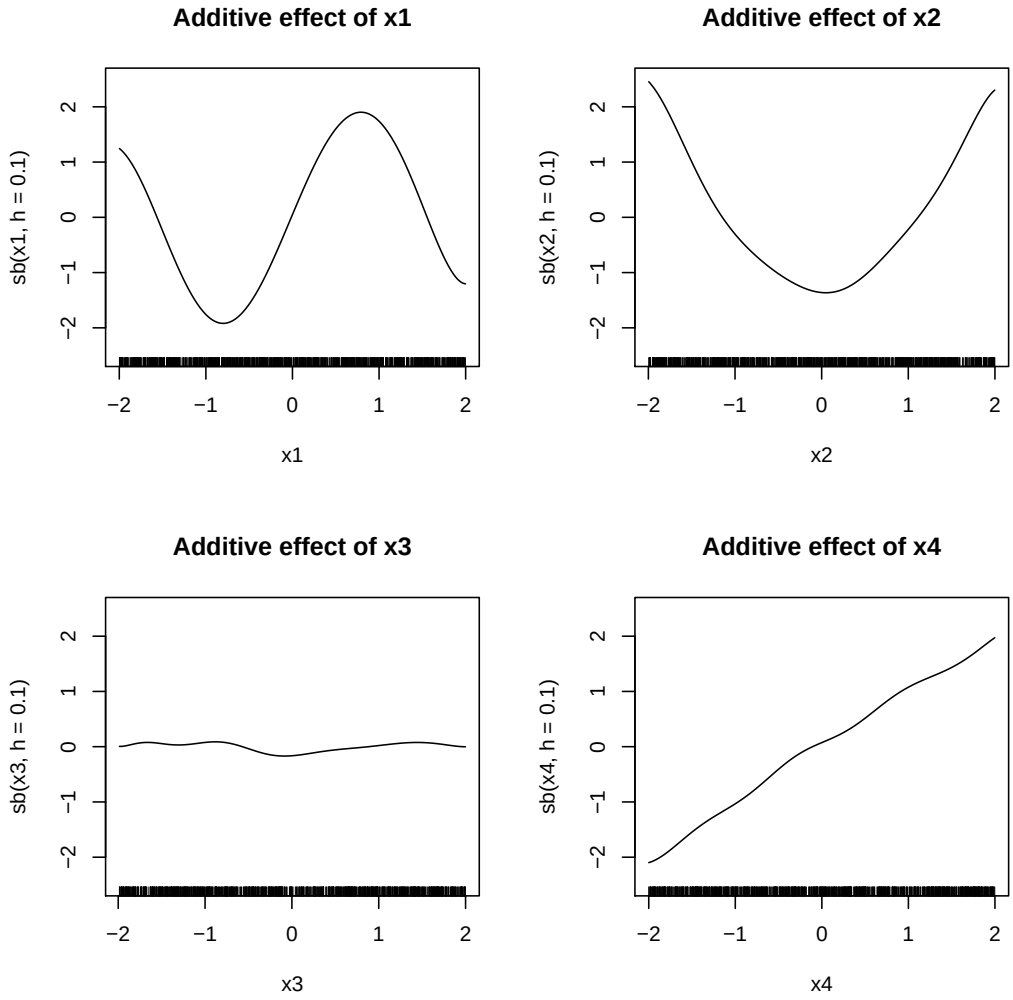


Figure 3: Model (12): Estimated nonparametric functions. These estimates correspond to the sum of the linear and the nonparametric local constant component, recall (11).

If the user is interested in plotting separately each component (α_j and $\tilde{g}_j$), then the argument `composed` is to be set to `FALSE`. The result is shown in Figure 4.

```
R> op <- par(mfrow = c(2,2))
R> plot(m0, select = 1, composed = FALSE, ylim = c(-2.5,2.5))
R> plot(m0, select = 2, composed = FALSE, ylim = c(-2.5,2.5))
R> plot(m0, select = 3, composed = FALSE, ylim = c(-2.5,2.5))
R> plot(m0, select = 4, composed = FALSE, ylim = c(-2.5,2.5))
R> par(op)
```

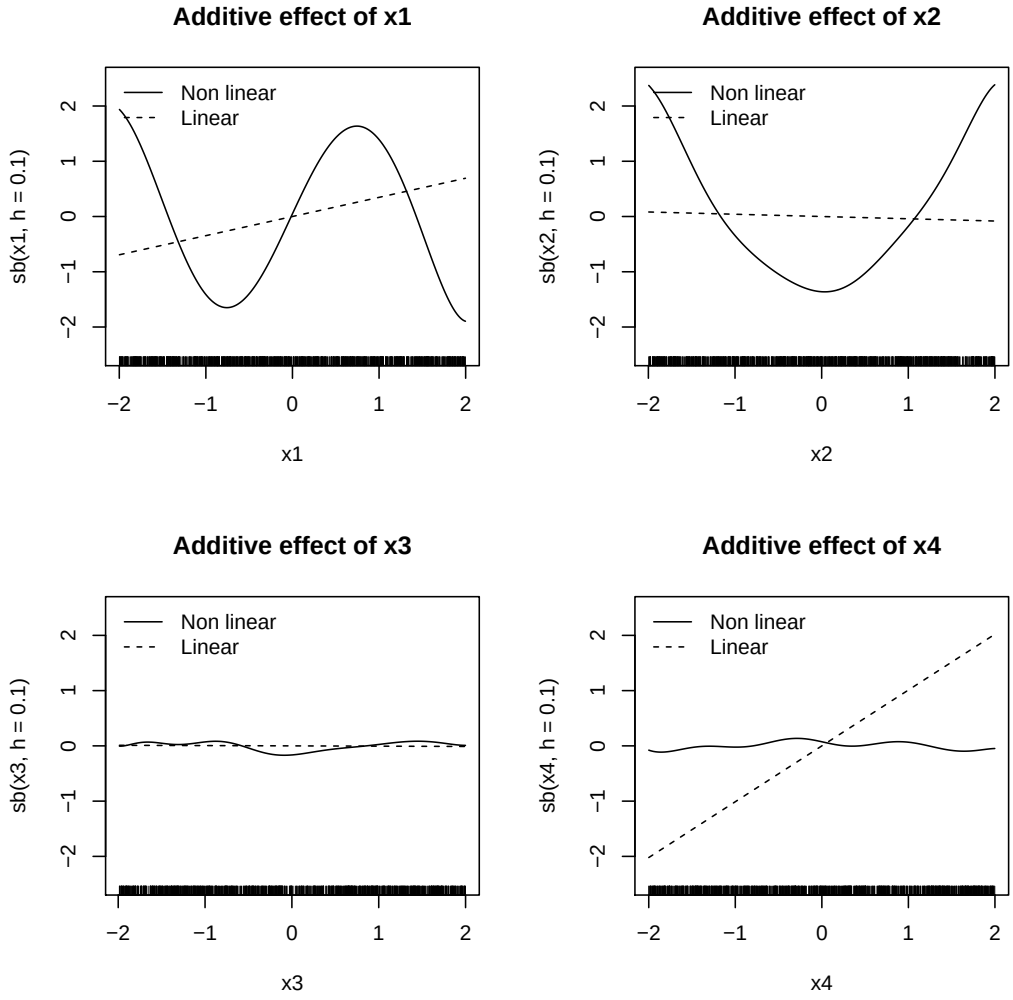


Figure 4: Model (12): Estimated linear and nonlinear components in which each nonparametric function is decomposed, recall (11).

Note that both `summary` and `plot` make use of the information contained in the `m0` object.

```
R> names(m0)

 [1] "call"          "formula"       "data"          "weights"
 [5] "offset"        "kernel"        "kbin"          "family"
 [9] "effects"       "fitted.values" "residuals"     "h"
[13] "coeff"         "err.CV"
```

This is the list of outputs created by `sback`. A detailed description of what each component of this list contains was given in Table 3. The user can access this information explicitly and individually, may it be to create its own plots or for further reporting.

As a next step in the analyses of our example, we use the fitted model to estimate the variance. In our example, this is considered to be a function of the binary covariate X_5. Call

```
R> resid <- y - m0$fitted.values
R> sig0 <- var(resid[x5 == 0])
R> sig1 <- var(resid[x5 == 1])
```

The third and final step is to re-estimate the mean model for efficiency reasons with weights that are the inverse of the estimated variance. The code, including the summary, is

```
R>  w <- x5/sig1 + (1-x5)/sig0
R>  m1 <- sback(formula = y ~ x5 + sb(x1, h = 0.1) + sb(x2, h = 0.1) +
+         sb(x3, h = 0.1) + sb(x4, h = 0.1),
+        weights = w, kbin = 30, data = df_gauss)
R> summary(m1)

Generalized Smooth Backfitting/wsbackfit:

Call: sback(formula = y ~ x5 + sb(x1, h = 0.1) + sb(x2, h = 0.1) +
    sb(x3, h = 0.1) + sb(x4, h = 0.1), data = df_gauss, weights = w,
    kbin = 30)

Sample size: 1000

Bandwidths used in model:
 Effect          h
 sb(x1, h = 0.1) 0.1
 sb(x2, h = 0.1) 0.1
 sb(x3, h = 0.1) 0.1
 sb(x4, h = 0.1) 0.1

Linear/Parametric components:
  Intercept          x1          x2          x3          x4         x51
 1.31707760  0.32888538 -0.01262394  0.01222234  1.00289877  1.33368035
```

In the previous fits of this example, we specified all bandwidths used. For the rest of this example, let us consider the case when we ask the program to choose the bandwidths via k-fold CV. We do this for all nonparametric functions, using the following code in which, for the sake of clarity and presentation, we specify `h = -1` although this is actually the default. For convenience, we also call the `summary` command directly:

```
R> m1cv <- sback(formula = y ~ x5 + sb(x1, h = -1) + sb(x2, h = -1) +
+   sb(x3, h = -1) + sb(x4, h = -1), weights = w, kbin = 30,
+   bw.grid = seq(0.01, 0.99, length = 30), KfoldCV = 5, data = df_gauss)
R> summary(m1cv)

Generalized Smooth Backfitting/wsbackfit:

Call: sback(formula = y ~ x5 + sb(x1, h = -1) + sb(x2, h = -1) + sb(x3,
    h = -1) + sb(x4, h = -1), data = df_gauss, weights = w, bw.grid = seq(0.01,
    0.99, length = 30), KfoldCV = 5, kbin = 30)

Sample size: 1000

Bandwidths used in model:
 Effect             h
 sb(x1, h = 0.0892) 0.0892
 sb(x2, h = 0.0887) 0.0887
 sb(x3, h = 0.0907) 0.0907
 sb(x4, h = 0.0912) 0.0912

Linear/Parametric components:
  Intercept          x1          x2          x3          x4         x51
 1.31708207  0.32881191 -0.01219359  0.01247328  1.00258427  1.33366258
```

We do not further discuss the results because their interpretation is the same as before, also because the automatically found data-driven optimal bandwidths are close to what we used as prefixed bandwidths in the former codes. We conclude this section with a brief example in which we specify the bandwidths for some of the nonparametric functions, while for the remaining ones, we let our CV procedure select the bandwidths. For brevity, we skip output and discussion.

```
R> m2cv <- sback(formula = y ~ x5 + sb(x1, h = 0.1) + sb(x2, h = -1) +
+         sb(x3, h = 0.1) + sb(x4, h = 0.1),
+        weights = w, kbin = 30, KfoldCV = 5, data = df_gauss)
```

Post-operative infection data

The next example is an application with data studied, among others, in Roca-Pardiñas and Sperlich (2010). They were taken from a prospective analysis conducted at the University Hospital of Santiago de Compostela in the North of Spain. A total of $n = 2318$ patients who underwent surgery at this center between January 1996 and March 1997 were considered. The main interest is learning about indicators that could predict whether patients may suffer (`inf=1`) or not post-operative infection (`inf=0`), and to see how they relate to the risk of infection. Such predictive indicators could be various, but given the previous studies we concentrate on the pre-operative values of plasma glucose (`gluc`) concentration (measured in mg/dl), and lymphocytes (`linf`, expressed as relative counts (in % of the white blood cell count). The data can be found in **wsbackfit** under the name `infect`.

```
R> data(infect)
R> head(infect)

 age sex linf gluc diab inf
1  85   2   28   55    2   0
2  38   1   18   56    2   1
3  49   2   29   56    2   1
4  63   2   20   60    2   0
5  91   2   17   62    2   0
6  26   2   22   66    2   0
```

In the original studies, it was controlled for other covariates like `age` (in years) and `sex` (coded as 1 = male; 0 = female). For illustrative purposes, we limit our analysis to the investigation of the association of the risk of post-operative infections `inf` with the predictors `linf` and `gluc`, putting all other covariates aside. It is well known that the effect of `linf` on `inf` varies strongly with the concentration of `gluc`. Therefore, one may think of a generalized varying coefficient model of type

$$\begin{aligned}\log \frac{P\left(\text{inf}=1|\text{linf},\text{gluc}\right)}{1-P\left(\text{inf}=1|\text{linf},\text{gluc}\right)} &= g_0+g_1(\text{gluc})+g_2(\text{gluc})\text{linf}\\ &= g_0+(\alpha_1\cdot\text{gluc}+\tilde{g}_1(\text{gluc}))+(g_{20}+\alpha_2\cdot\text{gluc}+\tilde{g}_2(\text{gluc}))\,\text{linf},\end{aligned} \tag{13}$$

in which we are working with the Logit link. This can be fitted using of the following code

```
R> data(infect)
R> # Generalized varying coefficient model with binary response
R> m2 <- sback(formula = inf ~ sb(gluc, h = 10) + sb(gluc, by = linf, h = 10),
+   data = infect, kbin = 15, family = "binomial")

R> summary(m2)

Generalized Smooth Backfitting/wsbackfit:

Call: sback(formula = inf ~ sb(gluc, h = 10) + sb(gluc, by = linf,
    h = 10), data = infect, kbin = 15, family = "binomial")

Sample size: 2312

Bandwidths used in model:
 Effect                       h
 sb(gluc, h = 10)             10
 sb(gluc, by = linf, h = 10) 10

Linear/Parametric components:
    Intercept          gluc          linf     gluc:linf
-1.4155401353  0.0068313875 -0.0346648080 -0.0000456441
```

Note that this model, recall (13), contains, in addition to the constant term (intercept), the main linear effects of `gluc` α_1 and `linf` g_{20}, and the linear interaction between `gluc` and `linf` α_2, all provided in the very last line. Our bandwidths have been chosen for graphical convenience. The graphical output, i.e., the plots of the estimated nonparametric functions, is obtained by the code

```
R> op <- par(mfrow = c(1,3))
R> plot(m2, composed = FALSE, ask = FALSE, cex.main = 2, cex = 2, cex.lab = 1.5,
+    cex.axis = 2)
```

```
R> par(op)

R> op <- par(mfrow = c(1,3))
R> plot(m2, composed = FALSE, ask = FALSE, cex.main = 2, cex = 2, cex.lab = 1.5,
+    cex.axis = 2)
R> par(op)
```

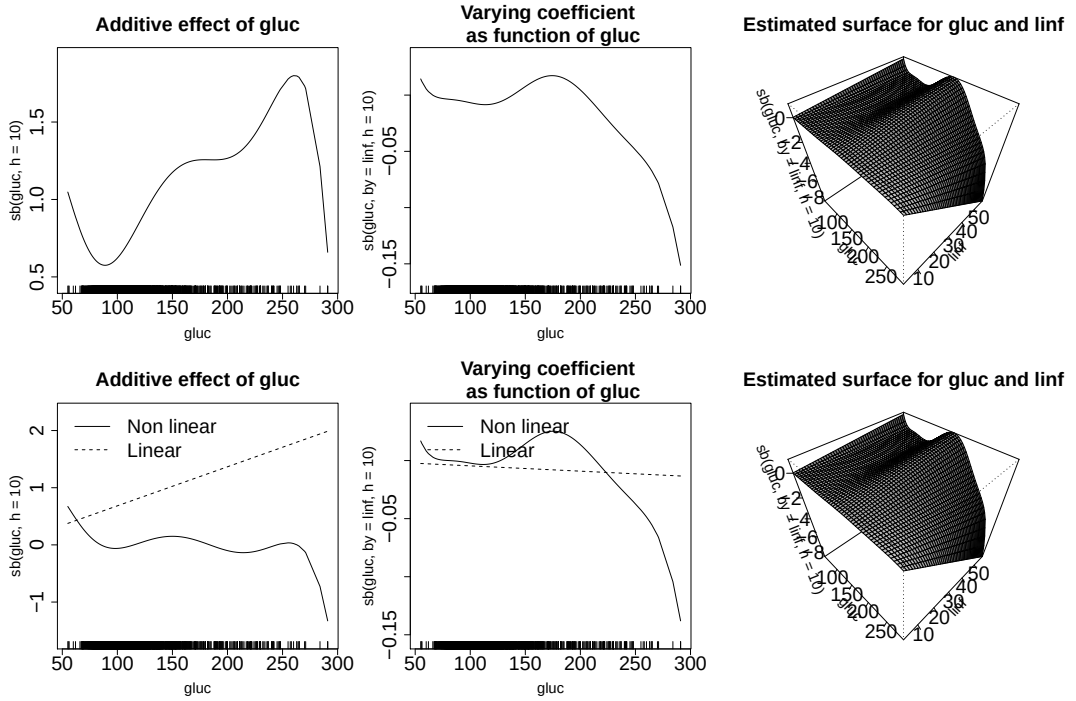


Figure 5: Post-operative infection data. Upper row: estimates of $\alpha_1 \cdot gluc + \tilde{g}_1(gluc)$ (left), $\alpha_2 \cdot gluc + \tilde{g}_2(gluc)$ (middle), and $(\alpha_2 \cdot gluc + \tilde{g}_2(gluc))\, linf$ (right) obtained from model (13). The plots on the left and the center in the bottom line show the estimates of the linear and nonlinear components separately. In this example, the right plot is simply repeated.

In Figure 5, you see the functionals of the nonparametric additive effect of `gluc` on the index η (left column), the varying coefficient (center), and the interaction surface (right column), because the interest is in revealing how the effect of lymphocytes changes with the plasma glucose concentration. If the interest is also in knowing the resulting probabilities of post-operational infection, then there are the options of plotting the two-dimensional function as a (dynamic) 3-D plot (less appropriate for printed figures) or by contour plots as done in Figure 6. This was created with the code

```
R> # Dataframe for prediction (and plotting)
R> ngrid <- 30
R> gluc0 <- seq(50,190, length.out=ngrid)
R> linf0 <- seq(0,45, length.out=ngrid)
R> infect_pred  <- expand.grid(gluc = gluc0, linf = linf0)

R> m2p <- predict(m2, newdata = infect_pred)
R> n <- sqrt(nrow(infect_pred))
R> Z <- matrix(m2p$pfitted.values, n, n)
R > filled.contour(z = Z, x = gluc0, y = linf0,
+ xlab = "Glucose (mg/dl)", ylab = "Lymphocytes (%)",
+ col = cm.colors(21))
```

As can be seen from Figure 6, high levels of `gluc` increase the post-operative infection risk, but higher `linf` values can mitigate this effect significantly.

Poisson regression with offset

Let us now consider a simulated example that illustrates the use of Poisson regression with a nontrivial use of option `offset`. We simulate data where each subject may have different levels of

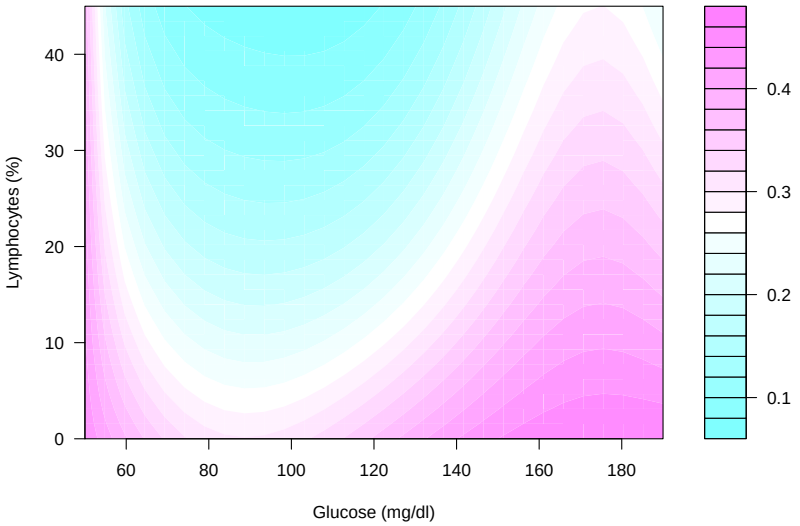


Figure 6: The post-operative infection data: Estimated probability of post-operational infection.

exposure to the event of interest. As explained in the above sections, this can be handled with the `offset` option. More specifically, for the level of exposure P we consider

$$Y \in Poisson\left(P \cdot \exp\left(2 + 3X_1^2 + 5X_2^3\right)\right) \ .$$

In our simulations, X_1 and X_2 were generated as independent continuous random variables uniformly distributed on $[-1, 1]$, and P as an approximately uniformly distributed discrete variable with support $\{50, 51, \ldots, 100\}$. The complete code for simulation, estimation, and summary of results is

```
R> set.seed(123)
R> # Generate the data
R> n <- 1000
R> x1 <- runif(n,-1,1)
R> x2 <- runif(n,-1,1)
R> eta <- 2 + 3*x1^2 + 5*x2^3
R> exposure <- round(runif(n, 50, 500))
R> y <- rpois(n, exposure*exp(eta))
R> df_poiss <- data.frame(y = y, x1 = x1, x2 = x2)
R> # Fit the model
R> m4 <- sback(formula = y ~ sb(x1, h = 0.1) + sb(x2, h = 0.1),
+        data = df_poiss, offset = log(exposure),
+        kbin = 30, family = "poisson")

R> summary(m4)

Generalized Smooth Backfitting/wsbackfit:

Call: sback(formula = y ~ sb(x1, h = 0.1) + sb(x2, h = 0.1), data = df_poiss,
    offset = log(exposure), kbin = 30, family = "poisson")

Sample size: 1000

Bandwidths used in model:
 Effect          h
 sb(x1, h = 0.1) 0.1
 sb(x2, h = 0.1) 0.1

Linear/Parametric components:
 Intercept         x1         x2
3.00099626 0.09698672 3.06092318
```

As for the previous examples, a graphical output can be obtained using the `plot` function like in

the following code. The results are shown in Figure 7, namely the additive components.

```
R> op <- par(mfrow = c(1,2))
R> plot(m4, ask = FALSE)
R> par(op)
```

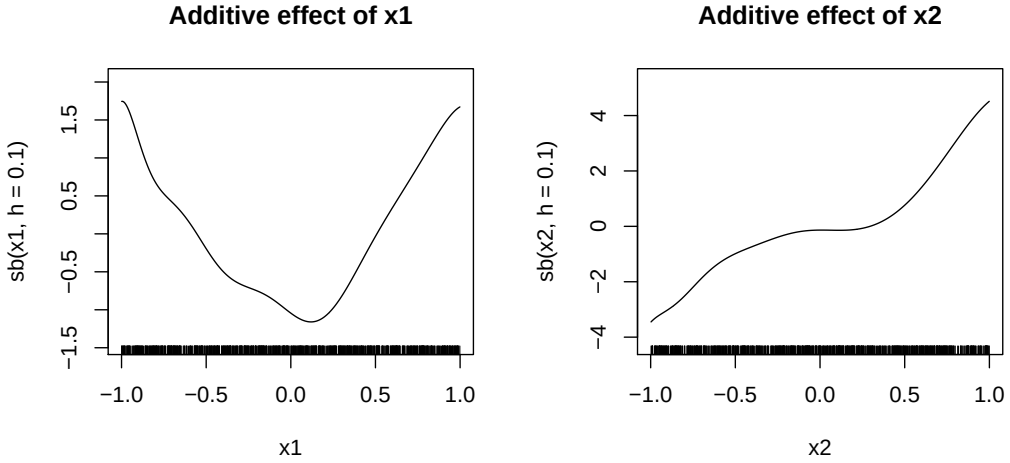


Figure 7: The Poisson Simulated Data: Estimated nonparametric functions.

Summary

We have first given a general introduction into the class of Generalized Structures Models, together with the powerful kernel-based smooth backfitting estimator to fit the members of this model class. This was accompanied by a (certainly incomplete) literature review and closed with a small review and discussion of existing methods and software for similar and related models. Except for the varying coefficient model estimator in the **np** package, they are all based on splines. We concluded that, while there is a huge body of literature on SB and its advantages, it is hardly used in practice due to the lack of software. The **wsbackfit** package intends to close this gap.

Next, we provided some insight into the weighted SB and the objective function that is minimized by our algorithm. This allowed us to better explain the users' options like `weights` and `offset`. We outlined which models can be estimated by the presently available package. The description of the procedure was complemented by a section on the implemented CV, bandwidth choice, binning, convergence, and identification issues to clarify the location and scaling of the resulting estimates.

The package description has been kept condense but its use has been illustrated along several examples that cover some of the estimable models. They comprise the use of all options provided. Moreover, the numerical examples give an idea of the estimators' performance. For more details, we recommend consulting the cited articles dealing with the particular models.

We believe that this package is an important enrichment of the existing methods with many useful applications of flexible data analysis and prediction. It can almost straightforwardly be used for testing (Cadarso-Suárez et al., 2006; Mammen and Sperlich, 2021) or studying the heterogeneity of causal effects (Benini and Sperlich, 2021) and many other interesting applications. The next challenge will be the extension of this package to cover the analysis of more complex data (Jeon and Park, 2020). The package is not just open for extensions. We explicitly invite people to contribute.

Bibliography

A. Arcagni and L. Bagnato. *sBF: Smooth Backfitting*, 2014. URL https://CRAN.R-project.org/package=sBF. R package version 1.1.1. [p331]

G. Benini and S. Sperlich. Modeling heterogeneous treatment effects in the presence of endogeneity. *Econometric Reviews*, forthcoming, 2021. [p347]

H. Binder and G. Tutz. A comparison of methods for the fitting of generalized additive models. *Statistics and Computing*, 18:87–99, 2008. URL https://doi.org/10.1007/s11222-007-9040-0. [p331]

N. Bissantz, H. Dette, T. Hildebrandt, and K. Bissantz. Smooth backfitting in additive inverse regression. *Annals of the Institute of Statistical Mathematics*, 68(4):827–853, 2016. URL https://doi.org/10.1007/s10463-015-0517-x. [p331]

A. Brezger, T. Kneib, and S. Lang. Bayesx: Analysing bayesian structured additive regression models. *Journal of Statistical Software*, 14(11), 2005. URL https://doi.org/10.18637/jss.v014.i11. [p330, 331]

C. Cadarso-Suárez, J. Roca-Pardiñas, and A. Figueiras. Effect measures in non-parametric regression with interactions between continuous exposures. *Statistics in Medicine*, 25(4):603–621, 2006. URL https://doi.org/10.1002/sim.2356. [p347]

L. Fahrmeier, T. Kneib, and S. Lang. Penalized structured additive regression for space-time data: a bayesian perspective. *Statistica Sinica*, 14(3):731–761, 2004. URL www.jstor.org/stable/24307414. [p331]

J. Fan and I. Gijbels. *Local polynomial modelling and its applications*. Number 66 in Monographs on statistics and applied probability series. Chapman & Hall/CRC, 1996. [p335]

K. Han and B. U. Park. Smooth backfitting for errors–in–variables additive models. *The Annals of Statistics*, 46(5):2216–2250, 2018. URL https://doi.org/10.1214/17-AOS1617. [p331, 334, 336]

T. Hastie. *gam: Generalized Additive Models*, 2019. URL https://CRAN.R-project.org/package=gam. R package version 1.16.1. [p331]

T. Hastie and R. Tibshirani. *Generalized Additive Models*. Chapman and Hall, London, New York, 1990. [p330, 331]

T. Hayfield and J. Racine. Nonparametric econometrics: The np package. *Journal of Statistical Software*, 27(5), 2008. URL https://doi.org/10.18637/jss.v027.i05. [p331]

M. Hiabu, E. Mammen, M. D. Martinez-Miranda, and J. P. Nielsen. Smooth backfitting of proportional hazards with multiplicative components. *Journal of the American Statistical Association*, 2020. URL doi.org/10.1080/01621459.2020.1753520. [p331]

M. J. Jeon and B. U. Park. Additive regression with hilbertian responses. *Annals of Statistics*, 48 (5):2671–2697, 2020. URL https://doi.org/10.1214/19-AOS1902. [p347]

T. Kneib, C. Belitz, A. Brezger, and S. Lang. Bayesx - bayesian inference in structured additive regression software. *ISBA Bulletin*, 15(1):11–13, 2008. URL https://bayesian.org/wp-content/uploads/2016/09/0803.pdf. [p331]

M. Köhler, A. Schindler, and S. Sperlich. A review and comparison of bandwidth selection methods for kernel regression. *International Statistical Review*, 82(2):243–274, 2014. URL https://doi.org/10.1111/insr.12039. [p334]

Y. K. Lee, E. Mammen, and B. U. Park. Backfitting and smooth backfitting for additive quantile models. *The Annals of Statistics*, 38(5):2857–2883, 2010. URL https://doi.org/10.1214/10-AOS808. [p330]

Y. K. Lee, E. Mammen, and B. U. Park. Flexible generalized varying coefficient regression models. *The Annals of Statistics*, 40(3):1906–1933, 2012. URL https://doi.org/10.1214/12-AOS1026. [p330, 332, 333, 335]

Q. Li and J. Racine. Smooth varying-coefficient estimation and inference for qualitative and quantitative data. *Econometric Theory*, 26(6):1607–1637, 2010. URL www.jstor.org/stable/40930669. [p331]

O. Linton and J. Nielsen. A kernel method of estimating structured nonparametric regression based on marginal integration. *Biometrika*, 82(1):93–100, 1995. URL https://www.jstor.org/stable/2337630. [p331]

O. Linton, S. Sperlich, and I. Van Keilegom. Estimation of a semiparametric transformation model. *The Annals of Statistics*, 36(2):686–718, 2008. URL https://doi.org/10.1214/009053607000000848. [p330, 332]

E. Mammen and J. Nielsen. Generalised structured models. *Biometrika*, 90(3):551–566, 2003. URL https://doi.org/10.1093/biomet/90.3.551. [p330, 331, 332]

E. Mammen and B. Park. Bandwidth selection for smooth backfitting in additive models. *The Annals of Statistics*, 33(3):1260–1294, 2005. URL https://doi.org/10.1214/009053605000000101. [p330, 333]

E. Mammen and S. Sperlich. Backfitting tests in generalised structured models. *Biometrika*, 2021. URL https://doi.org/10.1093/biomet/asaa108. [p347]

E. Mammen, O. Linton, and J. Nielsen. The existence and asymptotic properties of a backfitting projection algorithm under weak conditions. *The Annals of Statistics*, 27(5):1443–1490, 1999. URL https://doi.org/10.1214/aos/1017939138. [p330]

P. McCullagh and J. Nelder. *Generalized linear models*. Chapman and Hall, London, New York, 1989. [p332]

J. Nielsen and S. Sperlich. Smooth backfitting in practice. *Journal of the Royal Statistical Society, B*, 67(1):43–61, 2005. URL https://doi.org/10.1111/j.1467-9868.2005.00487.x. [p331, 333]

J. Opsomer. Asymptotic properties of backfitting estimators. *Journal of Multivariate Analysis*, 73 (2):166–179, 2000. URL https://doi.org/10.1006/jmva.1999.1868. [p330]

J. Opsomer and D. Ruppert. Fitting a bivariate additive model by local polynomial regression. *The Annals of Statistics*, 25(1):186–211, 1997. URL https://doi.org/10.1214/aos/1034276626. [p330]

B. U. Park, E. Mammen, Y. K. Lee, and E. R. Lee. Varying coefficient regression models: A review and new developments. *International Statistical Review*, 83(1):36–64, 2015. URL https://doi.org/10.1111/insr.12029. [p330]

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2016. URL https://www.R-project.org/. ISBN 3-900051-07-0. [p331]

J. Roca-Pardiñas and S. Sperlich. Estimating generalized structured models - a computational note. *Discussion Paper*, 2008. [p332]

J. Roca-Pardiñas and S. Sperlich. Feasible estimation in generalized structured models. *Statistics and Computing*, 20:367–379, 2010. URL https://doi.org/10.1007/s11222-009-9130-2. [p330, 331, 332, 333, 335, 344]

J. Rodríguez-Poó, S. Sperlich, and P. Vieu. Semiparametric estimation of weak and strong separable models. *Econometric Theory*, 19(6):1008–1039, 2003. URL https://doi.org/10.1017/S0266466603196065. [p330]

S. Sperlich and R. Theler. Modeling heterogeneity: A praise for varying-coefficient models in causal analysis. *Computational Statistics*, 30:693–718, 2015. URL https://doi.org/10.1007/s00180-015-0581-y. [p331]

S. Sperlich, D. Tjøstheim, and L. Yang. Nonparametric estimation and testing of interaction in additive models. *Econometric Theory*, 18(2):197–251, 2002. URL https://doi.org/doi:10.1017/S0266466602182016. [p331]

D. Stasinopoulos and R. Rigby. Generalized additive models for location scale and shape (gamlss) in r. *Journal of Statistical Software*, 23(7), 2007. URL https://doi.org/10.18637/jss.v023.i07. [p331]

C. Stone. The dimensionality reduction principle for generalized additive models. *The Annals of Statistics*, 14(2):590–606, 1986. URL https://doi.org/10.1214/aos/1176349940. [p330]

G. Tutz and H. Binder. Generalized additive modelling with implicit variable selection by likelihood based boosting. *Biometrics*, 62(4):961–971, 2006. URL https://doi.org/10.1111/j.1541-0420.2006.00578.x. [p331]

N. Umlauf, T. Kneib, and N. Klein. *BayesX: R Utilities Accompanying the Software Package BayesX*, 2019. URL https://CRAN.R-project.org/package=BayesX. R package version 0.3-1. [p331]

S. Wood. *Generalized Additive Models: An Introduction with R*. Chapman and Hall / CRC Press, New York, 2017. [p331]

Z. Xiao, O. Linton, R. J. Carroll, and E. Mammen. More efficient local polynomial estimation in nonparametric regression with autocorrelated errors. *Journal of the American Statistical Association*, 98:890–992, 2003. URL https://doi.org/10.1198/016214503000000936. [p331]

K. Yu, B. Park, and E. Mammen. Smooth backfitting in generalized additive models. *The Annals of Statistics*, 36(1):228–260, 2008. URL https://doi.org/10.1214/009053607000000596. [p330, 332]

Javier Roca-Pardiñas
Department of Statistics and O.R.
University of Vigo, Spain.
E-mail: roca@uvigo.es

María Xosé Rodríguez-Álvarez
BCAM - Basque Center for Applied Mathematics
and
IKERBASQUE, Basque Foundation for Science
Alameda de Mazarredo, 14
E-48009 Bilbao, Basque Country, Spain
E-mail: mxrodriguez@bcamath.org

Stefan Sperlich
Geneva School of Economics and Management
University of Geneva
Bd du Pont d'Arve 40, CH - 1211 Geneève 4, Switzerland
E-mail: Stefan.Sperlich@unige.ch

RLumCarlo: Simulating Cold Light using Monte Carlo Methods

by Sebastian Kreutzer, Johannes Friedrich, Vasilis Pagonis, Christian Laag, Ena Rajovic, and Christoph Schmidt

Abstract The luminescence phenomena of insulators and semiconductors (e.g., natural minerals such as quartz) have various application domains. For instance, Earth Sciences and archaeology exploit luminescence as a dating method. Herein, we present the R package **RLumCarlo** implementing sets of luminescence models to be simulated with Monte Carlo (MC) methods. MC methods make a powerful ally to all kinds of simulation attempts involving stochastic processes. Luminescence production is such a stochastic process in the form of charge (electron-hole pairs) interaction within insulators and semiconductors. To simulate luminescence-signal curves, we distribute single and independent MC processes to virtual MC clusters. **RLumCarlo** comes with a modularized design and consistent user interface: (1) C++ functions represent the modeling core and implement models for specific stimulations modes. (2) R functions give access to combinations of models and stimulation modes, start the simulation and render terminal and graphical feedback. The combination of MC clusters supports the simulation of complex luminescence phenomena.

Introduction

Light is perhaps the most basic everyday experience. Light emission that is *not* caused by the heating of a substance is called luminescence or 'cold light'. Various fields exploit this phenomenon. For instance, Earth Sciences and archaeology determine the timing of past events (e.g., last sunlight exposure or heating) with a technique called luminescence dating. Since 2012, the luminescence-dating (or more general trapped-charge dating) community has gradually adapted R as a universal tool to analyze, model, and visualize their data. Relevant related CRAN packages are: **BayLum** (Bayesian modeling: Philippe et al., 2019; Christophe et al., 2018), **Luminescence** (luminescence- data analysis, Kreutzer et al., 2012, 2020), **numOSL** (luminescence-data analysis, Peng et al., 2013; Peng and Li, 2018), **RLumModel** (luminescence-data modeling, Friedrich et al., 2016, 2020), **RLumShiny** (graphical interface to functions for plotting and calculation in the framework of luminescence-data analysis, Burow et al., 2016, 2019), and **tgcd** (curve deconvolution, Peng et al., 2016; Peng, 2020).

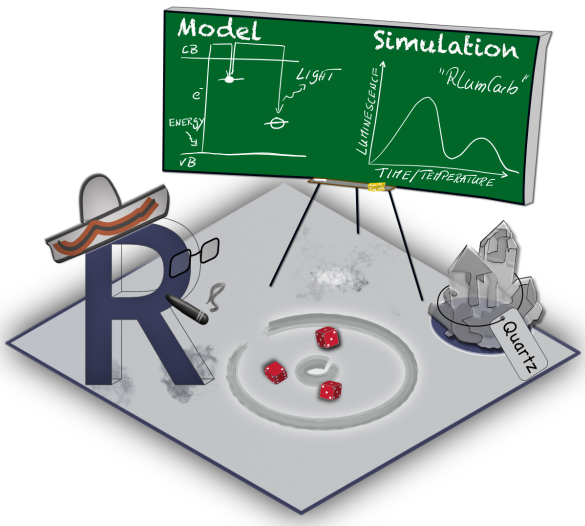


Figure 1: **RLumCarlo** simulates luminescence production in natural minerals such as quartz using Monte Carlo methods. Input is fairly simple (energy-band) models simulating movements of, e.g., electrons in the crystal lattice of quartz. The probability of observing such transitions is a function of energy input in the form of light, heat, or ionizing radiation (e.g., β- or γ-radiation). Th output of **RLumCarlo** is a luminescence curve. This is the light output observed when electrons descend from a higher energy state to a lower energy state emitting part of the energy difference as light.

The luminescence production process is a stochastic process involving discrete random state transitions of subatomic particles. In the case of luminescence, this translates to electrons (and holes) moving to different energy levels, e.g., in the crystal lattice of the natural mineral quartz. Such processes are ideal for Monte Carlo (MC) simulations, and their application has a long and propelling history in physics (cf. Landau and Binder, 2015). Figure 1 summarizes the purpose of **RLumCarlo** developed to simulate luminescence signals in semiconductors and insulators (e.g., quartz) using MC methods. To that end, **RLumCarlo** employs simple (energy-band) models that describe the physical processes in, e.g., the quartz crystal, to simulate xy-curves (luminescence curves). The modeling output expresses the evolution of the light production (luminescence process) over time.

Our contribution, and so **RLumCarlo**, sits on precedent work by Pagonis and Chen (2015), Pagonis et al. (2014), Pagonis et al. (2019), and Pagonis et al. (2020). The included collection of energy-band models for different stimulation modes adapted to MC methods are valuable for, e.g., studying the impact of model parameters on the signal-related stochastic uncertainties or statistic effects in tiny, dosimetric systems. Technically, our approach is closely related to the simulation of birth-and-death processes (for a review on birth-and-death process cf. Novozhilov et al., 2006). Each simulation run describes a Markov process. However, in our case, we allow only a reduction of an initial number of particles (i.e., only death processes).

Herein, we will not derive the full theoretical background of the models, but we will focus on the technical aspects of the package design and the integration of the MC methods. Such a presentation was beyond the scope of previous articles (e.g., Pagonis et al., 2019, 2020), but it is likely of interest to a broader community.

We structured our contribution as follows. The introduction continues with a brief paragraph on luminescence and the term 'cold light'. After that, we detail the rationales for our contribution by recalling conventional modeling approaches in the field. Readers familiar with these topics may safely skip this part. The subsequent section outlines the concept and the implementation of **RLumCarlo**, including code examples. The remainder addresses (A) the implementation of a virtual dosimetric system to simulate weak spatial correlation of dosimetric cluster groups. (B) We outline how **RLumCarlo** can simulate more complex models compared to other solutions, with respect to its strengths and limitations. An outlook outlining the potential to implement more interactions between models will close our manuscript.

'Cold light' in a nutshell

Light emissions of semiconductors or insulators *after* exposure to ionizing radiation is a luminescence phenomenon now and then paraphrased as 'cold light'. The term luminescence relates to light production *not* purely caused by the heating of a substance, a condition called 'incandescence' or black body radiation, but a phenomenon expressing the inherent capacity of a material (dosimeter) to emit light (energy) in the ultraviolet to infrared wavelength range (e.g., Newton Harvey, 1957; Mahesh et al., 1989). Heat-related luminescence phenomena of solids have been explored systematically in physics since the 1930s (Urbach, 1930) to characterize materials and understand charge transfers in dosimeters (e.g., McKeever, 1983; Mahesh et al., 1989). The amount of luminescence, in the context of this manuscript, correlates to the energy absorbed by a dosimeter during ionizing irradiation. The closest analogy to a dosimeter is a battery that can be charged by, e.g., γ-radiation and discharges while emitting light. Natural minerals such as quartz or feldspar are dosimeters. Defects and impurities in their crystal lattice can trap charges (electrons or holes) in metastable states between valence and conduction band. The time an electron spends in such a state can vary from a fraction of a second to millions of years, depending on the crystal lattice configuration and the environmental conditions. The amount of (potential) energy held by an electron in such a center is approximately the energy difference between the valence band and the energy level of the center. A transition of the electron to a lower energy state may lead to a photon emission of the form $E_{photon} = \hbar\omega_{nm} = E_n - E_m$[1]. Energy input ('stimulation') can move the electron out of the defect and eventually it recombines with a hole trapped at another defect ('recombination centre'). Types of stimulation methods relevant for our contribution are heat (thermally stimulated luminescence, TL), visible light (optically stimulated luminescence, OSL), and infrared light (infrared stimulated luminescence, IRSL).

Luminescence phenomena have versatile use in the fields of personal, medical, and accidental dosimetry (e.g., Yukihara and McKeever, 2011). As aforementioned, in Earth Sciences and archaeology, the luminescence of natural minerals gained considerable attractiveness as a dating method (luminescence dating). First attempts exploiting luminescence signals as a chronological tool reach back to the 1950s (Daniels et al., 1953; Houtermans and Stauffer, 1957; Grögler et al., 1958). Nevertheless, it needed a few decades more before the method took off and became today one of the most frequently

[1] $\hbar$ (eV s): Planck constant divided by 2π; ω (radians per s): frequency; E_n (eV): higher energy state; E_m (eV): lower energy state

used dating methods on sediments for the last 250,000 years and beyond (e.g., Aitken, 1985, 1998; Bateman, 2019).

Towards Monte Carlo simulations

To explain luminescence production, Johnson (1939) and Randall and Wilkins (1945) introduced the first basic energy-band models. Today, most of the commonly accepted luminescence models use series of more or less complicated systems of differential equations (for an overview, see Chen and McKeever, 1997; Bøtter-Jensen et al., 2003; Chen et al., 2011) employing energy-band models. Those models provide a proper phenomenological match with measured data for various experimental designs by simulating electronic transitions. 'Conventional' energy-band models available to simulate luminescence production are developed as a set of nonlinear differential equations. This brings some limitations:

1. The models become complex easily and cannot be solved analytically.
2. If numerical methods are used, some equations are numerically unstable, which may lead to wrong simulation results.
3. A convenient assumption in many of such models is a great abundance of spatially uniformly distributed traps and recombination centers. However, this is not always the most prudent assumption. A spatial correlation and cluster formation of centers may exist for various reasons (cf. Mandowski and Świałtek, 1992; Chen et al., 2011; Horowitz et al., 2017).
4. Deterministic models do not consider stochastic uncertainties and simulated curves are 'noise free'. These limits subsequent analyses for materials where such uncertainties would matter due to the low, finite, number of charge carriers, and in these scenarios, simulation results are used as reference data to test statistical models used for luminescence data analysis in general.

Modeling code for simulating luminescence production was often written with the tools at hand, e.g., *Mathematica* (e.g., Pagonis et al., 2006), which has led to a fragmentation of incompatible solutions. In 2016, Friedrich et al. (2016) introduced **RLumModel** (Friedrich et al., 2020), pooling available kinetic (non-MC) models available for the luminescence production in quartz. A tantamount suite of R code was presented simultaneously by Peng and Pagonis (2016). We will compare results from **RLumModel** and **RLumCarlo** at the end of this manuscript.

MC simulations offer an alternative and are indispensable if the simulation of defect clusters in combination with the analysis of stochastic uncertainties is desired. Usually, the underlying models are very simple, but can be combined to describe complex systems. Important early work simulating TL using MC methods goes back to Mandowski and Świałtek (1992) and Kulkarni (1994). Mandowski and Świałtek (1992) tried to overcome the prerequisite of a large number of sample carriers, and Kulkarni (1994) investigated MC methods to overcome very long calculation times encountered for numerical calculations in particular scenarios. Kulkarni (1994) (p. 103) also reported a "statistical fluctuation" (noise like scatter) caused by the MC simulations but considered this more as a disadvantage. Later, Pagonis et al. (2020) explicitly exploited this as a feature, similar to birth-and-death processes and their related random uncertainties, to investigate specifically the stochastic uncertainties.

Before we start to detail **RLumCarlo**, a preceding note of caution: Any attempt to answer the question of whether a particular model may better explain the one or the other effect measured in luminescence studies would open Pandora's box (e.g., Horowitz et al., 2017). Consequently, we will not engage in such a discussion. What we have implemented so far in **RLumCarlo** can be modified and exchanged. However, the underlying design concept remains applicable.

The concept of RLumCarlo

RLumCarlo implements energy-band models in a modular approach. Each model can simulate only an isolated effect (e.g., a single curve, see below), but the package design allows various combinations, e.g., in the form of clusters. Hence, **RLumCarlo** can evolve beyond a specific mathematical model through a combination of simple models.

To that end, **RLumCarlo** differs fundamentally from **RLumModel**, where the collected models allow the simulation of complex phenomena and even entire measurement sequences (Friedrich et al., 2016) but are self-contained by design. In other words, simulations cannot evolve beyond a specific mathematical model selected by the user. In **RLumCarlo**, the implemented energy-band models can simulate only isolated effects (e.g., a single curve, see below), but the package design allows a combination in the form of clusters. Throughout the text, we will use the word 'clusters' to (1) ascribe virtual units used in the MC simulation to run independent random processes (henceforth MC clusters)

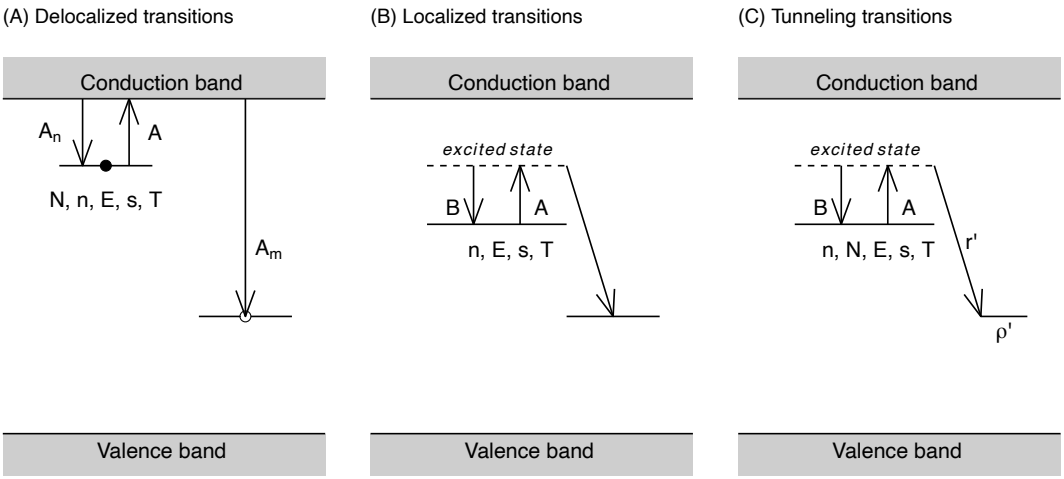


Figure 2: Energy-band model representation of the models implemented in **RLumCarlo**. Letters represent physical model parameters, and arrows indicate allowed transitions. (A) Delocalized transition: the model consists of one single trap and one recombination center. Transition processes involve the conduction band. (B) Localized transition: the model consists of two sub-conduction band energy levels. Charge transitions do not involve the conduction band but take place locally with a constant recombination rate. (C) Tunneling transition: The model consists of one trap and one recombination center. Transitions take place from the excited state into the recombination center without involving the conduction band, and the recombination rate depends on the distance between electrons and holes. All symbols are detailed in the package manual, the package vignette, and the main text. Side note: For the creation of the plots we used **scales** and **ggplot2** (`https://github.com/JohannesFriedrich/EnergyBandModels`).

and (2) to define groups of defects (defect clusters) e.g., defined by their spatial distance. Only the latter carry a physical meaning.

Implemented energy-band models

To date, **RLumCarlo** ships three simple energy-band models (Figure 2) to simulate luminescence production using (A) delocalized transitions, (B) localized transitions, and (C) excited state tunneling transitions. The models are distinguished by the allowed routes of electrons involved in the luminescence process from one energy state to another. Only the first model (Figure 2A) involves the conduction band, while the models in Figures 2B and C limit the allowed electron pathways to energy levels below the conduction band.

While the parameters differ from model to model and depend on the stimulation mode (heat or light, continuous or ramped), key entities remain alike across the models, such as the trap depth (the energy difference of the electron state from the conduction band) E (eV), the attempt to escape frequency of an electron from the trap (short: frequency factor) s (s^{-1}), the temperature T (K), and the trapped concentration of electrons n (cm^{-3}) in the trap. N (models A and B) is the total number of available electrons in cm^{-3} and ρ' is the dimensionless density of recombination centers (model C, Huntley, 2006). The symbols A_n, A_m, and A (model A), B and A (model B), and B, A, and r' (model C) plotted next to the arrows in Figure 2 parametrize, simply put, the rates of the electronic transitions.

The conditions of the simulations are defined through these parameters, with n being the crucial number. Once the electrons have all recombined, the simulation may still continue, but the luminescence signal is zero. As we will detail below, the essential point of the MC simulation, from the physical point of view, is that these concentrations become dimensionless, absolute numbers in a finite system.

Each model supports up to four different stimulation modes (Figure 3), i.e., the type of energy input (light or heat) and its modulation (continuous or ramped).

As an example, we will detail the mathematical background and its implementation for delocalized transitions below. For all other models, we may refer to the cited literature as well as the package manual.

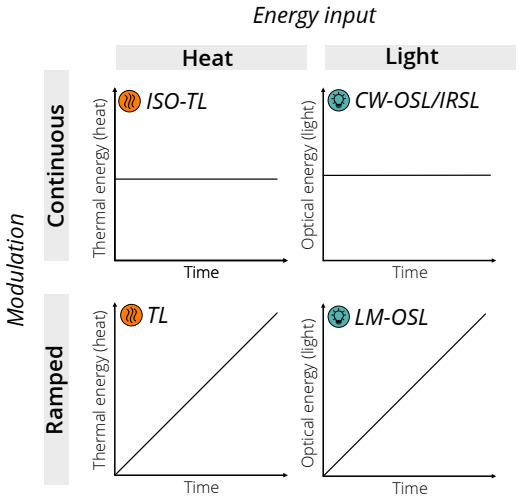


Figure 3: Stimulation modes in **RLumCarlo** applicable to the models. ISO-TL: isothermal TL, i.e., constant stimulation temperature over time. TL: thermal luminescence, i.e., temperature ramps (approximately linearly) over time. CW-OSL/IRSL: continuous wave optically stimulated luminescence respectively infrared stimulated luminescence, constant optical stimulation over time. LM-OSL: linearly modulated optically stimulated luminescence, i.e., linearly ramped optical stimulation over time.

Conceptional overview of the implementation

The basic implementation of the MC processes as a software algorithm consists of two nested loops. The outer loop iterates over a time $0 < t \leqslant t_{max}$ with $t \in \mathbb{R} > 0$. The inner part loops over particles $0 < j \leqslant n$ with $n \in \mathbb{Z}$. The model tests a random number, newly sampled with replacement in each run, against a threshold P. If the sampled random number is smaller than P, the absolute number of particles is reduced by one. The code below shows the basic algorithm outlined above for the radioactive decay, which we have chosen because it can be found in standard textbooks (e.g., Landau and Binder, 2015). Below we used R code for illustrative reasons, while the package implementation is written in C++.

```
n <- 1000
t <- 1:100
P <- 0.2
remaining <- numeric(length(t))

for (t in t) {
  for (j in 1:n) {
    if (runif(1) < P && n > 0)
      n <- n - 1

  }

  if (n > 0)
    remaining[t] <- n

}
```

For example, the algorithm starts with 1,000 particles. In time instant t_1, the random number was smaller for two particles j_6 and j_{576}. Hence, in time instant t_2, the inner loop iterates only over $j = \{1, 2, ..., 998\}$ particles. The absolute number of remaining particles for each t is stored in a vector of length $t_{max}/\Delta t$. This vector is the observed signal curve (in the case of luminescence, the righthand side graph on the green board in Figure 1).

Implementation example for the OTOR model

The implementation for luminescence production in **RLumCarlo** is very similar. To exemplify the adaptation of the models to be run as an MC simulation, we have selected the one-trap one-recombination

center (OTOR) model (based on Halperin and Braner, 1960) for TL. Our description below follows Pagonis and Chen (2015).

The OTOR model for TL can be expressed with the following set of differential equations:

$$\frac{dn}{dt} = -ns\, exp(-\frac{E}{k_B T}) + n_c(N-n)A_n \tag{1}$$

$$\frac{dn_c}{dt} = -\frac{dn}{dt} - n_c m A_m \tag{2}$$

$$I_{TL}(t) = -\frac{dm}{dt} = n_c m A_m. \tag{3}$$

Beyond already mentioned symbols, we used in the equations I_{TL}, the time-dependent intensity, and n_c (cm^{-3}), the current concentration of electrons in the conduction band. The concentration of recombination centers is represented by m (cm^{-3}), where for reasons of charge neutrality $m = n + n_c$. A_n and A_m (both in $cm^3\,s^{-1}$) are the capture coefficients for traps and recombinations centers, respectively. k_B ($eV\,K^{-1}$) is the Boltzmann constant and T (K), the absolute temperature.

By assuming quasi-static equilibrium conditions (Chen et al., 2011)

$$\left|\frac{dn_c}{dt}\right| \ll \left|\frac{dn}{dt}\right|, \left|\frac{dm}{dt}\right| \;;\; n_c \ll n,\;\; n \simeq m, \tag{4}$$

the resulting TL intensity becomes the general one trap equation, GOT:

$$I_{TL}(t) = -\frac{dn}{dt} = s\, exp(-\frac{E}{k_B T})\, \frac{A_m n^2}{(N-n)A_n + nA_m}. \tag{5}$$

$$T = T_0 + \beta \times t, \tag{6}$$

with T (K) and T_0 (K) being temperatures, β ($K\,s^{-1}$) the (heating) rate, and t (s) the simulation time. $p(t) = s\, exp(-\frac{E}{k_B T})$ is the rate of thermal excitation, and $R = \frac{A_n}{A_m}$ is the dimensionless retrapping ratio. The translation into a finite system with a discrete distribution of charge carriers (cf. Mandowski and Swiatek, 1991, 1994), can be expressed through

$$\chi n, \chi N \to \bar{n}, \bar{N}, \tag{7}$$

and the differential equation becomes a difference equation:

$$I_{TL}(t) = -\frac{1}{\beta}\frac{\Delta \bar{n}}{\Delta t} = p(t)\frac{\bar{n}^2}{\bar{N}R + \bar{n}(1-R)}. \tag{8}$$

χ (cm^3) is a constant, $\bar{n}, \bar{N} \in \mathbb{Z}$, and $\Delta t = 1$ s is an appropriate time interval. R is the dimensionless re-trapping ratio in the finite system. To simulate the luminescence process, the related Markov process renders similar to the theory of birth-and-death processes (e.g., Novozhilov et al., 2006), where the population (here of electrons) decreases over continuous time with the probability to observe a transition within Δt being $P = \mu_{\bar{n}} \Delta t$ (here $\mu_{\bar{n}}$ is the "death-rate" in s^{-1}) until the population is depleted. The so-called 'brute force' approach (e.g., Landau and Binder, 2015) tests the population of electrons ($\bar{n}$) per integer time step sequentially by comparing it against a random number sampled with replacement from a continuous distribution $r \sim \mathcal{U}(0,1)$ against the conditional probability P for an electron to get evicted from the trap. In our case, P is calculated as follows:

$$p(t) \times \delta t \times \frac{n}{\bar{N}R + \bar{n}(1-R)}. \tag{9}$$

The factor δt allows values of $\Delta t \neq 1$ while ensuring that $P \ll 1$. $p(t)$ depends on the stimulation mode and the chosen model. For TL (functions named `run_MC_TL_<model>()`) and isothermal TL (functions named `run_MC_ISO_<model>()`) applying the localized or delocalized model $p(t)$ becomes:

$$p(t) = s\, exp(-\frac{E}{k_B T}), \tag{10}$$

and for TL, from tunneling transitions it reads:

$$p(t) = s\, exp(-\frac{E}{k_B T})\, exp(-\rho'^{-1/3} r'), \tag{11}$$

with ρ' being the dimensionless concentration of recombination centers, and r' being the dimensionless tunneling radius (Huntley, 2006). The basic structure in **RLumCarlo** is, however, identical, except for the models based on excited-state tunneling. Here, an additional outer loop iterates over the dimensionless tunneling radius $0 \leqslant r' \leqslant 2$ (Huntley, 2006).

The package design

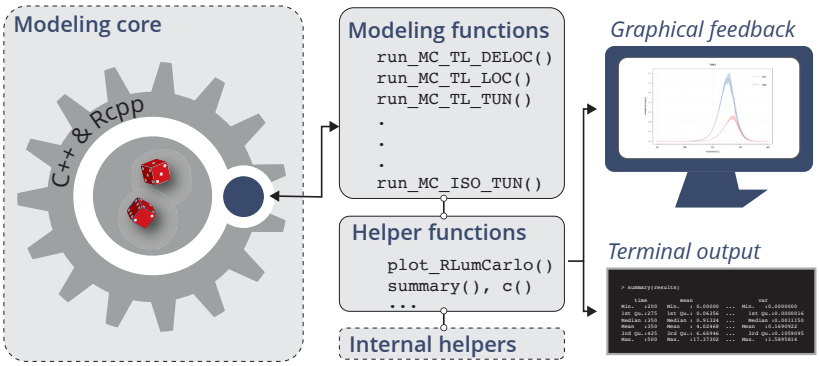


Figure 4: The conceptional design of **RLumCarlo**. User interaction is realized via exported R functions, one function for each model and stimulation type. For the MC runs, we use C++ functions interfaced via **Rcpp**. Helper functions support graphical and terminal feedback.

In Figure 4, we outline the basic layout of **RLumCarlo**. Two design decisions stand out: (1) Each stimulation mode/energy-band model combination has its own exported R function commencing with the prefix `run_MC`. (2) These functions interface one C++ function, each via **Rcpp** (Eddelbuettel et al., 2020) (for an overview, see the vignette of **RLumCarlo**). The R functions provide a convenient user interface, and the C++ functions constitute the workhorse, as shown in the modeling core (Figure 4). While the apparent reason for using C++ was speed, the implementation could have been programmed more concisely, i.e., completely in C++ instead of interfacing C++ with R. However, we wanted to allow code inspection by non-specialists from our field, who may wish to implement other models alike. We found that the separation of the user interface (in R) from the modeling core (in C++) aligns best with our premise of simplicity and flexibility.

As indicated above in the example implementation algorithm, each simulation run (Kulkarni, 1994, used the term 'particle tracking') starts with $n > 1$ and ends at t_{max}, while $I(t) = 0$ for $n = 0$. In reality, one has to execute several simulation runs separately (henceforth 'MC clusters', to be distinguished from defect clusters), either to reduce the statistical fluctuation or to estimate the stochastic error (Kulkarni, 1994). **RLumCarlo** runs the simulations in virtual MC clusters on single or multicore systems using **parallel** (R Core Team, 2020), **doParallel** (Microsoft Corporation and Weston, 2020), and **foreach** (Microsoft and Weston, 2020) supported by helper functions (Figure 4), to summarize results and to provide S3-class based graphical output.

Simple illustrative examples

Simulations start with a call of the respective function, e.g., for `TL` using the `DELOC` model `run_MC_TL_DELOC()` or `run_MC_TL_LOC()` for the `LOC` model, respectively (see Figures 2A-B).

```
results <- run_MC_TL_DELOC(
 s = 3.5e12,
 E = 1.45,
 clusters = 10,
 n_filled = 200,
 R = 1,
 times = 100:450)
```

The function parameter names follow the terminology used for the mathematical expression of the models as closely as possible. For example, `E` and `s` are E (trap depth in eV) and s (frequency factor in s^{-1}). `R` (R in Figure 3) is the so-called re-trapping ratio, expressing the chance an electron of *not* passing into the conduction band, and `n_filled` is $\bar{n}$, the number of electrons at the beginning of

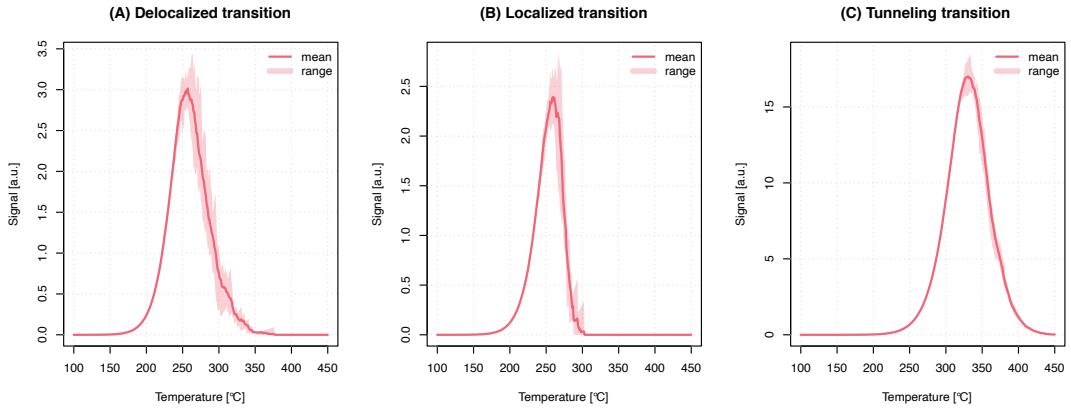


Figure 5: Exemplary comparison of TL signals simulated for all three RLumCarlo models: (A) delocalized (DELOC), (B) localized (LOC), and (C) tunneling (TUN). General physical parameters, such as E (1.45 eV), s ($3.5 \times 10^{12}\,s^{-1}$), and the stimulation temperature (100–450 °C) were kept constant for all models.

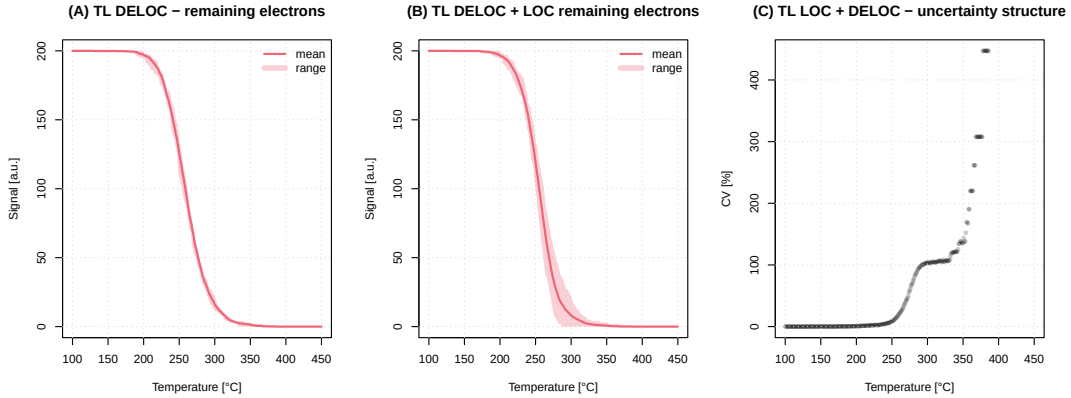


Figure 6: (A) Plot of the remaining electrons for the TL process using the delocalized transition model for which we show the luminescence signal in Figure 5A. (B) Plot of remaining electrons for two models combined in one system. (C) Stochastic uncertainty structure from (B).

the simulation. The duration of the simulation on the time domain (which is not the duration of the computation) is set by `times`. The parameter `clusters` sets the number of MC clusters, i.e., the number of Markov chains. High numbers in `clusters` increase the confidence in the simulation output at the cost of more computation time.

The output can be passed to a dedicated plot function (`plot_RLumCarlo()`). The function supports a couple of standard plot arguments, such as `main` for the title of the plot, which is passed down to `graphics::plot.default()` via `...` (type `?dots` in the R terminal).

```
plot_RLumCarlo(
  object = results,
  legend = TRUE,
  main = "(A) Delocalized transition")
```

The parameter `n_filled` can be a vector enabling different starting conditions for each MC cluster. Figure 5A shows the graphical output for delocalized transitions along with the simulation results for `TL` stimulation using localized (Figure 5B) and tunneling transitions (Figure 5C). The output is an object of class `RLumCarlo_Model_Output`, which is a `list` comprising a multi-dimensional `array` (one slice per MC cluster) with the resulting luminescence signal and a `numeric` vector for the stimulation time.

Currently we provide S3-generics for `summary()` and `c()`. The first one is also used internally by `plot_RLumCarlo()` to melt the array into a `data.frame` before plotting. The plot output adapts to the used stimulation mode provided via an attribute with each output object.

A straightforward application for this kind of simulation is the study of the impact of physical parameters on the luminescence signal output and the estimation of the stochastic uncertainties, which cannot be achieved with the deterministic approach of differential equations.

We provide more, always up-to-date examples with the package vignette, where we also compiled a table with meaningful physical parameter ranges for each model.

Advanced examples and further considerations

The examples so far presented may not appear very sophisticated, and still, they allow insight that goes beyond a simple educational purpose of simulating luminescence based on phenomenological models. Pagonis et al. (2020), who used a preliminary version of **RLumCarlo**, addressed in detail the stochastic uncertainties of TL and OSL models. These uncertainties come into play in nano-dosimetric materials with a small number of defect clusters where the "finite-size" (Mandowski and Swiatek, 1991) of the system starts to matter in terms of a presumed spatial correlation of defect cluster groups. To some extent, this should also be true for systems exposed to high-energy radiation causing defect clusters (e.g., Mandowski and Swiatek, 1991; Mandowski and Świaltek, 1992). Previously in this paper, we have used the term 'MC clusters'. For a start, in **RLumCarlo**, 'MC clusters' entail independent and continuous Monte Carlo Markov chains employed to simulate luminescence production, starting with a particular number of electrons in the system. Whether the processes are run in parallel or sequentially has no impact on the outcome, except for computation speed. In other words, 'MC clusters' carry no meaning regarding the underlying physics. However, as mentioned above, 'MC clusters' from different models (with the same stimulation mode) can be concatenated (see Figure 6B-C) to simulate defect clusters (also, dosimetric clusters), to which we can attribute physical meaning.

Spatial correlation

To simulate a three-dimensional (dosimetric) system, we can add meaning to MC clusters by reinterpreting them as dosimetric clusters. From the modeling perspective, nothing changes, but MC clusters gain a connotation of having a physical meaning.

Figure 7A illustrates the situation of model combinations transferred into a virtual, three-dimensional dosimetric system. Since all defect clusters are distributed evenly over the system, the distance to each neighboring point is identical, and it is a constant rather than a variable. In other words, the spatial distance between neighboring points does not matter and is of no relevance for the simulation but here chosen for illustrative reasons only. Figure 7B represents a situation that takes one step further. Here, the points are randomly distributed over the system, and points form groups (defect cluster groups). Additionally, **RLumCarlo** supports the mixing of models for the same stimulation mode as in Figure 7A (not shown here). The driving idea of the implementation is the assumption of an individual spatial ordering of defects in a, e.g., quartz crystal to which the luminescence production process might be assigned based on models mentioned above.

Such a system can be created in **RLumCarlo** via `create_ClusterSystem()`. The function distributes points randomly with their coordinates:

$$x_1, y_1, z_1, ..., x_k, y_k, z_k \sim \mathcal{U}(0,1) \mid k \in \mathbb{Z}. \tag{12}$$

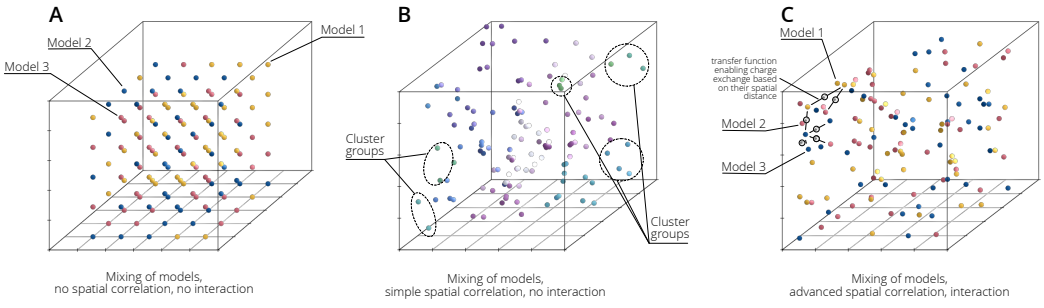


Figure 7: A dosimetric system with three possible approaches of cluster correlation and interaction. (A) Models can be mixed, but no spatial correlation is realized, and no interaction possible. This is the basic mode in **RLumCarlo**. (B) Clusters are grouped by their Euclidean distance, and models can be mixed. Electrons are distributed according to the spatial distance of clusters and also models can be mixed (not shown in the figure). The advanced mode in **RLumCarlo**. (C) Clusters can interact with each other and even exchange electrons. This last stage is subject to future developments of **RLumCarlo**.

Then, the Euclidean distance between the points is determined with `stats:dist()`, which is used by `stats:hclust()` to group the defect clusters (Ξ). To avoid too many small groups, we then cut the cluster tree using `stats:cutree()`, with the outcome shown in Figure 7B. The selection of `stats:hclust()` and `stats:cutree()` for defining the clusters is somewhat arbitrary and might be refined in the future. Therefore, more research, supported by measurements, is needed.

Now, any function from **RLumCarlo** can be used, and the output of `create_ClusterSystem()` is taken as input for the argument `clusters`. For example:

```
run_MC_TL_LOC(s = 3.5e12, E = 1.45, n_filled = 1000,
  clusters = create_ClusterSystem(100))
```

creates a system with 100 randomly distributed defect clusters. If the simulation is run in such a mode, the meaning of `n_filled` changes. Previously, it defined the number of electrons in each cluster ($\bar{n}_{cl_i}$). However, now the same parameter defines the total number of electrons in the entire system. The number of electrons in the ith cluster ($\bar{n}_{cl_i}$) is then an integer fraction of electrons available in each cluster group ($\bar{n}_{\Xi_i} = \texttt{n\_filled}/N_\Xi$, with N_Ξ the total number of cluster groups). The more clusters are in one group, the less electrons are available per cluster in the group and vice versa. While this is a very simple approach, it allows us to simulate basic spatial correlation. Figure 7C drafts a better way of mimicking spatial interaction of clusters, which is, however, not yet part of **RLumCarlo**. While it would be, based on the designed system, easy from the programming perspective, the needed equations to describe to exchange electrons are yet to be developed.

Comparison to RLumModel

In the remainder, we want to compare simulation results from **RLumCarlo**, with other types of solutions, such as **RLumModel** which uses coupled differential equations to simulate luminescence production. **RLumModel** was selected since it was developed by some of the authors of this contribution. However, in theory, simple scripts using any existing models to simulate luminescence should work as well (as long as the models are comparable).

In contrast to **RLumCarlo**, **RLumModel** input values for physical parameters are preset. **RLumModel** encourages users to write a virtual luminescence signal measurement sequence, which is processed based on a pre-defined model with preset physical parameters.

For the comparison, we have selected a TL curve simulated with the luminescence model for quartz by Bailey (2001).

```
output <- RLumModel::model_LuminescenceSignals(
  sequence = list(IRR = c(20, 10, 1), TL = c(20, 400, 1)),
  model = "Bailey2001"
)
```

The results are shown in Figure 8 (here already with the simulation results from **RLumCarlo** plotted on top of it). The output of **RLumModel** is a three-peak-shaped curve. To simulate the same curve in **RLumCarlo**, we used the parameters from the model by Bailey (2001) (his Table 1), e.g., for the first peak:

```
TL110 <- RLumCarlo::run_MC_TL_DELOC(
  s = 5e+12, E = 0.97, R = 5e-10, times = seq(20, 400, 2),
  N_e = output$`conc. level 1 (TL)`[1,2] / 1e+5)
```

`N_e` was divided by a constant to reduce the computation time. The dimensionless parameter `R` corresponds to B (s^{-1}) in Bailey (2001). The other two peaks were simulated alike (objects `TL230` and `TL325`) before all three objects were combined via:

```
object <- c(TL110, TL230, TL325)
```

and plotted on the top of the curve derived from **RLumModel**:

```
RLumCarlo::plot_RLumCarlo(
  object = object,
  plot_value = "sum",
  add = TRUE,
  FUN = function(x) {
    x * 1/(1 + (1e+7 * exp(-0.61/(8.617e-5 * (object$time + 273))))))}
)
```

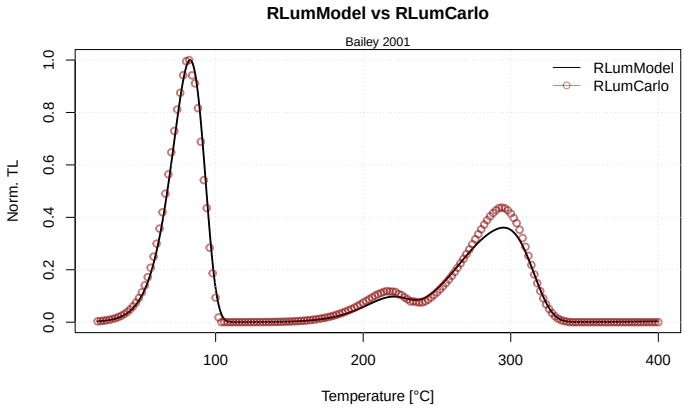


Figure 8: Simulation results of **RLumModel** and **RLumCarlo**. Qualitatively both approaches show a good match.

The argument `plot_value = "sum"` was used to plot the total count sum instead of its average. The additional function injected via the argument `FUN` corrects the TL curves for a phenomenon known as thermal quenching (Wintle, 1975). This is a reduction of luminescence production efficiency at higher temperatures. The chosen quenching parameters follow roughly data measured for quartz by Friedrich et al. (2018). In summary, the results in Figure 8 show that even complex luminescence models can be simulated through the combination of clusters, which brings us back to the initial 'simplicity' premise of **RLumCarlo**. Still, a big 'but' remains. Luminescence models such as those proposed by Bailey (2001) or Pagonis et al. (2008) go beyond single curve simulations. Their purpose is to deliver a general kinetic model for luminescence production of, e.g., quartz, including the simulation of trap filling by irradiation and the simulation of the thermal activation history of the mineral. By contrast, so far all simulations in **RLumCarlo** start with a predefined number of electrons in a trap and are not by default limited to a specific dosimeter. **RLumCarlo** can model more complex luminescence phenomena, but not in a pre-described way out of the box. Instead, **RLumCarlo** is more like a patch box with each model representing a socket ready to be flexibly rewired in many ways to simulate cascades of luminescence production. Due to the nature of the chosen MC approach, in theory (adhering to the patch box picture), the number of sockets is not limited.

Summary

The modeling of luminescence phenomena (cold light) of semiconductors and insulators after having received ionizing radiation is a challenging task. MC methods allow setting up flexible and simple systems to simulate luminescence with a finite number of charge carriers. This enables users to address effects usually observed for nano-dosimetric systems, and it provides insight into the stochastic uncertainty structure. We presented **RLumCarlo**, which renders, to our best knowledge, the first open-source and ready-to-use compilation of basic MC luminescence models for different stimulation modes (so far CW-OSL, LM-OSL, ISO-TL, and TL). We showed that the output from different models, which are simulated in separate MC chains in virtual clusters, can be combined to either simulate more complex systems or to mimic simple spatial correlations between cluster groups. The way of the implementation does not limit **RLumCarlo** to a specific dosimeter (e.g., quartz). In this light, **RLumCarlo** can be used in education, and research to test the impact of model parameters, such as cluster sizes and related stochastic uncertainties. Furthermore, **RLumCarlo** can help in in formulating research hypotheses and test them with commonly accepted or new models, still to be developed.

Future work will implement more models to run as MC simulation, e.g., for irradiation processes in crystals (including its luminescence output: radiofluorescence) and for an advanced interaction of clusters.

Acknowledgements

We thank two anonymous reviewers for their thorough reviews and constructive suggestions, which helped to improve our manuscript. Furthermore, we are grateful to the CRAN team in general for their tireless efforts in keeping such a great resource alive and in particular for their patience during the initial submission of **RLumCarlo**. Alex Roy Duncan is thanked for his support on the package development during our stay in Westminster. The development of **RLumCarlo** benefited from the

support of various funding bodies. The initial work by JF, SK and CS was supported by the Deutsche Forschungsgemeinschaft (2015–2018, DFG SCHM 3051/4-1, *"Modelling quartz luminescence signal dynamics relevant for dating and dosimetry"*). Later financial support was secured through the project *"ULTIMO: Unifying Luminescence Models of quartz and feldspar"* granted by the Deutscher Akademischer Austauschdienst (DAAD PPP USA 2018, ID: 57387041). SK was supported by the LabEx LaScArBx (ANR - n.°ANR-10-LABX-52) until 2019. From 2020, SK has received funding from the European Union's Horizon 2020 research and innovation program under the Marie Skłodowska-Curie grant agreement No 844457 (CREDit). This is IPGP contribution number 4202.

Bibliography

M. J. Aitken. *Thermoluminescence dating*. Studies in archaeological science. Academic Press, 1985. [p353]

M. J. Aitken. *An Introduction to Optical Dating*. Oxford University Press, 1998. [p353]

R. M. Bailey. Towards a general kinetic model for optically and thermally stimulated luminescence of quartz. *Radiation Measurements*, 33(1):17–45, 2001. URL https://doi.org/10.1016/S1350-4487(00)00100-1. [p360, 361]

M. D. Bateman. *Handbook of Luminescence Dating*. Whittles Publishing, 2019. [p353]

L. Bøtter-Jensen, S. W. S. McKeever, and A. G. Wintle. *Optically Stimulated Luminescence Dosimetry*. Elsevier Science B.V., 2003. [p353]

C. Burow, S. Kreutzer, M. Dietze, M. C. Fuchs, M. Fischer, C. Schmidt, and H. Brückner. RLumShiny - A graphical user interface for the R Package 'Luminescence'. *Ancient TL*, 34:22–32, 2016. URL http://ancienttl.org/ATL_34-2_2016/ATL_34-2_Burow_p22-32.pdf. [p351]

C. Burow, U. T. Wolpert, and S. Kreutzer. *RLumShiny: 'Shiny' Applications for the R Package 'Luminescence'*, 2019. URL https://CRAN.R-project.org/package=RLumShiny. R package version 0.2.2. [p351]

R. Chen and S. W. S. McKeever. *Theory of Thermoluminescence and Related Phenomena*. WORLD SCIENTIFIC, 1997. [p353]

R. Chen, J. L. Lawless, and V. Pagonis. A model for explaining the concentration quenching of thermoluminescence. *Radiation Measurements*, 46:1380–1384, 2011. URL https://doi.org/10.1016/j.radmeas.2011.01.022. [p353, 356]

C. Christophe, A. Philippe, S. Kreutzer, and G. Guerin. *BayLum: Chronological Bayesian Models Integrating Optically Stimulated Luminescence and Radiocarbon Age Dating*, 2018. URL https://CRAN.R-project.org/package=BayLum. R package version 0.1.3. [p351]

F. Daniels, C. A. Boyd, and D. F. Saunders. Thermoluminescence as a Research Tool. *Science*, 117(3040):343–349, 1953. URL https://doi.org/10.1126/science.117.3040.343. [p352]

D. Eddelbuettel, R. Francois, J. Allaire, K. Ushey, Q. Kou, N. Russell, D. Bates, and J. Chambers. *Rcpp: Seamless R and C++ Integration*, 2020. URL https://CRAN.R-project.org/package=Rcpp. R package version 1.0.5. [p357]

J. Friedrich, S. Kreutzer, and C. Schmidt. Solving ordinary differential equations to understand luminescence: 'RLumModel' an advanced research tool for simulating luminescence in quartz using R. *Quaternary Geochronology*, 35(C):88–100, 2016. URL https://doi.org/10.1016/j.quageo.2016.05.004. [p351, 353]

J. Friedrich, S. Kreutzer, and C. Schmidt. Radiofluorescence as a detection tool for quartz luminescence quenching processes. *Radiation Measurements*, 120:33–40, 2018. URL https://doi.org/10.1016/j.radmeas.2018.03.008. [p361]

J. Friedrich, S. Kreutzer, and C. Schmidt. *RLumModel: Solving Ordinary Differential Equations to Understand Luminescence*, 2020. URL https://CRAN.R-project.org/package=RLumModel. R package version 0.2.7. [p351, 353]

N. Grögler, F. G. Houtermans, and H. Stauffer. Radiation damage as a research tool for geology and prehistory. In *5° Rassegna Internazionale Elettronica E Nucleare, Supplemento Agli Atti Del Congresso Scientifico*, pages 5–15. Sezione Nucleare, 1958. [p352]

A. Halperin and A. A. Braner. Evaluation of Thermal Activation Energies from Glow Curves. *Physical Review*, 117(2):408–415, 1960. URL https://doi.org/10.1103/PhysRev.117.408. [p356]

Y. S. Horowitz, I. Eliyahu, and L. Oster. Kinetic Simulations of Thermoluminescence Dose Response: Long Overdue Confrontation with the Effects of Ionisation Density. *Radiation Protection Dosimetry*, 172(4):524–540, 2017. URL https://doi.org/10.1093/rpd/ncv495. [p353]

F. G. Houtermans and H. Stauffer. Thermolumineszenz als Mittel zur Untersuchung der Temperatur - und Strahlungsgeschichte von Mineralien und Gesteinen. *Helvetica Physica Acta*, 30:274–277, 1957. [p352]

D. J. Huntley. An explanation of the power-law decay of luminescence. *Journal of Physics: Condensed Matter*, 18(4):1359–1365, 2006. URL https://doi.org/10.1088/0953-8984/18/4/020. [p354, 357]

R. P. Johnson. Luminescence of Sulphide and Silicate Phosphors. *Journal of the Optical Society of America*, 29(9):387–391, 1939. URL https://doi.org/10.1364/JOSA.29.000387. [p353]

S. Kreutzer, C. Schmidt, M. C. Fuchs, M. Dietze, M. Fischer, and M. Fuchs. Introducing an R package for luminescence dating analysis. *Ancient TL*, 30(1):1–8, 2012. URL http://ancienttl.org/ATL_30-1_2012/ATL_30-1_Kreutzer_p1-8.pdf. [p351]

S. Kreutzer, C. Burow, M. Dietze, M. C. Fuchs, C. Schmidt, M. Fischer, J. Friedrich, S. Riedesel, M. Autzen, and D. Mittelstrass. *Luminescence: Comprehensive Luminescence Dating Data Analysis*, 2020. URL https://CRAN.R-project.org/package=Luminescence. R package version 0.9.8. [p351]

R. N. Kulkarni. The Development of the Monte Carlo Method for the Calculation of the Thermoluminescence Intensity and the Thermally Stimulated Conductivity. *Radiation Protection Dosimetry*, 51(2): 95–105, 1994. URL https://doi.org/10.1093/oxfordjournals.rpd.a082126. [p353, 357]

D. P. Landau and K. Binder. *A Guide to Monte Carlo Simulations in Statistical Physics*. Cambridge University Press, 2015. [p352, 355, 356]

K. Mahesh, P. S. Weng, and C. Furetta. *Thermoluminescence in solids and its application*. Thermoluminescence in solids and its application. Nuclear Technology Publishing, England, 1989. [p352]

A. Mandowski and J. Świaltek. Monte Carlo simulation of thermally stimulated relaxation kinetics of carrier trapping in microcrystalline and two-dimensional solids. *Philosophical Magazine B*, 65(4): 729–732, 1992. URL https://doi.org/10.1080/13642819208204910. [p353, 359]

A. Mandowski and J. Swiatek. On the determination of trap parameters from TSC spectra in finite-size systems. In *7th International Symposium on Electrets (ISE 7)*, pages 588–593. IEEE, 1991. URL https://doi.org/10.1109/ISE.1991.167278. [p356, 359]

A. Mandowski and J. Swiatek. Monte Carlo simulation of TSC and TL in spatially correlated systems. In *8th International Symposium on Electrets (ISE 8)*, pages 461–466. IEEE, 1994. URL https://doi.org/10.1109/ISE.1994.514813. [p356]

S. W. S. McKeever. *Thermoluminescence of solids*. Thermoluminescence of solids. Cambrigde University Press, 1983. [p352]

Microsoft and S. Weston. *foreach: Provides Foreach Looping Construct*, 2020. URL https://CRAN.R-project.org/package=foreach. R package version 1.5.1. [p357]

Microsoft Corporation and S. Weston. *doParallel: Foreach Parallel Adaptor for the 'parallel' Package*, 2020. URL https://CRAN.R-project.org/package=doParallel. R package version 1.0.16. [p357]

E. Newton Harvey. *A history of luminescence from the earliest times until 1900*. Philadelphia, American Philosophical Society, 1957. [p352]

A. S. Novozhilov, G. P. Karev, and E. V. Koonin. Biological applications of the theory of birth-and-death processes. *Briefings in Bioinformatics*, 7(1):70–85, 2006. URL https://doi.org/10.1093/bib/bbk006. [p352, 356]

V. Pagonis and R. Chen. Monte Carlo simulations of TL and OSL in nanodosimetric materials and feldspars. *Radiation Measurements*, 81:262–269, 2015. URL https://doi.org/10.1016/j.radmeas.2014.12.009. [p352, 356]

V. Pagonis, G. Kitis, and C. Furetta. *Numerical and Practical Exercises in Thermoluminescence*. Springer, 2006. [p353]

V. Pagonis, E. Balsamo, C. Barnold, K. Duling, and S. McCole. Simulations of the predose technique for retrospective dosimetry and authenticity testing. *Radiation Measurements*, 43(8):1343–1353, 2008. URL https://doi.org/10.1016/j.radmeas.2008.04.095. [p361]

V. Pagonis, E. Gochnour, M. Hennessey, and C. Knower. Monte Carlo simulations of luminescence processes under quasi-equilibrium (QE) conditions. *Radiation Measurements*, 67(C):67–76, 2014. URL https://doi.org/10.1016/j.radmeas.2014.06.005. [p352]

V. Pagonis, J. Friedrich, M. Discher, A. Müller-Kirschbaum, V. Schlosser, S. Kreutzer, R. Chen, and C. Schmidt. Excited state luminescence signals from a random distribution of defects - A new Monte Carlo simulation approach for feldspar. *Journal of Luminescence*, 207:266–272, 2019. URL https://doi.org/10.1016/j.jlumin.2018.11.024. [p352]

V. Pagonis, S. Kreutzer, A. R. Duncan, E. Rajovic, C. Laag, and C. Schmidt. On the stochastic uncertainties of thermally and optically stimulated luminescence signals: A Monte Carlo approach. *Journal of Luminescence*, 219:116945, 2020. URL https://doi.org/10.1016/j.jlumin.2019.116945. [p352, 353, 359]

J. Peng. *tgcd: Thermoluminescence Glow Curve Deconvolution*, 2020. URL https://CRAN.R-project.org/package=tgcd. R package version 2.5. [p351]

J. Peng and B. Li. *numOSL: Numeric Routines for Optically Stimulated Luminescence Dating*, 2018. URL https://CRAN.R-project.org/package=numOSL. R package version 2.6. [p351]

J. Peng and V. Pagonis. Simulating comprehensive kinetic models for quartz luminescence using the R program KMS. *Radiation Measurements*, 86:63–70, 2016. URL https://doi.org/10.1016/j.radmeas.2016.01.022. [p353]

J. Peng, Z. Dong, F. Han, H. Long, and X. Liu. R package numOSL: numeric routines for optically stimulated luminescence dating. *Ancient TL*, 31(2):41–48, 2013. URL http://ancienttl.org/ATL_31-2_2013/ATL_31-2_Peng_p41-48.pdf. [p351]

J. Peng, Z. Dong, and F. Han. tgcd: An R package for analyzing thermoluminescence glow curves. *SoftwareX*, pages 1–9, 2016. URL https://doi.org/10.1016/j.softx.2016.06.001. [p351]

A. Philippe, G. Guérin, and S. Kreutzer. BayLum - An R package for Bayesian analysis of OSL ages: An introduction. *Quaternary Geochronology*, 49:16–24, 2019. URL https://doi.org/10.1016/j.quageo.2018.05.009. [p351]

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2020. URL https://www.R-project.org/. [p357]

J. T. Randall and M. H. F. Wilkins. Phosphorescence and Electron Traps. I. The Study of Trap Distributions. *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 184 (999):365–389, 1945. URL https://doi.org/10.1098/rspa.1945.0024. [p353]

F. Urbach. Zur Lumineszenz der Alkalihalogenide II. Messungsmethoden; erste Ergebnisse; zur Theorie der Thermolumineszenz. *Sitzungsberichte der Akademie der Wissenschaften in Wien, Mathematisch-Naturwissenschaftliche Klasse, Abt. IIa, Mathematik, Astronomie, Physik, Meteorologie und Technik*, 139: 363–372, 1930. [p352]

A. G. Wintle. Thermal Quenching of Thermoluminescence in Quartz. *Geophysical Journal of the Royal Astronomical Society*, 41:107–113, 1975. URL https://doi.org/10.1111/j.1365-246X.1975.tb05487.x. [p361]

E. G. Yukihara and S. W. S. McKeever. *Optically Stimulated Luminescence*. Wiley, 2011. [p352]

Sebastian Kreutzer
Department of Geography & Earth Sciences, Aberystwyth University
Aberystwyth
SY23 3DB, Wales, United Kingdom
IRAMAT-CRP2A, UMR 5060, CNRS-Université Bordeaux Montaigne
Maison de l'Archéologie
Esplanade des Antilles
36607 Pessac Cedex, France

ORCiD: 0000-0001-9166-4563
sebastian.kreutzer@aber.ac.uk

Johannes Friedrich
Chair of Geomorphology, University of Bayreuth

Universitätsstr. 30
95447 Bayreuth, Germany

ORCiD: 0000-0002-0805-9547
johannes.friedrich@posteo.de

Vasilis Pagonis
Physics Department, McDaniel College
Westminster, MD 21157, USA

ORCiD: 0000-0002-4852-9312
vpagonis@mcdaniel.edu

Christian Laag
Université de Paris, Institut de Physique du Globe de Paris, CNRS
75005 Paris, France
Chair of Geomorphology, University of Bayreuth
95447 Bayreuth, Germany

ORCiD: 0000-0002-6012-1029
laag@ipgp.fr

Ena Rajovic
Chair of Geomorphology, University of Bayreuth
95447 Bayreuth, Germany

ena.rajovic@uni-bayreuth.de

Christoph Schmidt
Institute of Earth Surface Dynamics, University of Lausanne
1015 Lausanne, Switzerland

ORCiD: 0000-0002-2309-3209
christoph.schmidt@unil.ch

OneStep : Le Cam's One-step Estimation Procedure

by Alexandre Brouste, Christophe Dutang and Darel Noutsa Mieniedou

Abstract The **OneStep** package proposes principally an eponymic function that numerically computes Le Cam's one-step estimator, which is asymptotically efficient and can be computed faster than the maximum likelihood estimator for large datasets. Monte Carlo simulations are carried out for several examples (discrete and continuous probability distributions) in order to exhibit the performance of Le Cam's one-step estimation procedure in terms of efficiency and computational cost on observation samples of finite size.

Introduction

In the statistical experiments generated by i.i.d. observation samples, the sequence of maximum likelihood estimators (MLE) is known to be asymptotically efficient under very general assumptions and consequently presents the fastest convergence rate and the lowest possible asymptotic variance.

Although the sequence of MLE is asymptotically efficient, it is generally not expressed in a closed form and requires time consuming numerical computations. On the other hand, the other generic estimation procedures which can sometimes be computed faster, as the method of moments, do not generally reach the optimal asymptotic variance.

In R, the package **fitdistrplus** (see Delignette-Muller and Dutang (2015)) is commonly used to infer the parameters of univariate probability distributions. For non-censored datasets, **fitdistrplus** allows four different estimation methods: maximum likelihood, moment matching, quantile matching, and maximum goodness-of-fit estimation. The unified approach is provided by `fitdist()` which returns an S3-object having usual generic functions (`plot`, `summary`, `logLik`, ...) as well as dedicated `fitdist`-functions (`gofstat`, `bootdist`, ...).

Other packages with similar purposes are **EstimationTools** and **DistributionFitR**, providing MLE for non-censored data. Many other packages also provide estimation procedure on the basis of one function per probability distribution; see, e.g., packages **univariateML**, **propagate**. A final package, which is worth mentioning, is **fitter**, which fits a set of probability distributions on a given dataset sequentially.

However, as soon as the Fisher information matrix is sufficiently regular with respect to the parameter to be estimated, Le Cam's one-step estimation procedures can be achieved (see Le Cam (1956)). They are based on an initial sequence of guess estimators and a single Newton step or a Fisher scoring step on the loglikelihood function. Namely, they write

$$\overline{\vartheta}_n = \vartheta_n^* + \mathcal{I}(\vartheta_n^*)^{-1} \cdot \frac{1}{n} \sum_{j=1}^{n} \dot{\ell}(\vartheta_n^*, X_j), \quad n \geq 1, \tag{1}$$

for the Fisher scoring type procedure, where $(\vartheta_n^*, n \geq 1)$ is the initial sequence of guess estimators, $\ell(\vartheta)$ is the loglikelihood function, the dot is the notation for the gradient with respect to ϑ, and $\mathcal{I}(\vartheta) = -\mathbf{E}_\vartheta(\ddot{\ell}(\vartheta))$ is the Fisher information matrix and

$$\overline{\vartheta}_n = \vartheta_n^* + \widehat{\mathcal{I}}_n(\vartheta_n^*)^{-1} \cdot \frac{1}{n} \sum_{j=1}^{n} \dot{\ell}(\vartheta_n^*, X_j), \quad n \geq 1, \tag{2}$$

for the Newton type procedure, where $\widehat{\mathcal{I}}_n(\vartheta) = -\frac{1}{n}\sum_{j=1}^{n} \ddot{\ell}(\vartheta, X_j)$ is the opposite of the Hessian for the loglikelihood function.

The sequence of Le Cam's one-step estimators presents certain advantages over the sequence of MLE and over the initial sequence of estimators (method of moments, quantile matching method, etc.) in terms of computational cost and asymptotic variance. It is much less computationally expensive than the MLE, while it has the same rate and the same asymptotic variance. Since there is no full numerical optimization (but only one computation of the Newton step or the Fisher scoring step), the procedure is faster and appropriate for very large datasets. On the other hand, it is asymptotically optimal in terms of asymptotic variance, which is generally not the case for the initial sequence of guess estimators.

For several probability distributions (Gaussian, Exponential, Lognormal, Poisson, Geometric), a fast computable sequence of estimators is already asymptotically efficient, and no correction is

needed. When the Fisher information is given in a closed form (Gamma, Beta, χ^2, Weibull, Pareto II, Cauchy), Le Cam's one-step estimation procedure (1) is executed. In all the other cases, the estimation procedure (2) is used either with the score function and the Hessian in a closed form (Negative Binomial) or with the numerical approximation of the score function and the Hessian with the package **numDeriv** (see Gilbert and Varadhan (2019)).

Le Cam's one-step estimation procedures are implemented in the package **OneStep**, and the eponymic function **onestep** is described in this paper and used on different examples. Monte Carlo simulations are performed for several examples in order to exhibit the performance of Le Cam's one-step procedure on samples of finite size. They exhibit their asymptotic efficiency simultaneously with their better computational cost for several probability distributions.

The function `onestep`

Let $(X_1, X_2, \ldots, X_n)$ be a sample of i.i.d. random variables. The probability density function of X_1 is denoted f and depends on an unknown parameter $\vartheta \in \Theta \subset \mathbb{R}^p$ which is to be estimated.

Le Cam's one-step estimation procedure is based on an initial sequence of guess estimators and a Fisher scoring step or a single Newton step on the loglikelihood function. For the novice user, the function **onestep** automatically chooses the best procedure to be used. There is, consecutively, a single command for the user, which is

```
onestep(data,distr)
```

The function **onestep** presents several procedures internally depending on whether the initial sequence of guess estimators is in a closed-form or not and whether the score and the Fisher information matrix can be elicited in a closed-form.

Here is the list of the procedures taken into account in the **onestep** function:

1. Distributions for which the MLE is already explicit: `norm`, `exp`, `lnorm`, `invgauss` for continuous probability distributions, and `pois`, `geom` for discrete probability distributions. For this class, the explicit MLE is returned.
2. Distributions for which the initial sequence of guess estimators, the score and the Fisher information matrix have been elicited in a closed-form: `gamma`, `beta`, `chisq` with an initial sequence of moment estimators, `cauchy` with an initial sequence of quantile matching estimators, and `weibull` with an initial sequence of graphical plot estimators. For this class, Le Cam's one-step procedure (1) is applied.
3. Distributions for which the initial sequence of guess estimators, the score, and the Hessian have been elicited in a closed-form: `nbinom`. For this class, Le Cam's one-step procedure (2) is applied.
4. Distributions for which the initial sequence is numerically computed on a subsample, but the score and the Fisher information matrix have closed-form: `pareto`. For this class, Le Cam's one-step procedure (1) is executed.
5. For all other distributions, if the density function is well defined, the numerical computation of the Newton step in Le Cam's one-step procedure (2) is proposed with an initial sequence of guess estimators, which is the sequence of maximum likelihood estimators computed on a subsample.

As is described, all explicit distributions of the `mmedist` function from **fitdistrplus** have been corrected in the **onestep** function, except `unif` and `logis`. The example `unif` is a famous example of the singular behavior of the MLE. The correction of `logis` is of very small gain from the method of moments estimator to the MLE. For these two distributions, the method of moments is returned.

However, the package also offers several new explicit computations as `cauchy`, `chisq` and `weibull` and applies to distributions coming from the **actuar** package (see Dutang et al. (2008)) such as `invgauss` and `pareto`.

The function `onestep` allows the user to propose its own initial guess estimation in the one-step procedure by specifying the parameter `init` in the command

```
onestep(data,distr,init)
```

The user can consequently use different initial guess estimators for the aforementioned classical distributions (moment estimators, estimators based on the characteristic function, quantile matching estimators, Bayesian estimators, mode-type estimators, graphical methods, *etc.*). Several examples can be found in the documentation of the `onestep` function.

Monte Carlo simulations are done for several examples (discrete and continuous probability distributions) in order to exhibit the performance of Le Cam's one-step estimation procedure in terms of efficiency and computational cost on observation samples of finite size.

For the assessment of the efficiency, the proximity of the renormalized statistical errors $(Y_1, Y_2, \ldots, Y_M)$ to the centered Gaussian asymptotic distribution (for a coordinate in the multivariate setting) is evaluated with the Cramer-Von Mises statistic

$$T = M\omega_M^2 = M\int_{\mathbb{R}} (F_M(y) - F_*(y))^2 \, dF_*(y) = \frac{1}{12M} + \sum_{i=1}^{M} \left(F_*(Y_{(i)}) - \frac{2i-1}{2M} \right)^2,$$

where the order statistics are denoted $Y_{(1)} \leq Y_{(2)} \leq \cdots \leq Y_{(M)}$, $F_*(\cdot)$ is the theoretical Gaussian asymptotic cumulative distribution function and

$$F_M(y) = \frac{1}{M} \sum_{i=1}^{M} \mathbb{1}_{\{Y_i \leq y\}}$$

is the empirical cumulative distribution function. The asymptotic distribution of T is tabulated, for instance, in the `goftest` package. Note that a value of the statistic below 0.7434 corresponds to accept the null (and the equivalence to theoretical Gaussian asymptotic distribution) with an error of type I equal to 1%.

Timing performance (given in seconds) is done with the `proc.time` function (`"elapsed"` time) on a laptop with an Intel Core i7 2.7 GHz processor with 8GB RAM. The `onestep` function is compared to the MLE computed with `mledist` of the **fitdistrplus** package and to the sequence of initial guess estimators (for instance, the moment estimator is computed with `mmedist` for the `gamma`, `beta`, and `nbinom` distributions).

The functions `benchonestep` and `benchonestep.replicate` were used to compare the performance of estimators (see documentation of the **OneStep** package).

Closed-form sequence of initial guess estimators

For the majority of the "closed formula" cases, the initial sequence of guess estimators of the unknown parameter ϑ is the sequence of moment estimators.

Let $(X_1, X_2, \ldots, X_n)$ be a sample of i.i.d. random variables. Let us denote the theoretical moments $m_k(\vartheta) = \mathbf{E}_\vartheta(X_1^k)$ and the empirical moments $\tilde{m}_k = \frac{1}{n}\sum_{j=1}^{n} X_j^k$. The sequence of moment estimators (ME) is generally defined as the solution of the system of equations

$$m_k(\vartheta_n^*) = \tilde{m}_k, \quad k = 1, \ldots, p.$$

Indeed, under very mild conditions, the sequence of moment estimators $(\vartheta_n^*, n \geq 1)$ is asymptotically normal, and therefore, $\sqrt{n}$-consistent (see Ibragimov and Has'minskii (1981)). Namely,

$$\sqrt{n}\,(\vartheta_n^* - \vartheta) \longrightarrow \mathcal{N}\left(0, J^{-1}(\vartheta)A(\vartheta)J^{-T}(\vartheta)\right) \tag{3}$$

in law as $n \to \infty$, where

$$J(\vartheta) = \left(\frac{\partial}{\partial \vartheta_j} m_i(\vartheta) \right)_{1 \leq i \leq p, 1 \leq j \leq p}$$

and

$$A(\vartheta) = \mathbf{E}_\vartheta \left((X_1^i - m_i(\vartheta))(X_1^j - m_j(\vartheta)) \right)_{1 \leq i \leq p, 1 \leq j \leq p}.$$

Here, the notation M^T means the transpose matrix of M. Several examples (Gamma, Beta, and Negative Binomial) are given later on in this section.

However, other $\sqrt{n}$-consistent sequences of initial guess estimators can be used. For instance, an initial sequence of quantile matching estimators is employed for the Cauchy distribution.

It had been shown in Le Cam (1956) that for a $\sqrt{n}$–consistent initial sequence of guess estimators and an uniformly continuous Fisher information matrix, the sequence of Le Cam's one-step estimators defined in (1) or (2) is consistent, asymptotically normal and efficient (in the Fisher sense) with

$$\sqrt{n}\left(\widehat{\vartheta}_n - \vartheta\right) \to \mathcal{N}\left(0, \mathcal{I}(\vartheta)^{-1}\right).$$

In other words, for an initial sequence which is asymptotically rate but not variance efficient, the new sequence is asymptotically rate and variance efficient.

Gamma The first example is the joint estimation of the shape parameter α and scale parameter β in the statistical experiment generated by a sample $(X_1, X_2, \ldots, X_n)$ of i.i.d. Gamma random variables

whose probability density function is given by

$$f(x) = \frac{\beta^\alpha}{\Gamma(\alpha)} x^{\alpha-1} \exp(-\beta x), \quad x > 0.$$

Let us denote $\vartheta = (\alpha, \beta)$. In this statistical experiment, the sequence of maximum likelihood estimators $(\widehat{\vartheta}_n, n \geq 1)$ of ϑ is not in a closed-form. The sequence of MLE satisfies

$$\sqrt{n}\left(\widehat{\vartheta}_n - \vartheta\right) \to \mathcal{N}\left(0, \mathcal{I}(\vartheta)^{-1}\right),$$

where

$$\mathcal{I}(\vartheta) = \begin{pmatrix} \psi^{(2)}(\alpha) & -\frac{1}{\beta} \\ -\frac{1}{\beta} & \frac{\alpha}{\beta} \end{pmatrix}.$$

Here, $\psi^{(n)}$ is the polygamma functions (see (Abramowitz and Stegun, 1972, section 6.4.1, page 260)) defined by $\psi^{(n)}(\alpha) = \frac{\partial^n}{\partial \alpha^n} \log \Gamma(\alpha)$. Consider that

$$m_k(\vartheta) = \frac{\alpha(\alpha+1)\ldots(\alpha+k-1)}{\beta^n}.$$

Consecultively, the sequence of moment estimators $(\vartheta_n^* = (\alpha_n^*, \beta_n^*), n \geq 1)$ given by

$$\alpha_n^* = \frac{\tilde{m}_1^2}{\tilde{m}_2 - \tilde{m}_1^2} \quad \text{and} \quad \beta_n^* = \frac{\tilde{m}_1}{\tilde{m}_2 - \tilde{m}_1^2}$$

is asymptotically normal (see (3)) with

$$J = \begin{pmatrix} \frac{1}{\beta} & -\frac{\alpha}{\beta^2} \\ \frac{1+2\alpha}{\beta^2} & -\frac{2\alpha(\alpha+1)}{\beta^3} \end{pmatrix} \quad \text{and} \quad A(\vartheta) = \begin{pmatrix} \frac{\alpha}{\beta^2} & \frac{2\alpha(\alpha+1)}{\beta^3} \\ \frac{2\alpha(\alpha+1)}{\beta^3} & \frac{\alpha(4\alpha+6)(\alpha+1)}{\beta^4} \end{pmatrix},$$

and consequently does not reach asymptotical efficiency.

We can see in Figure 1 (and in Table 1 with the Cramer-Von Mises statistics) that the sequence of Le Cam's one-step estimators (LCE) reaches efficiency and naturally overperforms the initial sequence of ME in terms of asymptotic variance. Moreover, this sequence is faster to be computed on $M = 10000$ Monte Carlo simulations than the sequence of MLE, as shown in Table 1, displaying total computation times over M.

	MLE	ME	LCE
Computation time (s)	1559.74	29.72	37.46
CvM statistic alpha	0.63	1.04	0.23
CvM statistic beta	0.47	0.86	0.19

Table 1: Computation time and CvM statistics

As it was mentioned in the introduction, the major advantage of the sequence of one-step estimators is that it is computed faster than the maximum likelihood estimator for large datasets. We illustrate this fact in Table 2, where the average computation times (over 10 Monte Carlo simulations) are done for different sample sizes $n = 10^r$, $r = 3, \ldots, 9$ for both MLE and LCE. LCE is between 20 times and 55 times faster than MLE, especially for large sample sizes when there is memory overload.

	10^3	10^4	10^5	10^6	10^7	10^8	10^9
MLE	0.0121	0.1193	1.0056	10.4735	99.4203	2339.0437	31600.8313
LCE	0.0007	0.0053	0.0252	0.2361	2.3152	40.0440	540.7393

Table 2: Average computation time (s)

Beta The second example is the joint estimation of the first shape parameter α and the second shape parameter β in the statistical experiment generated by a sample $(X_1, X_2, \ldots, X_n)$ of i.i.d. Beta random variables whose probability density function is given by

$$f(x) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1}(1-x)^{\beta-1}, \quad x \in [0,1].$$

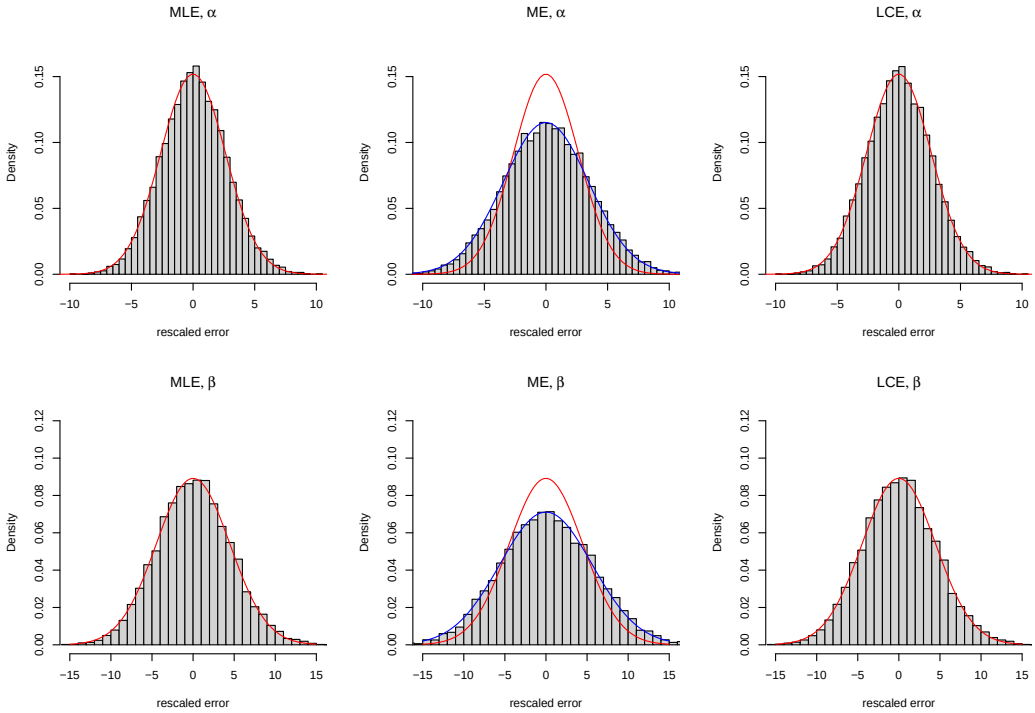


Figure 1: Histograms for the $M = 10000$ Monte Carlo simulations of the rescaled statistical error of the MLE, ME, and LCE for the Gamma distribution with $(\alpha, \beta) = (2, 3)$ and $n = 10000$. Superimposed red and blue lines are the theoretical centered Gaussian asymptotic distributions of the MLE and the ME, respectively.

Let us denote $\vartheta = (\alpha, \beta)$. In this statistical experiment, the sequence of maximum likelihood estimators $(\widehat{\vartheta}_n, n \geq 1)$ of ϑ is not in a closed-form. The sequence of MLE satisfies

$$\sqrt{n}\left(\widehat{\vartheta}_n - \vartheta\right) \to \mathcal{N}\left(0, \mathcal{I}(\vartheta)^{-1}\right),$$

where

$$\mathcal{I}(\vartheta) = \begin{pmatrix} \psi^{(2)}(\alpha) - \psi^{(2)}(\alpha+\beta) & -\psi^{(2)}(\alpha+\beta) \\ -\psi^{(2)}(\alpha+\beta) & \psi^{(2)}(\beta) - \psi^{(2)}(\alpha+\beta) \end{pmatrix}.$$

It is worth mentioning that for the Beta distribution

$$m_k(\vartheta) = \frac{\Gamma(\alpha+k)\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\alpha+\beta+k)}$$

that allows one to build closed-form moment estimators. Namely, denoting $\tilde{w}_n = \frac{\tilde{m}_1(1-\tilde{m}_1)}{\tilde{m}_2 - \tilde{m}_1^2} - 1$, we obtain

$$\alpha_n^* = \tilde{m}_1 \tilde{w}_n \quad \text{and} \quad \beta_n^* = (1-\tilde{m}_1)\tilde{w}_n.$$

Asymptotic variance in (3) of the sequence of moment estimators $(\vartheta_n^* = (\alpha_n^*, \beta_n^*), n \geq 1)$ can also be computed with

$$J = \begin{pmatrix} \frac{\beta}{(\alpha+\beta)^2} & -\frac{\alpha}{(\alpha+\beta)^2} \\ \frac{2\alpha^2\beta+2\alpha\beta^2+2\alpha\beta+\beta^2+\beta}{(\alpha+\beta)^2(\alpha+\beta+1)^2} & -\frac{\alpha(\alpha+1)(2\alpha+2\beta+1)}{(\alpha+\beta)^2(\alpha+\beta+1)^2} \end{pmatrix}$$

and

$$A(\vartheta) = \begin{pmatrix} \frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)} & \frac{2\alpha\beta(\alpha+1)}{(\alpha+\beta)^2(\alpha+\beta+1)(\alpha+\beta+2)} \\ \frac{2\alpha\beta(\alpha+1)}{(\alpha+\beta)^2(\alpha+\beta+1)(\alpha+\beta+2)} & \frac{\alpha(\alpha+1)(2\alpha^3+6\alpha^2\beta+4\alpha\beta^2+14\alpha\beta+4\alpha^2+4\alpha+6\beta^2+6\beta)}{(\alpha+\beta)^2(\alpha+\beta+1)(\alpha+\beta+2)(\alpha+\beta+3)} \end{pmatrix},$$

and consequently the sequence does not reach asymptotical efficiency.

Here again, the sequence of Le Cam's one-step estimators naturally overperforms the initial sequence of ME in terms of asymptotic variance (see Figure 2 and the next Table for CvM statistics).

It is computed faster than the sequence of MLE, as shown in Table 3.

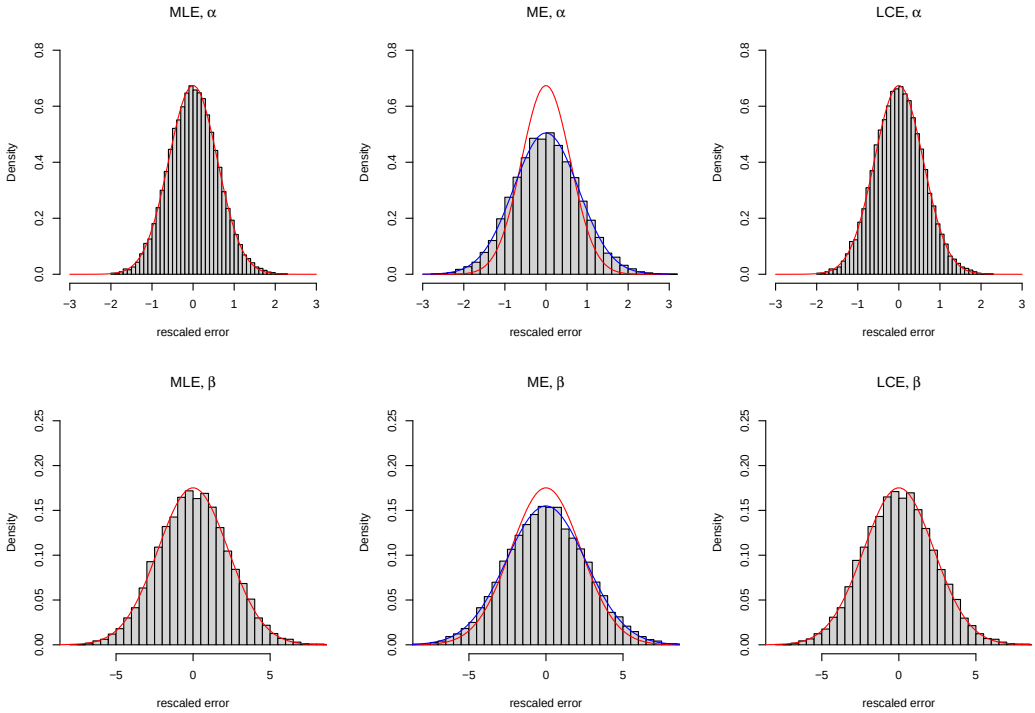


Figure 2: Histograms for the $M = 10000$ Monte Carlo simulations of the rescaled statistical error of the MLE, ME, and LCE for the Beta distribution with $(\alpha, \beta) = (0.5, 1.5)$ and $n = 10000$. Superimposed red and blue lines are the theoretical centered Gaussian asymptotic distributions of the MLE and the ME, respectively.

	MLE	ME	LCE
Computation time (s)	2324.99	43.26	52.02
CvM statistic alpha	0.22	0.06	0.05
CvM statistic beta	0.34	0.11	0.20

Table 3: Computation time and CvM statistics

Cauchy The third example is the joint estimation of the location parameter $m \in \mathbb{R}$ and the scale parameter $d > 0$ in the statistical experiment generated by a sample $(X_1, X_2, \ldots, X_n)$ of i.i.d. Cauchy random variables whose probability density function is given by

$$f(x) = \frac{d}{\pi\left(d^2 + (x-m)^2\right)}, \quad x \in \mathbb{R}. \tag{4}$$

Let us denote $\vartheta = (m, d)$. In this statistical experiment, the sequence of maximum likelihood estimators $(\widehat{\vartheta}_n, n \geq 1)$ of ϑ is not in a closed-form. The sequence of MLE satisfies

$$\sqrt{n}\left(\widehat{\vartheta}_n - \vartheta\right) \to \mathcal{N}\left(0, \mathcal{I}(\vartheta)^{-1}\right),$$

where

$$\mathcal{I}(\vartheta) = \begin{pmatrix} \frac{1}{2d^2} & 0 \\ 0 & \frac{1}{2d^2} \end{pmatrix}.$$

It is worth mentioning that the Cauchy distribution has no first moment, and consecutively its unknown parameter ϑ cannot be estimated via the classical method of moments. But a quantile matching method allows one to define

$$m_n^* = Q_n\left(\frac{1}{2}\right) \quad \text{and} \quad d_n^* = \frac{1}{2}\left(Q_n\left(\frac{3}{4}\right) - Q_n\left(\frac{1}{4}\right)\right), \tag{5}$$

where the sample quantile is usually computed for $p \in (0,1)$ as

$$Q_n(p) = X_{(\lfloor np \rfloor)},$$

with the order statistics denoted $X_{(1)} \leq X_{(2)} \leq \cdots \leq X_{(n)}$. This sequence of estimators $(\vartheta_n^* = (m_n^*, d_n^*), n \geq 1)$ can be shown to be consistent and asymptotically normal, namely

$$\sqrt{n}\,(\vartheta_n^* - \vartheta) \longrightarrow \mathcal{N}(0, \Gamma_1),$$

where

$$\Gamma_1 = \begin{pmatrix} \frac{1}{4f(m)^2} & 0 \\ 0 & \frac{1}{16f(q_1)^2} \end{pmatrix}. \tag{6}$$

Here q_1 is the theoretical first quartile. Consequently, this estimator does not reach the efficient asymptotic variance.

As mentioned previously, the sequence of Le Cam's one-step estimators overperforms the initial sequence of quantile matching estimators (QME) in terms of asymptotic variance (see Figure 3 and the next Table for CvM statistics).

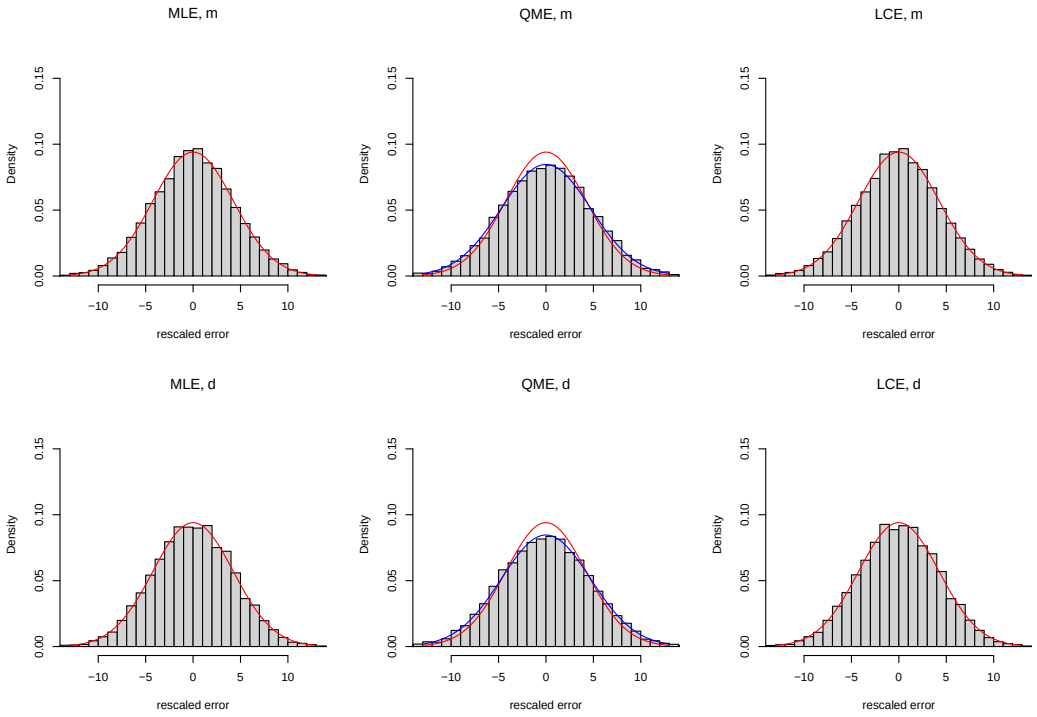


Figure 3: Histograms for the $M = 10000$ Monte Carlo simulations of the rescaled statistical error of the MLE, QME, and LCE for the Cauchy distribution with $(m, d) = (2, 3)$ and $n = 10000$. Superimposed red and blue lines are the theoretical centered Gaussian asymptotic distributions of the MLE and the QME, respectively.

It is computed faster than the sequence of MLE, as shown in Table 4.

	MLE	QME	LCE
Computation time (s)	225.32	7.40	27.59
CvM statistic m	0.08	0.42	0.08
CvM statistic d	0.04	0.10	0.04

Table 4: Computation time and CvM statistics

Pólya (negative binomial) The fourth example is the joint estimation of the size parameter r and the mean parameter μ in the statistical experiment generated by a sample $(X_1, X_2, \ldots, X_n)$ of i.i.d.

negative binomial random variables whose (discrete) probability density function is given by

$$f(x) = \frac{\Gamma(r+x)}{\Gamma(r)x!}\left(\frac{r}{\mu+r}\right)^r\left(\frac{\mu}{\mu+r}\right)^x, \quad x \in \mathbb{N}.$$

Let us denote $\vartheta = (r, \mu)$. In this statistical experiment, the sequence of maximum likelihood estimators $(\widehat{\vartheta}_n, n \geq 1)$ of ϑ is not in a closed-form and the Fisher information matrix neither.

For this distribution,

$$m_1(\vartheta) = \mu \quad \text{and} \quad m_2(\vartheta) = \mu^2 + \mu + \frac{\mu^2}{r}$$

that gives closed-form sequence of moment estimators $(\vartheta_n^* = (r_n^*, \mu_n^*), n \geq 1)$ given by

$$r_n^* = \frac{\tilde{m}_1^2}{(\tilde{m}_2 - \tilde{m}_1^2) - \tilde{m}_1} \quad \text{and} \quad \mu_n^* = \tilde{m}_1.$$

In this discrete case, the sequence of Le Cam's one-step estimators (2) defined with the Hessian still overperforms the initial sequence of ME in terms of asymptotic variance (see Figure 4 and the next Table for CvM statistics). Let us mention that the CvM statistics are computed with the estimated variances for MLE and LCE.

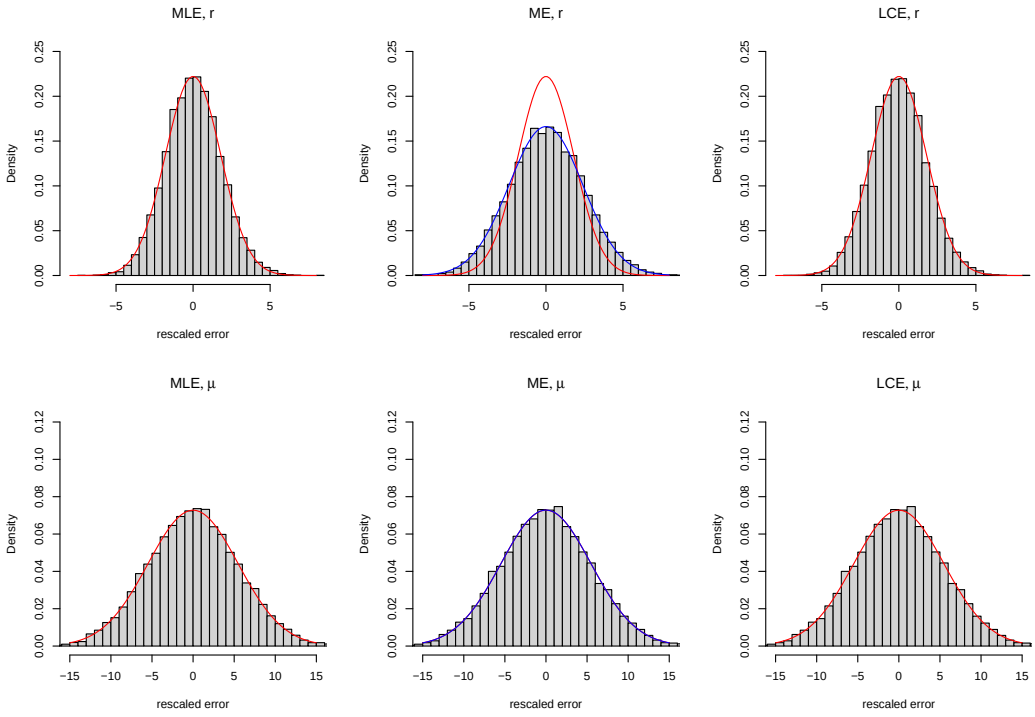


Figure 4: Histograms for the $M = 10000$ Monte Carlo simulations of the rescaled statistical error of the MLE, ME, and LCE for the Polya distribution with $(r, \mu) = (1, 5)$ and $n = 10000$. Superimposed red line is the empirical Gaussian asymptotic distributions of the MLE. Superimposed blue line is the theoretical centered Gaussian asymptotic distributions of the ME.

It also overperforms the sequence of MLE in terms of computation time, as shown in Table 5.

	MLE	ME	LCE
Computation time (s)	2038.33	39.59	104.75
CvM statistic r	0.18	0.71	0.19
CvM statistic mu	0.05	0.05	0.05

Table 5: Computation time and CvM statistics

Weibull The fifth example is the joint estimation of the shape parameter $\tau > 0$ and the rate parameter $\beta > 0$ in the statistical experiment generated by a sample $(X_1, X_2, \ldots, X_n)$ of i.i.d. Weibull random

variables whose probability density function is given by

$$f(x) = \beta\tau(\beta x)^{\tau-1}\exp\left(-(\beta x)^{\tau}\right), \quad x \in \mathbb{R}_*^+.$$

Let us denote $\vartheta = (\tau, \beta)$. In this example, neither the ME nor the MLE is in closed-form. However, the score and the Fisher information matrix can be explicitly computed. For instance,

$$\mathcal{I}(\vartheta) = \begin{pmatrix} \frac{1}{\tau^2}\left(\psi^{(2)}(1) + [\psi^{(1)}(2)]^2\right) & \frac{1}{\beta}\psi^{(1)}(2) \\ \frac{1}{\beta}\psi^{(1)}(2) & \frac{\tau^2}{\beta^2} \end{pmatrix}.$$

Moreover, it is possible to define a closed-form initial sequence of graphical plot estimators (based on the explicit form of the cumulative distribution function). Let us denote $x_i = \log(-\log(1 - i/(n+1)))$ and $Z_i = \log\left(X_{(i)}\right)$ for $i = 1, \ldots, n$. The sequence of ordinary least square (OLS) based estimators $(\vartheta_n^* = (\tau_n^*, \beta_n^*), n \geq 1)$ is defined by

$$\tau_n^* = \left(\frac{\sum_{i=1}^{n}(x_i - \overline{x}_n)Z_i}{\sum_{i=1}^{n}(x_i - \overline{x}_n)^2}\right)^{-1} \quad \text{and} \quad \beta_n^* = \exp\left(\overline{Z}_n - (\tau_n^*)^{-1}\overline{x}_n\right),$$

where $\overline{Z}_n$ stands for the average of $Z_1, \ldots, Z_n$.

The sequence of Le Cam's one-step estimators overperforms the initial sequence of OLS-based estimators in terms of asymptotic variance (see Figure 5 and the next Table). Since OLS presents a small bias for samples of finite size, its CvM statistics are bigger in the next Table.

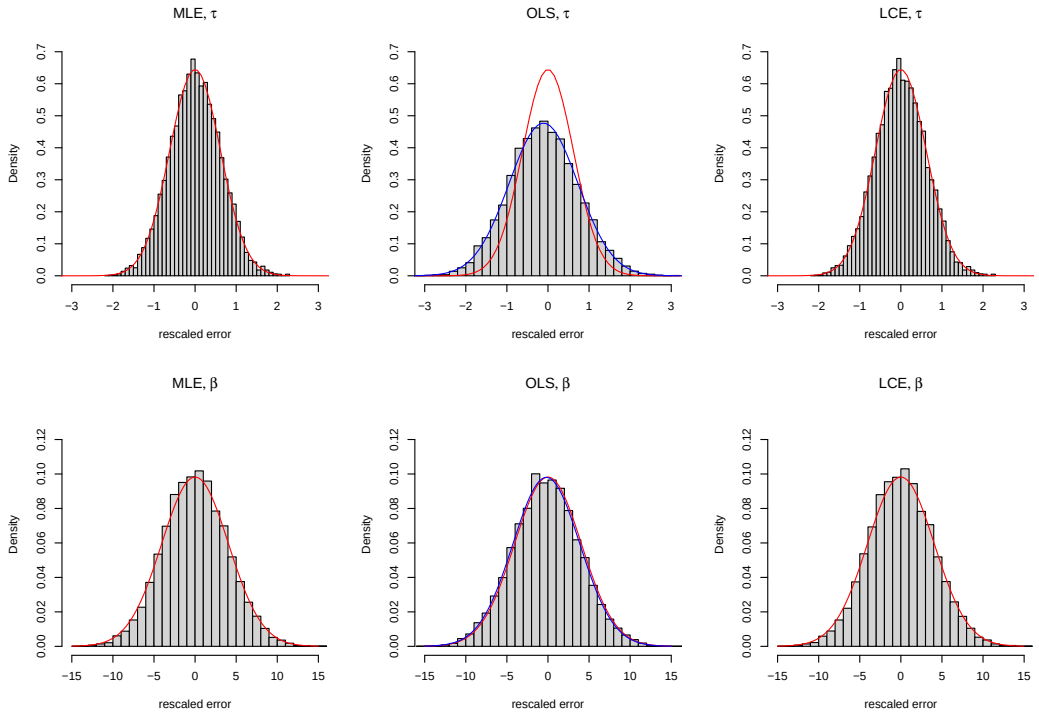


Figure 5: Histograms for the $M = 10000$ Monte Carlo simulations of the rescaled statistical error of the MLE, OLS, and LCE for the Weibull distribution with $(\tau, \beta) = (0.8, 3)$ and $n = 10000$. Superimposed red line is the theoretical centered Gaussian asymptotic distributions of the MLE. Superimposed blue line is the empirical Gaussian asymptotic distributions of the OLS.

It also overperforms the sequence of MLE in terms of computation time for large datasets, as shown in Table 6.

	MLE	OLS	LCE
Computation time (s)	568.83	32.64	82.64
CvM statistic tau	0.13	17.26	0.06
CvM statistic beta	0.59	2.33	0.59

Table 6: Computation time and CvM statistics

Numerically computed initial sequence of guess estimators and numerical computations in the generic case

As soon as the ME is not in closed form or no other closed form estimators can be elicited, the use of a numerical ME as an initial sequence of guess estimator in Le Cam's one-step procedure is difficult to justify. Indeed, no gain will be given in terms of time computation and the use of the MLE is finally equivalent. We propose therefore to use in the **onestep** function an improvement of the classical Le Cam procedure.

Indeed, it can be shown (see Kamatani and Uchida (2015) or Kutoyants and Motrunich (2016)) that for a $n^{\delta/2}$–consistent initial sequence of guess estimators (with $\frac{1}{2} < \delta \leq 1$) and a Lipshitz Fisher information matrix, the sequence of Le Cam's one-step estimators is also consistent, asymptotically normal and efficient (in the Fisher sense). In this setting, for a initial sequence which is neither asymptotically rate nor variance efficient, the new sequence is asymptotically rate and variance efficient.

This result allows one to use the numerical computation of the MLE on a subsample (of size n^δ, for $\frac{1}{2} < \delta \leq 1$) as an initial sequence of guess estimators. Namely, for this initial sequence of guess estimators $(\vartheta_n^*, n \geq 1)$,

$$n^{\frac{\delta}{2}} \left(\vartheta_n^* - \vartheta\right) \to \mathcal{N}\left(0, \mathcal{I}(\vartheta)^{-1}\right).$$

Then the sequence of Le Cam's one-step estimators given by (2) is also consistent, asymptotically normal and efficient (in the Fisher sense) with

$$\sqrt{n}\left(\widehat{\vartheta}_n - \vartheta\right) \to \mathcal{N}\left(0, \mathcal{I}(\vartheta)^{-1}\right).$$

The choice of the exponent δ that measures the size of the subsample $\frac{1}{2} < \delta \leq 1$ is set to 0.9 by default and can be chosen by the user with the parameter `control`, for instance in the call

```
onestep(data, distr, control=list(delta=0.7))
```

The example of Pareto II distribution is interesting and shows the gain in terms of variance (with respect to the initial sequence of guess estimators) and in terms of computation time in comparison with the MLE.

This improved method is also used when the distribution does not belong to the closed-formula family. The initial sequence of maximum likelihood estimators is computed on a subsample with the `mledist` function of the **fitdistrplus** package. The score and the Hessian in the Newton step of the Le Cam procedure (2) are numerically computed with the functions `grad` and `hessian` of the **numDeriv** package (see Gilbert and Varadhan (2019)).

Pareto II The sixth example is the joint estimation of the shape parameter $\alpha > 0$ and the scale parameter $\sigma > 0$ in the statistical experiment generated by a sample $(X_1, X_2, \ldots, X_n)$ of i.i.d. Pareto II (Lomax) random variables whose probability density function is given by

$$f(x) = \frac{\alpha\sigma^\alpha}{(\sigma + x)^{\alpha+1}}, \quad x \in \mathbb{R}^+.$$

In this example neither the ME (when it exists for $\alpha > 2$) nor the MLE are in closed-form. But the score and the Fisher information matrix can be explicitly computed.

Let $\vartheta = (\alpha, \sigma)$. Then, considering the MLE computed on a subsample of size n^δ, $\frac{1}{2} < \delta \leq 1$, as an initial sequence of guess estimators $(\vartheta_n^*, n \geq 1)$, we get

$$n^{\frac{\delta}{2}} \left(\vartheta_n^* - \vartheta\right) \to \mathcal{N}\left(0, \mathcal{I}(\vartheta)^{-1}\right)$$

with

$$\mathcal{I}(\vartheta) = \begin{pmatrix} \frac{\alpha}{\sigma^2(\alpha+1)} & -\frac{1}{\sigma(\alpha+1)} \\ -\frac{1}{\sigma(\alpha+1)} & \frac{1}{\alpha^2} \end{pmatrix}.$$

The computation of the Fisher information matrix can be found in Brazauskas (2003).

Here again, the sequence of Le Cam's one-step estimators naturally overperforms the initial sequence of MLE computed on a subsample in terms of asymptotic variance (see Figure 6 and the next Table for the CvM statistics).

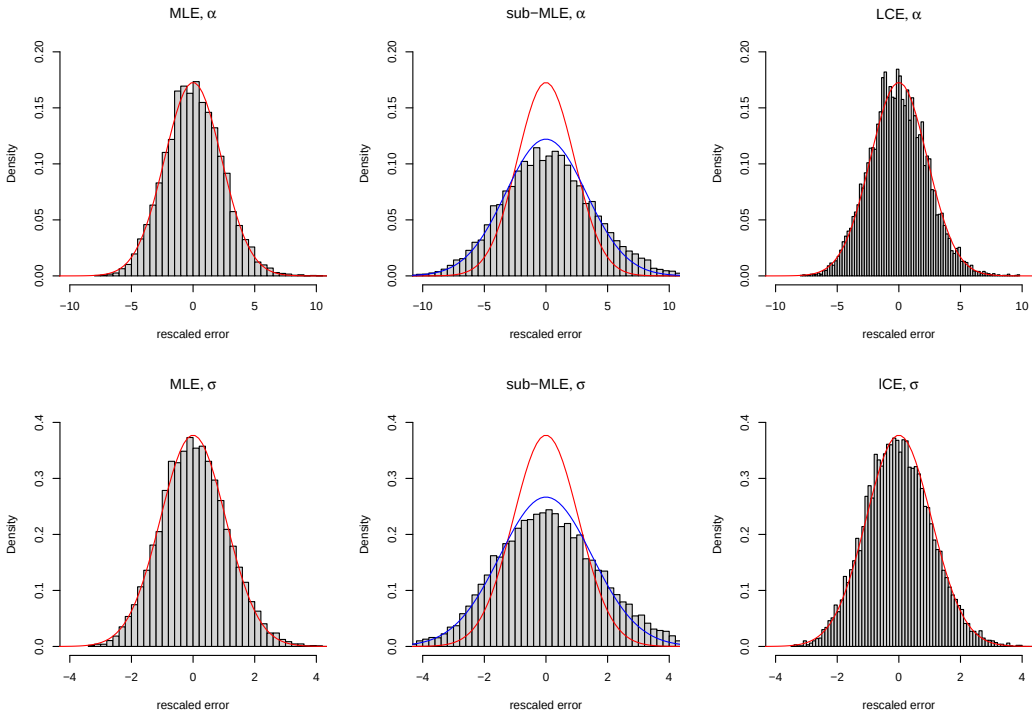


Figure 6: Histograms for the $M = 10000$ Monte Carlo simulations of the rescaled statistical error of the MLE, ME, and LCE for the Pareto II distribution with $(\alpha, \sigma) = (1.1, 0.3)$ and $n = 10000$. Superimposed red and blue lines are the theoretical centered Gaussian asymptotic distributions of the MLE and the MLE on a subsample, respectively.

It is also computed faster than the sequence of MLE, as shown in Table 7.

	MLE	MLEsub	LCE
Computation time (s)	1013.00	393.42	414.08
CvM statistic alpha	0.25	4.72	1.44
CvM statistic sigma	0.43	5.54	1.42

Table 7: Computation time and CvM statistics

Generic function For the generic example, we study the Weibull distribution again and force it to be numerically computed with the function parameter `method="numeric"` in order not to use the closed form processing.

We recall that, in the generic procedure, the score and the Hessian in the Newton step of the Le Cam procedure (2) are numerically computed with the functions `grad` and `hessian` of the **numDeriv** package. By default, the initial sequence of maximum likelihood estimators is computed on a subsample if size n^δ with $\delta = 0.9$.

The numerically computed sequence of Le Cam's one-step estimators reaches asymptotic efficiency for a simulation of $M = 10000$ Monte-Carlo replications of samples of size $n = 10000$ as shown by the CvM statistics summarized in the next table. It is still computed faster than the sequence of MLE, see Table 8.

Acknowledgements We would like to thank Yury Kutoyants and Marius Soltane for fruitful discussions on the topic. We also thank the referees for their valuable comments that improve the

	MLE	LCE	GLCE
Computation time (s)	568.83	82.64	405.10
CvM statistic tau	0.13	0.06	0.25
CvM statistic beta	0.59	0.59	1.08

Table 8: Computation time and CvM statistics

paper.

Bibliography

M. Abramowitz and I. Stegun. Handbook of mathematical functions. *National Bureau of Standards*, 1972. [p369]

V. Brazauskas. Information matrix for Pareto (IV), Burr, and related distributions. *Communications in Statistics – Theory and Methods*, 32(2):315–325, 2003. URL https://doi.org/10.1081/sta-120018188. [p376]

M. L. Delignette-Muller and C. Dutang. fitdistrplus: An R package for fitting distributions. *Journal of Statistical Software*, 64(4):1–34, 2015. URL http://www.jstatsoft.org/v64/i04/. [p366]

C. Dutang, V. Goulet, and M. Pigeon. actuar: An R package for actuarial science. *Journal of Statistical Software*, 25(7):38, 2008. URL http://www.jstatsoft.org/v25/i07. [p367]

P. Gilbert and R. Varadhan. *numDeriv: Accurate Numerical Derivatives*, 2019. URL https://CRAN.R-project.org/package=numDeriv. R package version 2016.8-1.1. [p367, 375]

I. Ibragimov and R. Has'minskii. *Statistical Estimation - Asymptotic Theory*. Springer-Verlag, 1981. URL https://10.1007/978-1-4899-0027-2. [p368]

K. Kamatani and M. Uchida. Hybrid multi-step estimators for stochastic differential equations based on sampled data. *Stat Inference Stoch Process*, 18(2):177–204, 2015. URL https://doi.org/10.1007/s11203-014-9107-4. [p375]

Y. Kutoyants and A. Motrunich. On multi-step MLE-process for Markov sequences. *Metrika*, 79: 705–724, 2016. URL https://doi.org/10.1007/s00184-015-0574-4. [p375]

L. Le Cam. On the asymptotic theory of estimation and testing hypotheses. In *Proceedings of the Third Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Contributions to the Theory of Statistics*, pages 129–156, Berkeley, Calif., 1956. University of California Press. URL https://projecteuclid.org/euclid.bsmsp/1200501652. [p366, 368]

Alexandre Brouste
Laboratoire Manceau de Mathématiques, Le Mans Université
Avenue Olivier Messiaen, 72085 LE MANS
France
ORCID: 0000-0001-6719-7432
Alexandre.brouste@univ-lemans.fr

Christophe Dutang
CEREMADE, CNRS, Université Paris-Dauphine, Université PSL
Place du Maréchal de Lattre de Tassigny, 75016 PARIS
France
ORCID: 0000-0001-6732-1501
christophe.dutang@dauphine.psl.eu

Darel Noutsa Mieniedou
Laboratoire Manceau de Mathématiques, Le Mans Université
Avenue Olivier Messiaen, 72085 LE MANS
France

The HBV.IANIGLA Hydrological Model

by Ezequiel Toum, Mariano H. Masiokas, Ricardo Villalba, Pierre Pitte and Lucas Ruiz

Abstract Over the past 40 years, the HBV (Hydrologiska Byråns Vattenbalansavdelning) hydrological model has been one of the most used worldwide due to its robustness, simplicity, and reliable results. Despite these advantages, the available versions impose some limitations for research studies in mountain watersheds dominated by ice-snow melt runoff (i.e., no glacier module, a limited number of elevation bands, among other constraints). Here we present HBV.IANIGLA, a tool for hydroclimatic studies in regions with steep topography and/or cryospheric processes which provides a modular and extended implementation of the HBV model as an R package. To our knowledge, this is the first modular version of the original HBV model. This feature can be very useful for teaching hydrological modeling, as it offers the possibility to build a customized, open-source model that can be adjusted to different requirements of students and users.

Introduction

Hydrological modeling is widely used by engineers, meteorologists, geographers, geologists, and researchers interested in knowing the runoff of rivers in the coming days or the variations of the snowpack under certain temperature or precipitation changes, among many other hydrological processes.
The Swedish Meteorological and Hydrological Institute (SMHI) ran the first successful simulation of the HBV model in 1972. It was developed to forecast river runoff for hydropower generation in Sweden (Bergström and Lindström, 2015). Up to now, many versions have been developed: HBV-ETH (Switzerland - Braun and Renner (1992)), HBV-Light (Switzerland - Seibert and Vis (2012)), HBV-D (Germany - Krysanova et al. (1999)), HBV-CE (Canada - Stahl et al. (2008)), **TUWmodel** (Austria - Viglione and Parajka (2016)), among others. Despite all these free versions, none of them allows the users to build their own model using a self-defined combination of modules.

Buytaert et al. (2008) identified some prerequisites for hydrological model development: (1) *accessibility* in order to reproduce experimental results; (2) *modularity* as a key element for the development of new *'ad-hoc'* models to evaluate several aspects of the hydrological cycle and to propose improvements; (3) *portability*, so the model can run in many operating systems; and (4) *open-source code* as a fundamental scientific requirement that allows users to revise, correct, and suggest code improvements.

Slater et al. (2019) highlighted some of the key R packages for hydrological modeling; **TUWmodel** is an R version of the HBV model originally written in Fortran (Viglione and Parajka, 2016); **topmodel** and **dynatopmodel** are the R versions of the well-known semi-distributed models TOPMODEL and Dynamic TOPMODEL (Buytaert, 2018; Metcalfe et al., 2015); **airGR** (Coron et al., 2017, 2020) includes several conceptual rainfall-runoff models, a snow accumulation and melt model and the associated functions for their calibration and evaluation; finally, **hydromad** (Andrews et al., 2011) provides a modeling framework for environmental hydrology through water balance accounting and flow routing in spatially aggregated catchments.

Of the models mentioned above, only **airGR**, **hydromad**, and **TUWmodel** present a snow routine to account for accumulation and melting processes (temperature index model), but none of them have routines to account for glacier mass balance. On the other hand, the **glacierSMBM** package (Groos and Mayer, 2017) allows the modeling of glacier surface mass balance in a fully distributed manner, but it was designed to work on the mass balance of a single glacier and to run on a raster-based grid, two aspects that limit its applicability at the basin scale.

The **HBV.IANIGLA** (Toum, 2021) package was built with the aim of providing a modular hydrological model approach that adds to the classic HBV routines functions for the modeling of the surface mass balance of clean and debris-covered glaciers, a fundamental aspect in the hydrological cycle of cold regions of the Andes (Masiokas et al., 2020). The main objective of this article is to present the **HBV.IANIGLA** model structure through its implementation as an R package to serve as a practical guide to better understand how it works. The paper is organized as follows:

- In the next section, we describe the modeling philosophy under HBV and justify the use of a modular approach. We then present the **HBV.IANIGLA** modules and related equations (with some conceptual drawings). We end this section with a small study on model computation times, a fundamental aspect for sensitivity and uncertainty analysis.
- Following the methodology, we focus on two examples: on a synthetic basin and on glacier mass balance. The reader will find more reproducible examples in the package vignettes.

- Finally, we condense the key points of the current version of **HBV.IANIGLA** and propose future improvements.

The HBV.IANIGLA model

The HBV model

The HBV model has been used for 40 years for hydrological studies in mountain regions around the world (Bergström and Lindström, 2015). The model requires relatively few data inputs (air temperature, precipitation, and potential evapotranspiration), which makes it very appropriate in scarce data regions such as the Southern Andes. It has been well-documented by other authors (Seibert and Vis, 2012; Parajka and Blöschl, 2008; Stahl et al., 2008), a feature that facilitates writing new codes and modifying or improving existing equations. Also , it is a bucket-type model with relatively few free parameters to calibrate.

The **HBV.IANIGLA** version not only takes into account precipitation phase partitioning, snow accumulation and melting, actual evaporation and streamflow discharge, but also incorporates a module for simulating the surface mass balance of clean and debris-covered glaciers and another module for glacier-melt routing. In addition, the package has been designed in a modular fashion, allowing users to build their own model. To our knowledge, this is the first HBV version and R hydro-modeling package to combine these two features.

General modeling philosophy

According to Bergström and Lindström (2015), the HBV model was inspired in the works developed in the early 1970s by Nash and Sutcliffe (1970), O'Connell et al. (1970), and Mandeville et al. (1970). The primary objective of this model was operational: to forecast streamflow discharge for the Swedish hydropower industry. This overriding requirement dictated the characteristics of the model: it should not be too complex but physically sound; the input data should conform to standard Swedish meteorological measurements; the number of free parameters should be kept to a minimum; and it should be easy to understand.

The above features and lessons learned over more than two decades (Bergström, 1991) resulted in a hydrological model composed of four modules: (1) a temperature index model with an air temperature-based precipitation partitioning algorithm; (2) a soil moisture routine with a nonlinear empirical algorithm to account for abstractions, actual evaporation, and antecedent conditions; (3) a bucket-type model (many variants exist up to now) to simulate the catchment storage effect; and (4) a transfer function to adjust the timing of the hydrograph to the observed discharge.

To date, the model has not only been used in operational hydrology but also in scientific research. Konz and Seibert (2010) used the HBV-Light version in three alpine catchments in Switzerland and Austria to show the value of glacier mass balances in constraining uncertainty in the parameter estimation of conceptual models such as HBV. Ali et al. (2018) also applied HBV-Light to evaluate model performance in a climate change context in the snow- and ice-dominated Hunza River basin in the Karakoram Mountains, Pakistan. Finger et al. (2015) compared model performance in simulations of increasing complexity for glacier mass balance and streamflow at the outflow of three Swiss watersheds. Stahl et al. (2008) used HBV-CE to estimate streamflow sensitivity to different climate change scenarios in British Columbia, Canada. Staudinger et al. (2017) studied the variation of water storage with elevation in 21 Swiss alpine and pre-alpine catchments using four different methods: water balance analysis, flow recession analysis, calibration of the HBV model, and calibration of a transfer function hydrograph separation model using stable isotope observations. In another interesting application, Ren et al. (2018) combined HBV with a Bayesian neural network to improve seasonal water supply forecasting in the Yarkant River basin, Central Asia. Therefore, the original conception of HBV and its evolution have made it a longstanding multipurpose tool for a diverse and dynamic user community.

Modules and equations

Models are based on a perceptual conception of the basin's functioning. This perception leads to the decision of the equations (hydrological processes) and the construction of a conceptual model (Beven, 2012). In the **HBV.IANIGLA** model, these first two stages have already been decided, as the equations and coding are in the package, but the user still has a choice on how the watershed or glacier will be discretized (in terms of land use and spatial aggregation) and on how the different modules will be assembled. This decision should be guided by the objective of the project, the knowledge of the

hydrological driving process at the chosen modeling scale, and the data available not only for the implementation but also for the model evaluation.

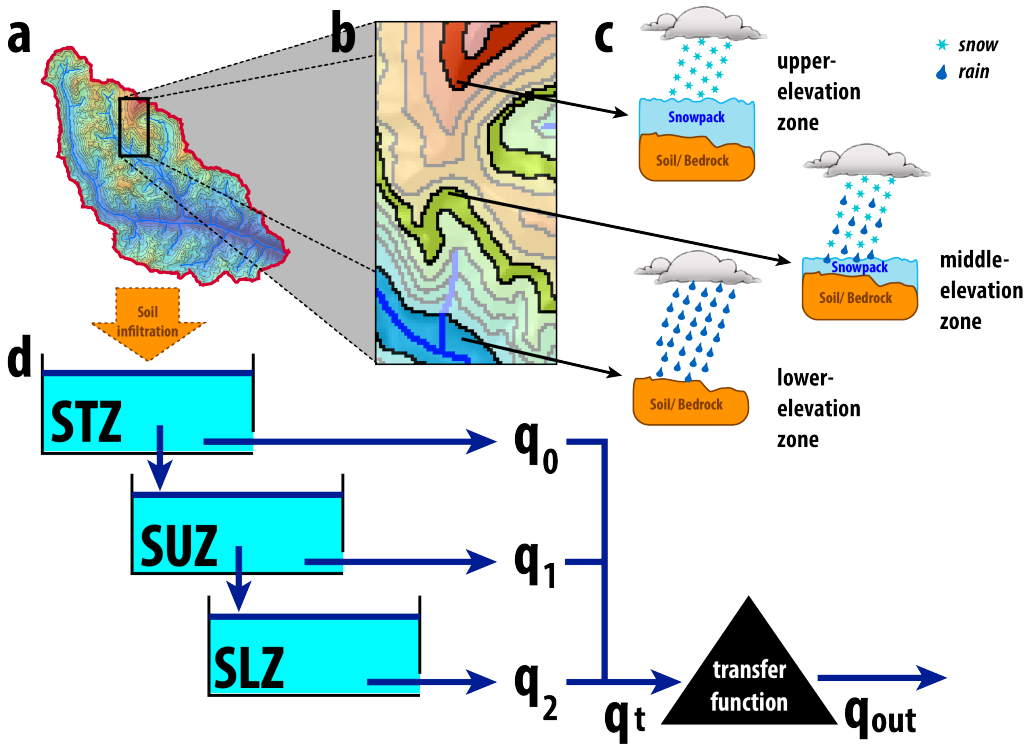


Figure 1: Example of **HBV.IANIGLA** module assembly in a mountain basin. To account for snow accumulation and snowmelt, the basin has been discretized into elevation bands (**a** and **b**). Each of these polygons has snow and soil routine (**c**), the effective soil recharge, weighted according to the relative area of the elevation band, is passed to the bucket model (**d**). Finally, the river runoff timing is adjusted by a triangular transfer function.

The following lines describe the modules that must be assembled to build a complete **HBV.IANIGLA** hydrological model. There are three other functions within the package: `PET`, `Pecip_model`, and `Temp_model`. The first function contains a potential evapotranspiration model that provides a simple and straightforward way to calculate one of the inputs to the soil routine. However, for real-world applications we strongly recommend the use of the specialized **Evapotranspiration** package (Guo et al., 2020). The other two functions are linear models to extrapolate air temperature and precipitation records. Since we consider that their use is straightforward, we refer the user to the package manual.

Snow and ice melt models – `SnowGlacier_HBV()`

Precipitation is considered to be either snow or rain, depending on whether the temperature is above or below a threshold temperature *Tr* (ºC).

$$\begin{aligned} P_{rain} &= P \qquad if \quad T_{air} > Tr \\ P_{snow} &= P * SFCF \qquad if \quad T_{air} \leq Tr \end{aligned} \tag{1}$$

After partitioning, the snowfall is corrected using the `SFCF` parameter to account for the under-capture effect of the precipitation gauge on snow events.

This function uses a temperature index approach for snow and glacier melt simulation. This kind of approximation has been widely used in snow hydrology and glaciology, and different formulations have emerged (Hock, 2003; Seibert and Vis, 2012; Braun and Renner, 1992). The temperature index formulation takes into account the strong correlation between snow line retreat and accumulated temperatures above a certain threshold (with typical values around 0ºC). Hence although many authors have proposed more complex formulations (e.g., HBV-Light uses a refreezing and liquid retention factor) or even a radiation term (Pellicciotti et al., 2005), this empirical formulation must be parsimonious to avoid problems of overparameterization (Kirchner, 2006).

$$Melt = (T_{air} - Tt) * f_x \qquad if \quad T_{air} > Tt, \tag{2}$$

where T_{air} is the measured or estimated air temperature, Tt is the melting temperature, and f_x is a generic expression of melting factors for snow, clean, or debris-covered ice.

If the air temperature is above the threshold (Tt), melting occurs at a rate proportional to the melting factor (f_x). Both the temperature threshold and the melting factor are parameters that must be calibrated by the user. Note that the time units depend on the resolution of the input data. Although the examples shown in this article are in a daily time step, the model can be used in the hourly or monthly resolution. In the next lines, we will describe in detail the different arguments of the function.

The `model` argument presents three options:

1. **Temperature index model**: this model is described by equation 2. Here, the user can apply the most common and recommended set of temperature index formulations.
2. **Temperature index model with variable snow cover area**: this option is an attempt to offer, within the package, the same temperature index model as in the Snowmelt Runoff Model (DeWalle and Rango, 2008). However, this routine has certain limitation: the snow cover series forces the model to simulate a total effective value (e.g., snow water equivalent), which is not in-line with the original idea of modeling in elevation bands, where average values are expected.
3. **Temperature index model with a variable glacier area**: this routine explicitly takes into account the change in glacier area. Since the automatic reduction of glacier area forces the simulation to the observed values, the user should evaluate the correspondence between the simulated and observed mass balances.

The package documentation contains all the necessary information (vignettes with reproducible examples included) to correctly construct the `inputData` argument. The data matrix must not contain missing values (`NA`'s) because **HBV.IANIGLA** is a continuous hydrological model, meaning that it simulates all the variables in every time step.

The initial conditions of the model are (`initCond`):

1. **Initial snow water equivalent**: this is a state variable, whose initial value will be used in the first loop. Unless field data is available, it is recommended to use a zero value. Because uncertainties are common in the initial state variables of the model, it is recommended to use a warm-up period (between one and two years in daily time step modeling). If the period covered by the data is very limited, these same values can be used as calibration parameters.
2. **Numeric integer indicating the surface type**: 1: clean ice; 2: soil; 3: debris-covered ice. **HBV.IANIGLA** uses this argument to know which parameters (`param` argument) to look for. It also constrains the function output.
3. **Area of the glacier(s) (in the elevation band) relative to the basin**: this is required only if the surface is a clean or debris-covered glacier. The area is used to scale the total amount of water produced (rainfall plus melted water) according to the area of the polygon in the basin. Thus, if the area of this portion of the glacier corresponds to 5% of the basin area, a value of 0.05 should be assigned.

The last argument is a numeric vector that stores the parameter values (`param`) of the modules. For debris-covered glaciers, a dummy value for the clean glacier melting factor (f_{ic}) must be supplied. This value will not be used internally but simplifies the calibration exercise when working in a basin with both types of glaciers.

It should be noted that this function allows the construction of a single and lumped simulation. In order to develop the model for the example shown in figure 1, it will be necessary to build the model by running the function once per every elevation band (see examples in `vignette(package = "HBV.IANIGLA")`).

Soil routine – `Soil_HBV()`

This routine is based on an empirical formulation that takes into account actual evapotranspiration, antecedent conditions, and effective soil infiltration. This relationship is described by the so-called beta function (Bergström and Lindström, 2015),

$$Inf = (Melt + Rainfall) * \left(\frac{SM}{FC}\right)^{\beta}, \tag{3}$$

where Inf is the soil box infiltration, SM is the soil moisture state variable, and β is a nonlinear parameter between the total amount of water entering the soil box, soil moisture storage, and runoff generation. This equation is not unique among bucket-type hydrological models. A similar formulation can be found in the VIC model (Liang et al., 1994). **HBV.IANIGLA** assumes that all evapotranspiration occurs from the soil box, so this function implicitly accounts for all abstractions:

$$E_{act} = E_{pot} * min\left(\frac{SM}{FC * LP}; 1.00\right), \tag{4}$$

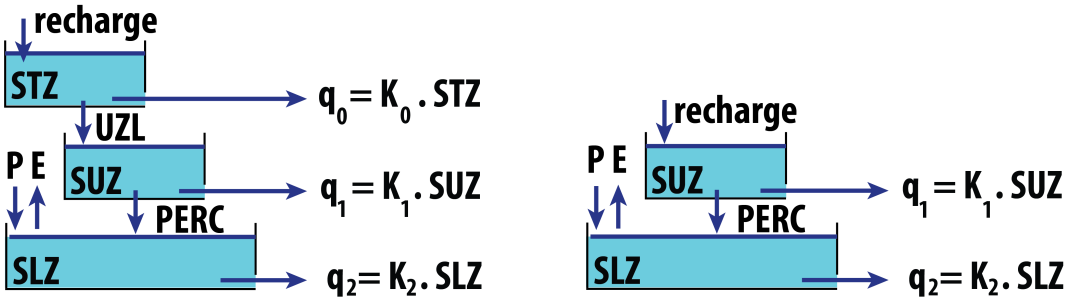
where E_{act} is the actual evapotranspiration, E_{pot} is the potential evapotranspiration, FC is the soil box water capacity parameter, and LP is a reduction factor.

This type of relationship between potential, actual evapotranspiration, and soil moisture content has been found by Zhang et al. (2003) in eastern Asia, suggesting that despite its empirical formulation, in some places, it could have some physical meaning. Finally, and similar to the snow and ice melt modules, this routine represents a single and lumped simulation.

Routine module – `Routing_HBV()`

After infiltration, the water follows several complex pathways to streams (McDonnell, 2003). A detailed description and modeling of these water pathways requires field data and measurements that are generally not available. An early engineering-based solution to this issue was to consider this multi-causal delay as a water storage effect at the catchment scale (Dooge, 1973). This practical modeling approach could be seen as series of linearly interconnected and interrelated reservoirs (Sivapalan and Blöschl, 2017).

The current **HBV.IANIGLA** (version 0.2.1) has five different bucket formulations, which are selected by changing the model argument number (figure 2). To solve the time step change in the bucket water storage, we used the explicit finite difference form of the mass balance equation over a discrete-time step (Beven, 2012). Although the general solution has been implemented for a single linear reservoir (figure 3), we provide solutions for the five-bucket models.



(a) Model 1 **(b)** Model 2

Figure 2: Diagrams for two of the five available bucket models. The reader will find all the diagrams in the help menu (`?Routing_HBV`).

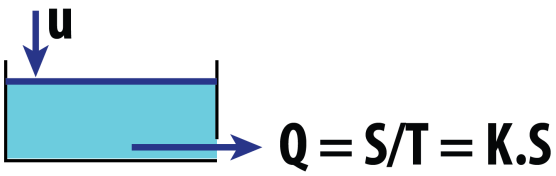


Figure 3: General outline for a water reservoir.

$$\frac{dS}{dt} = u - Q \qquad \textit{mass balance equation} \tag{5}$$

$$Q = K * S \qquad \textit{continuity equation} \tag{6}$$

from (6),

$$S = \frac{Q}{K} = T * Q \tag{7}$$

we replace (7) into (5),

$$T * \frac{dQ}{dt} = u * Q$$

In discrete time steps, we use the explicit finite difference form,

$$\frac{Q_t - Q_{t-\Delta t}}{\Delta t} = \frac{u_t - Q_{t-\Delta t}}{T}$$

$$Q_t = \frac{\Delta t}{T} * u_t + (1 - \frac{\Delta t}{T}) * Q_{t-\Delta t}$$

$$a = (1 - \frac{\Delta t}{T})$$

$$b = \frac{\Delta t}{T}$$

$$Q_t = a * Q_{t-\Delta t} + b * u_t \tag{8}$$

$$\frac{S_t - S_{t-\Delta t}}{\Delta t} = u_t - Q_{t-\Delta t}$$

$$S_t = S_{t-\Delta t} + \Delta t * (u_t - Q_{t-\Delta t}) \tag{9}$$

Glacier routine module – `Glacier_Disch()`

Following an approach similar to previous routines, we adopt bucket storage and release scheme (Jansson et al., 2003). In **HBV.IANIGLA**, we use the approach proposed by Stahl et al. (2008) for the HBV-EC model, employed to estimate glacier and streamflow responses to future climate scenarios in the Bridge River Basin (British Columbia, Canada). The glacier outflow is calculated as:

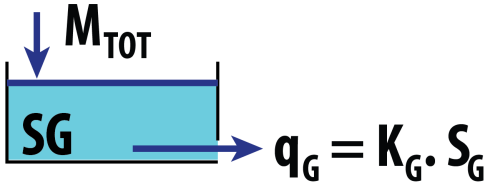


Figure 4: The glacier runoff release (precipitation plus snow and ice melt) is modeled as a linear water reservoir with a variable storage coefficient (K_G), which is a function of the snow water equivalent above the ice body.

$$K_G = K_{Gmin} + dK_G * \exp(SWE/AG) \tag{10}$$

$$q_G = K_G * S_G, \tag{11}$$

where K_G is the actual glacier outflow coefficient, K_{Gmin} a minimum storage release coefficient, dK_G the maximum glacier outflow increment, SWE the total snow water equivalent over the glacier, AG a scaling parameter, S_G the glacier water storage, and q_G the glacier runoff.

Note that the storage coefficient is a function of a minimum coefficient (denoting poor drainage conditions on the glacier), the snow water equivalent, and a calibration parameter. When the snowpack is at its maximum value, drainage occurs at a minimum rate, the opposite occurs in the late summer when all the snow on the glacier has melted.

For the resolution of the time step change, we also use the explicit finite difference formulation of the mass balance equation.

Transfer Function – `UH()`

To represent the runoff routing in streams, we provide a single parameter triangular function. This parameter is calibrated to adjust the timing of the simulated river discharge,

$$Q = \sum_{i=1}^{B_{max}} Q_{t-i+1} * b_i, \tag{12}$$

where B_{max} is the base of the triangular weighting function, b_i is the weight for the i^{th} step, and Q_{t-i+1} is the sum of the glacier and soil bucket runoff.

Computation times

The **HBV.IANIGLA** functions were written using **Rcpp** (Eddelbuettel, 2013; Eddelbuettel et al., 2019), a package that extends the R language using C++. This approach combines the speed and efficiency of C++, a compiled language, with the powerful interactive environment of R (see table 1), a language where it is easy to implement specific hydrological workflows (from data retrieval to results analysis) in a single environment (Slater et al., 2019).

min	lq	mean	median	uq	max
1.79	1.95	2.65	2.01	2.19	55.81

Table 1: Summary of computation times (in *milliseconds*) over 1000 runs of the `glacier_hbv` function (see `vignette("alerce_mass_balance")`) . The glacier was discretized into 8 elevation bands ($\sim$ 100 m range). The model was built with the modules `Temp_model`, `Precip_model`, and `SnowGlacier` and was run on a daily time step over a period of almost 9 years (from 2010-01-01 to 2018-05-30). The analysis was performed on a CPU with an Intel Core i7-4790 processor at 3.60GHz, on a 64-bit OS running Ubuntu 18.04 using the **microbechmark** package (Mersmann, 2019).

Speed is an important issue for hydrological models, as it allows the user to perform not only uncertainty and sensitivity analysis in reasonable times, but also to apply demanding optimization algorithms such as **DEoptim** (Ardia et al., 2016) or different model structures. This is a recommended practice in the field of hydrological modeling (Beven, 2006, 2008; Pianosi et al., 2016). In addition, the package only depends on **Rcpp** (v 0.12.0), a fact that supports its long-term maintenance.
If the reader is interested in comparing the computation times of different R hydrological models we recommend the work of Astagneau et al. (2020).

Case studies

Lumped synthetic catchment

As a first attempt at applying **HBV.IANIGLA**, a synthetic lumped catchment (the simplest hydrological model) is used to introduce the construction of the model and to present a basin discharge calibration exercise.

Initially, the dataset containing: date, air temperature, precipitation, potential evapotranspiration, and the catchment outflow is loaded, and then the model construction is conducted (from top to bottom).

```
library(HBV.IANIGLA)

# load the lumped catchment dataset
data("lumped_hbv")

# take a look at our dataset
head(lumped_hbv)
summary(lumped_hbv)
```

For a basin without glaciers, the `SnowGlacier` module is used only with soil as the underlying surface. In this exercise, we provide the correct initial conditions and parameters for all modules except the `Routing_HBV` function. Consistent with the development of hydrologic models, we build our

model in a top-down direction, from precipitation to streamflow routing (note that most hydrological books are structured in the same way).

```
# consider the SnowGlacier module to take into account
# precipitation partitioning and the snow accumulation/melting.
snow_module <-
  SnowGlacier_HBV(model = 1,
                  inputData = as.matrix( lumped_hbv[ , c(2, 3)] ),
                  initCond = c(20, 2),
                  param = c(1.20, 1.00, 0.00, 2.5) )

# now pass rainfall plus snowmelt to
# the soil routine. Note that we are using the PET series,

soil_module <-
  Soil_HBV(model = 1,
           inputData = cbind(snow_module[ , "Total"], lumped_hbv[ , "PET(mm/d)"]),
           initCond = c(100, 1),
           param = c(200, 0.8, 1.15) )
```

The actual evapotranspiration, soil moisture, and recharge series are obtained from the last module. Subsequently, the recharge is incorporated into the routing function. Recall that the routing parameters (`param` argument) are not calibrated.

```
routing_module <-
  Routing_HBV(model = 1,
              lake = F,
              inputData = as.matrix(soil_module[ , "Rech"]),
              initCond = c(0, 0, 0),
              param = c(0.9, 0.01, 0.001, 0.5, 0.01) )

# finally apply the transfer function in order to adjust
# the hydrograph timing

tf_module <-
  round(
    UH(model = 1,
       Qg = routing_module[ , "Qg"],
       param = c(1.5) ),
    2)

# plot the "true" and simulated hydrographs

library(ggplot2)

ggplot(data = data.frame(date = lumped_hbv[ , "Date"],
                         qsim = tf_module,
                         qobs = lumped_hbv[ , "qout(mm/d)"]),
       aes(x = date)) +
  geom_line(aes(y = qsim), col = "dodgerblue") +
  geom_line(aes(y = qobs), col = "red") +
  xlab(label = " ") + ylab(label = "q(mm/d)") +
  theme_minimal() +
  scale_x_date(date_breaks = "1 year") +
  scale_y_continuous(breaks = seq(0, 15, 2.5)) +
  theme(
    title = element_text(color = "black", size = 12, face = "bold"),
    axis.title.x = element_text(color = "black", size = 12, face = "bold"),
    axis.title.y = element_text(color = "black", size = 12, face = "bold"),
    legend.text = element_text(size = 11),
    axis.text = element_text(size = 11),
    axis.text.x = element_text(angle = 90) )
```

It is required to manually change the `Routing_HBV` parameters to approximate the simulation to the observed basin discharge. Users will find in the package vignette (`vignette("lumped_basin")`)

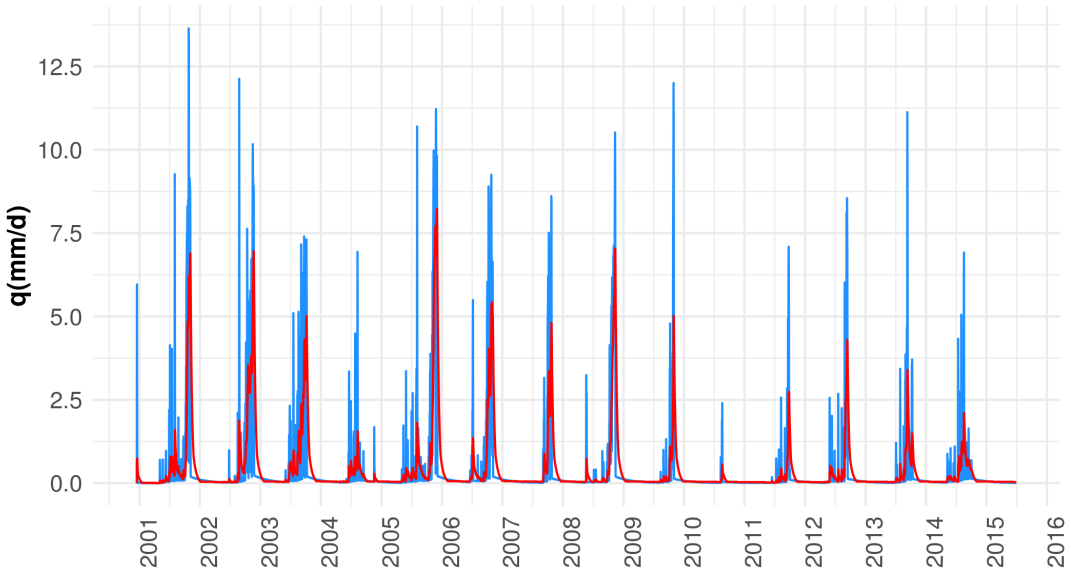


Figure 5: Observed (red) and simulated (blue) basin discharge. Note that the simulation does not precisely reproduce the observed basin discharge, suggesting that the model needs some calibration.

more information on this example, including the construction of an HBV model as a function and how to run a sensitivity analysis.

Semi-distributed glacier mass balance

In mountain areas with scarce meteorological information, temperature index models are widely used to simulate snow and ice melting (Hock, 2003; Konz and Seibert, 2010; Finger et al., 2015; Ayala et al., 2017). Since air temperature is the most readily available meteorological data in remote areas, the temperature index approach has been widely used in glaciological and hydrological modeling (Ohmura, 2001). This package has been built with the `SnowGlacier_HBV` function, a module that uses this empirical approach to simulate snow, clean ice, and debris-covered melting.

In this section, we simulate the glacier mass balance for the Alerce glacier. Located on Monte Tronador (41.15° S ; 71.88° W), nearby the border between Argentina and Chile in the Andes of Northern Patagonia, Alerce is a medium-size mountain glacier with an area of about 2.33 km^2 that ranges between 1629 and 2358 masl showing a SE aspect (Ruiz et al., 2017; IANIGLA-ING, 2018).

Since 2013, the Alerce glacier has been part of the monitoring network of the *National Glacier Inventory* (IANIGLA-ING, 2010). Measurements are conducted following the glaciological method for seasonal mass balance computation (Kaser et al., 2003). Puerto Montt precipitation (*Dirección General de Aguas, Chile*) and Bariloche air temperature (*Servicio Meteorológico Nacional - Argentina*) were used as meteorological records to simulate the annual mass balance of the glacier (`data(alerce_data)`). When calibrating the model parameters, simulations showing an annual mass balance in the range of $MB \pm 400\,mm$ were considered acceptable. MB is the annual surface mass balance of the glacier.

```
## load the dataset
data(alerce_data)

  # now extract
    meteo_data   <- alerce_data[["meteo_data"]]   # meteorological forcing series
    mass_balance <- alerce_data[["mass_balance"]] # annual glacier mass balances
    mb_dates     <- alerce_data[["mb_dates"]]     # fix seasonal dates
    gl_topo      <- alerce_data[["topography"]]   # elevation bands

    z_tair   <- alerce_data[["station_height"]][1] # topo. elev. air temp.
    z_precip <- alerce_data[["station_height"]][2] # topo. elev. precip.
```

To evaluate the topographic effect on surface mass balance (derived from field measurements), the glacier was discretized into elevation bands. To solve this problem, a semi-distributed glacier surface mass balance model (`glacier_hbv`), an aggregation function (`agg_mb` - since measurements and

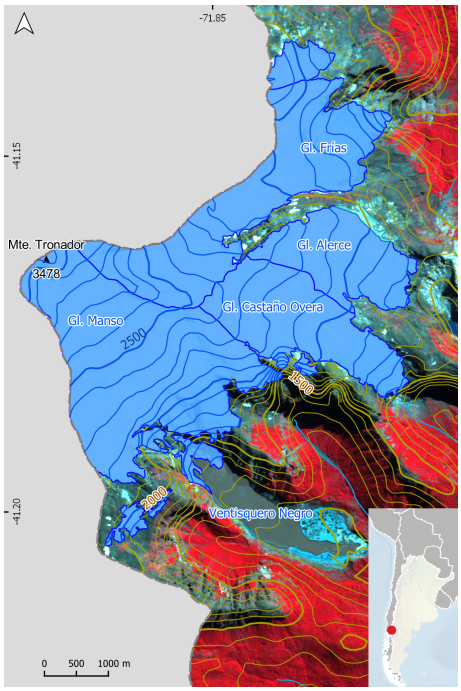


Figure 6: Satellite image of the Monte Tronador showing the location of the main glaciers. The Alerce glacier, located in the eastern sector, is one of the smallest glaciers at Monte Tronador.

simulations are on different temporal scales), and a goodness-of-fit function (`my_gof`) were constructed. The definition of this functions are included in `vignette("alerce_mass_balance")`.

In the following code lines, the sampling strategy for finding acceptable parameter sets is indicated.

```
# air temperature model
tair_range <- rbind(
  t_grad  = c(-9.8, -2)
)

# precip model
precip_range <- rbind(
  p_grad  = c(5, 25)
)

# glacier module
glacier_range <- rbind(
  sfcf = c(1, 2),
  tr   = c(0, 3),
  tt   = c(0, 3),
  fm   = c(1, 4),
  fi   = c(4, 8)
)

## aggregate them in a matrix
param_range <-
  rbind(
    tair_range,
    precip_range,
    glacier_range
  )
```

In the next step, we generate the random parameter sets:

```
# set the number of model runs that you want to try
n_run <- 10000

# build the matrix
```

```
n_it <- nrow(param_range)

param_sets <- matrix(NA_real_, nrow = n_run, ncol = n_it)

colnames(param_sets) <- rownames(param_range)

set.seed(123) # just for reproducibility
for(i in 1:n_it){

  param_sets[ , i] <- runif(n = n_run,
                            min = param_range[i, 1],
                            max = param_range[i, 2]
  )

}
```

Now, we combine the functions and extract our best simulations.

```
# goodness of fit vector
gof <- c()

# make a loop
for(i in 1:n_run){

  # run the model
  glacier_sim <- glacier_hbv(topography = gl_topo,
                             meteo = meteo_data,
                             z_topo = c(z_tair, z_precip),
                             param_tair = param_sets[i, rownames(tair_range)],
                             param_precip = param_sets[i, rownames(precip_range) ],
                             param_ice = param_sets[i, rownames(glacier_range)] )

  # aggregate the simulation
  annual_mb <- agg_mb(x = glacier_sim,
                      start_date = as.Date( mb_dates$winter[-4] ),
                      end_date = as.Date( mb_dates$winter[-1] ) - 1 )

  # compare the simulations with measurements
  gof[i] <- my_gof(obs_upp = mass_balance$upp,
                   obs_lwr = mass_balance$lwr,
                   sim = annual_mb[ , 3])

  rm(glacier_sim, annual_mb)
}

param_sets <- cbind(param_sets, gof)

# we apply a filter
param_subset <- subset(x = param_sets, subset = gof == 3)
```

Once we have the subsetted our parameter matrix, we run the simulations to obtain a mean value (one per year).

```
# now we run the model again to get our simulations
n_it <- nrow(param_subset)

mb_sim <- matrix(NA_real_, nrow = 3, ncol = n_it)

for(i in 1:n_it){
  glacier_sim <- glacier_hbv(topography = gl_topo,
                             meteo = meteo_data,
                             z_topo = c(z_tair, z_precip),
                             param_tair = param_subset[i, rownames(tair_range)],
                             param_precip = param_subset[i, rownames(precip_range) ],
                             param_ice = param_subset[i, rownames(glacier_range)] )
```

```
  annual_mb <- agg_mb(x = glacier_sim,
                      start_date = as.Date( mb_dates$winter[-4] ),
                      end_date = as.Date( mb_dates$winter[-1] ) - 1 )

  mb_sim[ , i] <- annual_mb[ , 3]

  rm(i, glacier_sim, annual_mb)

}

# now we are going to make a data frame with the mean surface mass balance simulation
mean_sim <- cbind( mass_balance,
                   "mb_sim" =  rowMeans(mb_sim)    )

# make the plot
library(ggplot2)
g1 <-
  ggplot(data =  mean_sim, aes(x = year)) +
  geom_pointrange(aes(y = `mb(mm we)`, ymin = `lwr`, color = 'obs',
                      ymax = `upp` ), size = 1,  fill = "white", shape = 21) +
  geom_point(aes(y = `mb_sim`, fill = 'sim'), shape = 23,
             size = 3) +
  geom_hline(yintercept = 0) +
  scale_y_continuous(limits = c(-1500, 500), breaks = seq(-1500, 500, 250) ) +
  scale_color_manual(name = '', values = c('obs' = 'blue') ) +
  scale_fill_manual(name = '', values = c('sim' = 'red') ) +
  ggtitle('') +
  xlab('') + ylab('mb (mm we)') +
  theme_minimal() +
  theme(
    title = element_text(color = "black", size = 12, face = "bold"),
    axis.title.x = element_text(color = "black", size = 12, face = "bold"),
    axis.title.y = element_text(color = "black", size = 12, face = "bold"),
    legend.text = element_text(size = 11),
    axis.text = element_text(size = 11))
```

Summary

In this study, we present the **HBV.IANIGLA** package, a modular version of HBV hydrological model that incorporates routines for clean and debris-covered glacier modeling. We explain its modeling principles and philosophy; address the package modules and related equations; and reinforce the importance of C++ code to speed up calculations, a characteristic that facilitates sensitivity and uncertainty analysis. To our knowledge, this is the first freely available, open-source modular version of the HBV model that incorporates routines for glacier surface mass balance modeling (clean and debris-covered ice).

We present two examples. The first one consists of a synthetic case to show how to build a model. This is the simplest hydrological modeling case and should be use to understand how to concatenate the package functions and how a numerical hydrological model works. The `vignette("lumped_basin")` also illustrates the importance of sensitivity analysis and shows how it can be easily done in R. In fact, any kind of sensitivity or uncertainty analysis (Pianosi et al., 2016) could be incorporated into **HBV.IANIGLA**. The second example, a real-world glacier surface mass balance estimation, was selected to show the use of the glacier module in a semi-distributed case. This module could be of interest to the glaciological community, extending the use of the R language to other scientific communities.

The new package version (0.2.1) documentation has been greatly improved in relation to previous versions, not only by clarifying some aspects of the existing function's documentation but also by adding six vignettes with reproducible examples (see vignette(package = "HBV.IANIGLA")).

The modular design of the package allows the use of various spatio-temporal scales with dissimilar objectives in the same environment (e.g., real-time streamflow forecasting, hydrological model teaching, or glacier mass balance simulation). The different modules can be combined with other

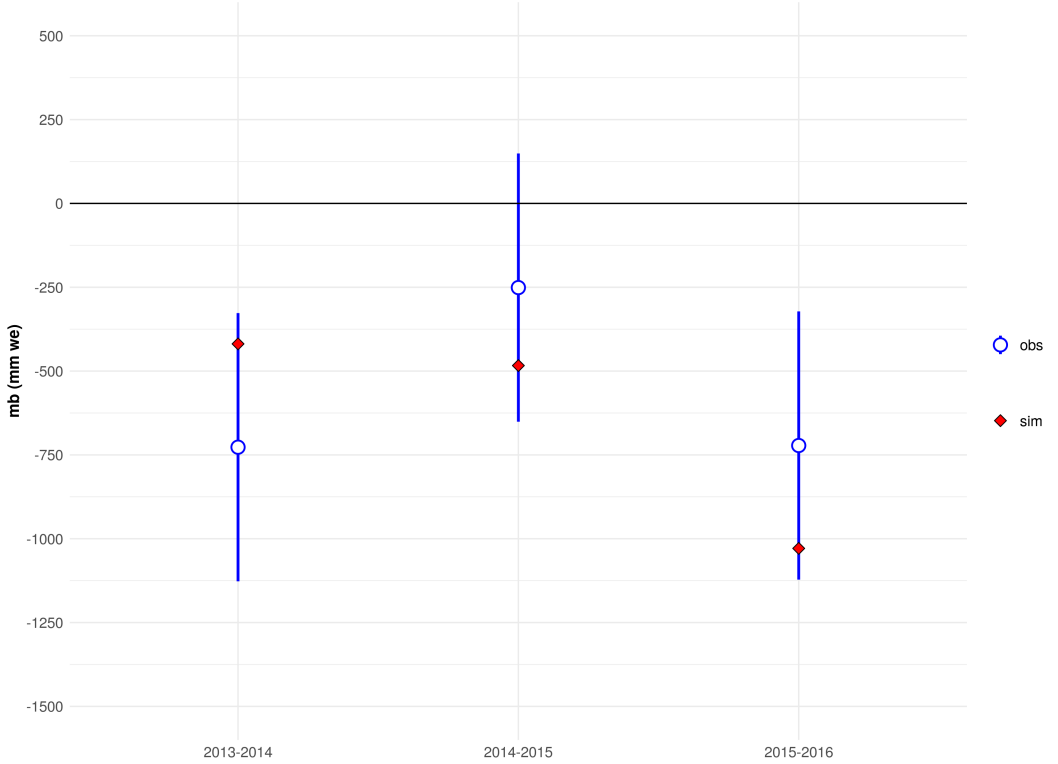


Figure 7: Annual mass balances for the period April 2013 - March 2016. All acceptable simulations lies between the observed uncertainty bounds (despite the fact that we have plot just the mean value).

R-related hydrological packages (e.g., **Evapotranspiration**, **DEoptim**, **topmodel**) or functions (Guo et al., 2019; Ardia et al., 2016; Buytaert, 2018).

HBV.IANIGLA can also be combined with packages such as **tidyverse**, **sp**, **raster**, **hydroGOF**, or **plotly** to build a single environmental hydrological workflow (Wickham, 2019; Pebesma and Bivand, 2017; Hijmans, 2017; Mauricio Zambrano-Bigiarini, 2017; Sievert et al., 2019). Thus, a hydrological project can be developed from the beginning to the end in the R environment, facilitating reproducible and repeatable research (Hutton et al., 2016; Ceola et al., 2015). This type of model design opens up the possibility for applications beyond the Andes region as well as to incorporate new functions, such as modules, to explicitly considering the dynamics of glaciers (Huss et al., 2010).

The package functions were built under generic classes (numeric vectors and matrices). This is an aspect where future improvements can be made. Since **HBV.IANIGLA** is available in modules, it could be greatly enhanced using the object-oriented programming (OOP) paradigm. In doing so, the model could represent an object with properties (e.g., areas, polygons, elevations, among others), and the HBV routines as part of the methods (functional OOP - S4 types). These methods may also include (but not limited to): sensitivity and uncertainty analysis, automatic plotting of results, and temporal aggregation functionality. Even some methods could be recycled from the **hydroToolkit** OOP package (Toum, 2020).

The package could also be improved by adding some GUI functionality keeping in mind that in the words of Chambers (2017), *...extending R is about contributing to the language through applications designed for a wider audience than the package author itself. Moreover, this objective should not be a target by its own, but a part or a piece of a bigger project directed to solve real world problems*.

Acknowledgements

E.T. acknowledges support from CONICET Argentina through a full-time PhD scholarship. The constructive suggestions from two anonymous reviewers were very useful for improving the article, the package documentation, and the vignettes, and are greatly appreciated.

Bibliography

A. F. Ali, C.-d. Xiao, X.-p. Zhang, M. Adnan, M. Iqbal, and G. Khan. Projection of future streamflow of the Hunza River Basin, Karakoram Range (Pakistan) using HBV hydrological model. *J. Mt. Sci.*, 15(10):2218–2235, Oct. 2018. ISSN 1993-0321. doi: 10.1007/s11629-018-4907-4. URL https://doi.org/10.1007/s11629-018-4907-4. 0272. [p379]

F. T. Andrews, B. F. W. Croke, and A. J. Jakeman. An open software environment for hydrological model assessment and development. *Environmental Modelling & Software*, 26(10):1171–1185, Oct. 2011. ISSN 1364-8152. doi: 10.1016/j.envsoft.2011.04.006. URL http://www.sciencedirect.com/science/article/pii/S1364815211001046. [p378]

D. Ardia, K. Mullen, B. Peterson, and J. Ulrich. *DEoptim: Global Optimization by Differential Evolution*, 2016. URL https://CRAN.R-project.org/package=DEoptim. R package version 2.2-4. [p384, 390]

P. C. Astagneau, G. Thirel, O. Delaigue, J. H. A. Guillaume, J. Parajka, C. C. Brauer, A. Viglione, W. Buytaert, and K. J. Beven. Hydrology modelling R packages: a unified analysis of models and practicalities from a user perspective. *Hydrology and Earth System Sciences Discussions*, pages 1–48, Oct. 2020. ISSN 1027-5606. doi: 10.5194/hess-2020-498. URL https://hess.copernicus.org/preprints/hess-2020-498/. Publisher: Copernicus GmbH. [p384]

A. Ayala, F. Pellicciotti, N. Peleg, and P. Burlando. Melt and surface sublimation across a glacier in a dry environment: distributed energy-balance modelling of Juncal Norte Glacier, Chile. *Journal of Glaciology*, 63(241):803–822, Oct. 2017. ISSN 0022-1430, 1727-5652. doi: 10.1017/jog.2017.46. 0236. [p386]

S. Bergström. Principles and Confidence in Hydrological Modelling. *Hydrology Research*, 22(2):123–136, Apr. 1991. ISSN 0029-1277. doi: 10.2166/nh.1991.0009. URL https://doi.org/10.2166/nh.1991.0009. [p379]

S. Bergström and G. Lindström. Interpretation of runoff processes in hydrological modelling—experience from the HBV approach. *Hydrol. Process.*, 29(16):3535–3545, July 2015. ISSN 1099-1085. doi: 10.1002/hyp.10510. URL http://onlinelibrary.wiley.com/doi/10.1002/hyp.10510/abstract. [p378, 379, 381]

K. Beven. A manifesto for the equifinality thesis. *Journal of Hydrology*, 320(1):18–36, Mar. 2006. ISSN 0022-1694. doi: 10.1016/j.jhydrol.2005.07.007. URL http://www.sciencedirect.com/science/article/pii/S002216940500332X. 0126. [p384]

K. Beven. *Environmental Modelling: An Uncertain Future?* CRC Press, London, 1 edition edition, Sept. 2008. ISBN 978-0-415-45759-0. [p384]

K. J. Beven. *Rainfall - Runoff Modelling*. Wiley, Chichester, 2 edition edition, Mar. 2012. ISBN 978-0-470-86671-9. [p379, 382]

L. N. Braun and C. B. Renner. Application of a conceptual runoff model in different physiographic regions of Switzerland. *Hydrological Sciences Journal*, 37(3):217–231, June 1992. ISSN 0262-6667. doi: 10.1080/02626669209492583. URL https://doi.org/10.1080/02626669209492583. [p378, 380]

W. Buytaert. *topmodel: Implementation of the Hydrological Model TOPMODEL in R*, 2018. URL https://CRAN.R-project.org/package=topmodel. R package version 0.7.3. [p378, 390]

W. Buytaert, D. Reusser, S. Krause, and J.-P. Renaud. Why can't we do better than Topmodel? *Hydrol. Process.*, 22(20):4175–4179, Sept. 2008. ISSN 1099-1085. doi: 10.1002/hyp.7125. URL http://onlinelibrary.wiley.com/doi/10.1002/hyp.7125/abstract. 0208. [p378]

S. Ceola, B. Arheimer, E. Baratti, G. Blöschl, R. Capell, A. Castellarin, J. Freer, D. Han, M. Hrachowitz, Y. Hundecha, C. Hutton, G. Lindström, A. Montanari, R. Nijzink, J. Parajka, E. Toth, A. Viglione, and T. Wagener. Virtual laboratories: new opportunities for collaborative water science. *Hydrology and Earth System Sciences*, 19(4):2101–2117, Apr. 2015. ISSN 1027-5606. doi: https://doi.org/10.5194/hess-19-2101-2015. URL https://www.hydrol-earth-syst-sci.net/19/2101/2015/hess-19-2101-2015.html. [p390]

J. M. Chambers. *Extending R*. CRC Press, Dec. 2017. ISBN 978-1-4987-7572-4. Google-Books-ID: kxxjDAAAQBAJ. [p390]

L. Coron, G. Thirel, O. Delaigue, C. Perrin, and V. Andréassian. The suite of lumped GR hydrological models in an R package. *Environmental Modelling and Software*, 94:166–171, 2017. doi: 10.1016/j.envsoft.2017.05.002. [p378]

L. Coron, O. Delaigue, G. Thirel, C. Perrin, and C. Michel. *airGR: Suite of GR Hydrological Models for Precipitation-Runoff Modelling*, 2020. URL https://CRAN.R-project.org/package=airGR. R package version 1.4.3.65. [p378]

D. R. DeWalle and A. Rango. *Principles of Snow Hydrology*. Cambridge University Press, Cambridge, UK ; New York, July 2008. ISBN 978-0-521-82362-3. [p381]

J. Dooge. *Linear Theory of Hydrologic Systems*. Agricultural Research Service, U.S. Department of Agriculture, 1973. [p382]

D. Eddelbuettel. *Seamless R and C++ Integration with Rcpp*. Use R! Springer-Verlag, New York, 2013. ISBN 978-1-4614-6867-7. URL https://www.springer.com/gp/book/9781461468677. [p384]

D. Eddelbuettel, R. Francois, J. Allaire, K. Ushey, Q. Kou, N. Russell, D. Bates, and J. Chambers. *Rcpp: Seamless R and C++ Integration*, 2019. URL https://CRAN.R-project.org/package=Rcpp. R package version 1.0.2. [p384]

D. Finger, M. Vis, M. Huss, and J. Seibert. The value of multiple data set calibration versus model complexity for improving the performance of hydrological models in mountain catchments. *Water Resour. Res.*, 51(4):1939–1958, Apr. 2015. ISSN 1944-7973. doi: 10.1002/2014WR015712. URL http://onlinelibrary.wiley.com/doi/10.1002/2014WR015712/abstract. [p379, 386]

A. R. Groos and C. Mayer. *glacierSMBM: Glacier Surface Mass Balance Model*, 2017. URL https://CRAN.R-project.org/package=glacierSMBM. R package version 0.1. [p378]

D. Guo, S. Westra, and T. Peterson. *Evapotranspiration: Modelling Actual, Potential and Reference Crop Evapotranspiration*, 2019. URL https://CRAN.R-project.org/package=Evapotranspiration. R package version 1.14. [p390]

D. Guo, S. Westra, and T. Peterson. *Evapotranspiration: Modelling Actual, Potential and Reference Crop Evapotranspiration*, 2020. URL https://CRAN.R-project.org/package=Evapotranspiration. R package version 1.15. [p380]

R. J. Hijmans. *raster: Geographic Data Analysis and Modeling*, 2017. URL https://CRAN.R-project.org/package=raster. R package version 2.6-7. [p390]

R. Hock. Temperature index melt modelling in mountain areas. *Journal of Hydrology*, 282(1):104–115, Nov. 2003. ISSN 0022-1694. doi: 10.1016/S0022-1694(03)00257-9. URL http://www.sciencedirect.com/science/article/pii/S0022169403002579. [p380, 386]

M. Huss, G. Jouvet, D. Farinotti, and A. Bauder. Future high-mountain hydrology: a new parameterization of glacier retreat. *Hydrol. Earth Syst. Sci.*, 14(5):815–829, May 2010. ISSN 1607-7938. doi: 10.5194/hess-14-815-2010. URL https://www.hydrol-earth-syst-sci.net/14/815/2010/. 0019. [p390]

C. Hutton, T. Wagener, J. Freer, D. Han, C. Duffy, and B. Arheimer. Most computational hydrology is not reproducible, so is it really science? *Water Resources Research*, 52(10):7548–7555, 2016. ISSN 1944-7973. doi: 10.1002/2016WR019285. URL https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1002/2016WR019285. [p390]

IANIGLA-ING. Inventario Nacional de Glaciares y Ambiente Periglacial:Fundamentos y Cronograma de Ejecución. Technical report, CONICET, Mendoza, 2010. URL http://www.glaciaresargentinos.gob.ar/wp-content/uploads/legales/fundamentos_cronograma_ejecucion.pdf. [p386]

IANIGLA-ING. IANIGLA-Inventario Nacional de Glaciares. 2018. Informe de las subcuencas de los ríos Manso, Villegas y Foyel. Cuenca de los ríos Manso y Puelo. IANIGLA-CONICET, Ministerio de Ambiente y Desarrollo Sustentable de la Nación. Technical report, IANIGLA, 2018. URL http://www.glaciaresargentinos.gob.ar. [p386]

P. Jansson, R. Hock, and T. Schneider. The concept of glacier storage: a review. *Journal of Hydrology*, 282(1):116–129, Nov. 2003. ISSN 0022-1694. doi: 10.1016/S0022-1694(03)00258-0. URL http://www.sciencedirect.com/science/article/pii/S0022169403002580. 0229. [p383]

G. Kaser, A. Fountain, and P. Jansson. A manual for monitoring the mass balance of mountain glaciers. Manual 59, UNESCO, Paris, 2003. URL http://ulis2.unesco.org/images/0012/001295/129593E.pdf. 0396. [p386]

J. W. Kirchner. Getting the right answers for the right reasons: Linking measurements, analyses, and models to advance the science of hydrology. *Water Resources Research*, 42(3), Mar. 2006. ISSN 1944-7973. doi: 10.1029/2005WR004362. URL https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2005WR004362. 0216. [p380]

M. Konz and J. Seibert. On the value of glacier mass balances for hydrological model calibration. *Journal of Hydrology*, 385(1):238–246, May 2010. ISSN 0022-1694. doi: 10.1016/j.jhydrol.2010.02.025. URL http://www.sciencedirect.com/science/article/pii/S0022169410000958. 0189. [p379, 386]

V. Krysanova, A. Bronstert, and D.-I. Müller-Wohlfeil. Modelling river discharge for large drainage basins: from lumped to distributed approach. *Hydrological Sciences Journal*, 44(2):313–331, Apr. 1999. ISSN 0262-6667. doi: 10.1080/02626669909492224. URL https://doi.org/10.1080/02626669909492224. [p378]

X. Liang, D. P. Lettenmaier, E. F. Wood, and S. J. Burges. A simple hydrologically based model of land surface water and energy fluxes for general circulation models. *Journal of Geophysical Research: Atmospheres*, 99(D7):14415–14428, 1994. ISSN 2156-2202. doi: https://doi.org/10.1029/94JD00483. URL https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/94JD00483. _eprint: https://agupubs.onlinelibrary.wiley.com/doi/pdf/10.1029/94JD00483. [p382]

A. N. Mandeville, P. E. O'Connell, J. V. Sutcliffe, and J. E. Nash. River flow forecasting through conceptual models part III - The Ray catchment at Grendon Underwood. *Journal of Hydrology*, 11 (2):109–128, Aug. 1970. ISSN 0022-1694. doi: 10.1016/0022-1694(70)90098-3. URL http://www.sciencedirect.com/science/article/pii/0022169470900983. [p379]

M. H. Masiokas, A. Rabatel, A. Rivera, L. Ruiz, P. Pitte, J. L. Ceballos, G. Barcaza, A. Soruco, F. Bown, E. Berthier, I. Dussaillant, and S. MacDonell. A Review of the Current State and Recent Changes of the Andean Cryosphere. *Frontiers in Earth Science*, 8, 2020. ISSN 2296-6463. doi: 10.3389/feart.2020.00099. URL https://www.frontiersin.org/articles/10.3389/feart.2020.00099/full. 0424. [p378]

Mauricio Zambrano-Bigiarini. *hydroGOF: Goodness-of-Fit Functions for Comparison of Simulated and Observed Hydrological Time Series*, 2017. URL https://CRAN.R-project.org/package=hydroGOF. R package version 0.3-10. [p390]

J. J. McDonnell. Where does water go when it rains? Moving beyond the variable source area concept of rainfall-runoff response. *Hydrological Processes*, 17(9):1869–1875, 2003. ISSN 1099-1085. doi: https://doi.org/10.1002/hyp.5132. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/hyp.5132. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/hyp.5132. [p382]

O. Mersmann. *microbenchmark: Accurate Timing Functions*, 2019. URL https://CRAN.R-project.org/package=microbenchmark. R package version 1.4-7. [p384]

P. Metcalfe, K. Beven, and J. Freer. Dynamic TOPMODEL: A new implementation in R and its sensitivity to time and space steps. *Environmental Modelling & Software*, 72:155–172, Oct. 2015. ISSN 1364-8152. doi: 10.1016/j.envsoft.2015.06.010. URL http://www.sciencedirect.com/science/article/pii/S1364815215001735. 0306. [p378]

J. E. Nash and J. V. Sutcliffe. River flow forecasting through conceptual models part I — A discussion of principles. *Journal of Hydrology*, 10(3):282–290, Apr. 1970. ISSN 0022-1694. doi: 10.1016/0022-1694(70)90255-6. URL http://www.sciencedirect.com/science/article/pii/0022169470902556. 0004. [p379]

P. E. O'Connell, J. E. Nash, and J. P. Farrell. River flow forecasting through conceptual models part II - The Brosna catchment at Ferbane. *Journal of Hydrology*, 10(4):317–329, June 1970. ISSN 0022-1694. doi: 10.1016/0022-1694(70)90221-0. URL http://www.sciencedirect.com/science/article/pii/0022169470902210. [p379]

A. Ohmura. Physical basis for the temperature-based melt-index method. *Journal of Applied Meteorology*, 40(4):753–761, Apr. 2001. ISSN 0894-8763. doi: 10.1175/1520-0450(2001)040<0753:PBFTTB>2.0.CO;2. URL https://doi.org/10.1175/1520-0450(2001)040<0753:PBFTTB>2.0.CO;2. [p386]

J. Parajka and G. Blöschl. The value of MODIS snow cover data in validating and calibrating conceptual hydrologic models. *Journal of Hydrology*, 358(3):240–258, Sept. 2008. ISSN 0022-1694. doi: 10.1016/j.jhydrol.2008.06.006. URL http://www.sciencedirect.com/science/article/pii/S0022169408002862. 0199. [p379]

E. Pebesma and R. Bivand. *sp: Classes and Methods for Spatial Data*, 2017. URL https://CRAN.R-project.org/package=sp. R package version 1.2-5. [p390]

F. Pellicciotti, B. Brock, U. Strasser, P. Burlando, M. Funk, and J. Corripio. An enhanced temperature-index glacier melt model including the shortwave radiation balance: development and testing for Haut Glacier d'Arolla, Switzerland. *Journal of Glaciology*, 51(175):573–587, 2005. ISSN 0022-1430, 1727-5652. doi: 10.3189/172756505781829124. Publisher: Cambridge University Press. [p380]

F. Pianosi, K. Beven, J. Freer, J. W. Hall, J. Rougier, D. B. Stephenson, and T. Wagener. Sensitivity analysis of environmental models: A systematic review with practical workflow. *Environmental Modelling & Software*, 79:214–232, May 2016. ISSN 1364-8152. doi: 10.1016/j.envsoft.2016.02.008. URL http://www.sciencedirect.com/science/article/pii/S1364815216300287. 0324. [p384, 389]

W. W. Ren, T. Yang, C. S. Huang, C. Y. Xu, and Q. X. Shao. Improving monthly streamflow prediction in alpine regions: integrating HBV model with Bayesian neural network. *Stoch Environ Res Risk Assess*, 32(12):3381–3396, Dec. 2018. ISSN 1436-3259. doi: 10.1007/s00477-018-1553-x. URL https://doi.org/10.1007/s00477-018-1553-x. 0359. [p379]

L. Ruiz, E. Berthier, M. Viale, P. Pitte, and M. H. Masiokas. Recent geodetic mass balance of Monte Tronador glaciers, northern Patagonian Andes. *The Cryosphere*, 11(1):619–634, Feb. 2017. ISSN 1994-0424. doi: 10.5194/tc-11-619-2017. URL https://www.the-cryosphere.net/11/619/2017/. 0237. [p386]

J. Seibert and M. J. P. Vis. Teaching hydrological modeling with a user-friendly catchment-runoff-model software package. *Hydrol. Earth Syst. Sci.*, 16(9):3315–3325, Sept. 2012. ISSN 1607-7938. doi: 10.5194/hess-16-3315-2012. URL https://www.hydrol-earth-syst-sci.net/16/3315/2012/. 0021. [p378, 379, 380]

C. Sievert, C. Parmer, T. Hocking, S. Chamberlain, K. Ram, M. Corvellec, and P. Despouy. *plotly: Create Interactive Web Graphics via 'plotly.js'*, 2019. URL https://CRAN.R-project.org/package=plotly. R package version 4.9.0. [p390]

M. Sivapalan and G. Blöschl. The Growth of Hydrological Understanding: Technologies, Ideas, and Societal Needs Shape the Field. *Water Resources Research*, 53(10):8137–8146, 2017. ISSN 1944-7973. doi: 10.1002/2017WR021396. URL https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1002/2017WR021396. 0413. [p382]

L. J. Slater, G. Thirel, S. Harrigan, O. Delaigue, A. Hurley, A. Khouakhi, I. Prodoscimi, C. Vitolo, and K. Smith. Using R in hydrology: a review of recent developments and future directions. *Hydrology and Earth System Sciences Discussions*, pages 1–33, Feb. 2019. ISSN 1027-5606. doi: https://doi.org/10.5194/hess-2019-50. URL https://www.hydrol-earth-syst-sci-discuss.net/hess-2019-50/. [p378, 384]

K. Stahl, R. D. Moore, J. M. Shea, D. Hutchinson, and A. J. Cannon. Coupled modelling of glacier and streamflow response to future climate scenarios. *Water Resour. Res.*, 44(2):W02422, Feb. 2008. ISSN 1944-7973. doi: 10.1029/2007WR005956. URL http://onlinelibrary.wiley.com/doi/10.1029/2007WR005956/abstract. 0013. [p378, 379, 383]

M. Staudinger, M. Stoelzle, S. Seeger, J. Seibert, M. Weiler, and K. Stahl. Catchment water storage variation with elevation. *Hydrological Processes*, 31(11):2000–2015, 2017. ISSN 1099-1085. doi: 10.1002/hyp.11158. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/hyp.11158. [p379]

E. Toum. *hydroToolkit: Hydrological Tools for Handling Hydro-Meteorological Data from Argentina and Chile*, 2020. R package version 0.1.1. [p390]

E. Toum. *HBV.IANIGLA: Modular Hydrological Model*, 2021. URL https://CRAN.R-project.org/package=HBV.IANIGLA. 0.2.0. [p378]

A. Viglione and J. Parajka. *TUWmodel: Lumped Hydrological Model for Education Purposes*, 2016. URL https://CRAN.R-project.org/package=TUWmodel. R package version 0.1-8. [p378]

H. Wickham. *tidyverse: Easily Install and Load the 'Tidyverse'*, 2019. URL https://CRAN.R-project.org/package=tidyverse. R package version 1.3.0. [p390]

Y. Zhang, T. Ohata, K. Ersi, and Y. Tandong. Observation and estimation of evaporation from the ground surface of the cryosphere in eastern Asia. *Hydrological Processes*, 17(6):1135–1147, Apr. 2003. ISSN 1099-1085. doi: 10.1002/hyp.1183. URL http://onlinelibrary.wiley.com/doi/10.1002/hyp.1183/abstract. [p382]

Ezequiel Toum
IANIGLA-CONICET
Av. Ruiz Leal s/n Parque General San Martin - Mendoza
Argentina
etoum@mendoza-conicet.gob.ar

Mariano H. Masiokas
IANIGLA-CONICET
Av. Ruiz Leal s/n Parque General San Martin - Mendoza
Argentina
mmasiokas@mendoza-conicet.gob.ar

Ricardo Villalba
IANIGLA-CONICET
Av. Ruiz Leal s/n Parque General San Martin - Mendoza
Argentina
ricardo@mendoza-conicet.gob.ar

Pierre Pitte
IANIGLA-CONICET
Av. Ruiz Leal s/n Parque General San Martin - Mendoza
Argentina
pierrepitte@mendoza-conicet.gob.ar

Lucas Ruiz
IANIGLA-CONICET
Av. Ruiz Leal s/n Parque General San Martin - Mendoza
Argentina
lruiz@mendoza-conicet.gob.ar

The R Package smicd: Statistical Methods for Interval-Censored Data

by Paul Walter

Abstract The package allows the use of two new statistical methods for the analysis of interval-censored data: 1) direct estimation/prediction of statistical indicators and 2) linear (mixed) regression analysis. Direct estimation of statistical indicators, for instance, poverty and inequality indicators, is facilitated by a non parametric kernel density algorithm. The algorithm is able to account for weights in the estimation of statistical indicators. The standard errors of the statistical indicators are estimated with a non parametric bootstrap. Furthermore, the package offers statistical methods for the estimation of linear and linear mixed regression models with an interval-censored dependent variable, particularly random slope and random intercept models. Parameter estimates are obtained through a stochastic expectation-maximization algorithm. Standard errors are estimated using a non parametric bootstrap in the linear regression model and by a parametric bootstrap in the linear mixed regression model. To handle departures from the model assumptions, fixed (logarithmic) and data-driven (Box-Cox) transformations are incorporated into the algorithm.

Introduction

Interval-censored or grouped data occurs when only the lower A_{k-1} and upper A_k interval bounds (A_{k-1}, A_k) of a variable are observed, and its true value remains unknown. Instead of measuring the variable of interest on a continuous scale, for instance, income data, the scale is divided into n_k intervals. The variable k $(1 \leq k \leq n_k)$ indicates in which of the n_k intervals an observation falls into. This leads to a loss of information since the shape of the distribution within the intervals remains unknown. In the field of survey statistics, asking for interval-censored data is often done in order to avoid item non-response and thus increase data quality. Item non-response is avoided because interval-censored data offers a higher level of data privacy protection (Hagenaars and Vos, 1988; Moore and Welniak, 2000). Among others, popular surveys and censuses that collect interval-censored data are the German Microcensus (Statistisches Bundesamt, 2017), the Colombian census (Departamento Administrativo Nacional De Estadística, 2005), and the Australian census (Australian Bureau of Statistics, 2011). While item non-response is reduced or avoided, the statistical analysis of the data requires more elaborate mathematical methods. Even statistical indicators that are easily calculated for metric data, e.g., the mean, cannot be estimated using standard formulas (Fahrmeir et al., 2016). Also, estimating linear and linear mixed regression models, which are applied in many fields of statistics, requires advanced statistical methods when the dependent variable is interval censored. Therefore, the presented R package implements three major functions: `kdeAlgo()` to estimate statistical indicators (e.g., the mean) from interval-censored data, `semLm()`, and `semLme()` to estimate linear and linear mixed regression models with an interval-censored dependent variable. The package code and the open-source contribution guidelines for the package are available on GitHub. Potential code contributions, feature requests, and bugs can be reported there by creating issues.

For the estimation of statistical indicators from interval-censored data, different approaches are described in the literature. These approaches can be broadly categorized into four groups: Estimation on the midpoints (Fahrmeir et al., 2016), linear interpolation of the distribution function, non parametric modeling via splines (Berger and Escobar, 2016), and fitting a parametric distribution function to the censored data (Dagum, 2008; McDonald, 1984; Bandourian et al., 2002). Some of these methods are implemented in R packages available on the Comprehensive R Archive Network (CRAN). The method of linear interpolation is implemented for the estimation of quantiles in the R package **actuar** (Goulet et al., 2020; Dutang et al., 2008). The package also enables the estimation of the mean on the interval midpoints. Fitting a parametric distribution to interval-censored data can be done by using the R package **fitdistrplus** (Delignette-Muller et al., 2020; Delignette-Muller and Dutang, 2015).

In survey statistics, interval-censored data is often collected for income or wealth variables. Thus, the performance of the above-mentioned methods is commonly evaluated by simulation studies that rely on data that follows some kind of income distribution. The German statistical office (DESTATIS) uses the method of linear interpolation for the estimation of statistical indicators from interval-censored income data collected by the German Microcensus. This approach gives the same results as assuming a uniform distribution within the income intervals. Estimation results are reasonably accurate if the estimated indicators do not depend on the whole shape of the distribution, e.g., the median (Lenau and Münnich, 2016). Fitting a parametric distribution to the data enables the estimation of indicators that rely on the whole shape of the distribution. This method works well when the data is censored to

only a few equidistant intervals (Lenau and Münnich, 2016). Non parametric modeling via splines shows especially good results for a high number of intervals in ascending order (Lenau and Münnich, 2016). However, according to Lenau and Münnich (2016), all of the above-mentioned methods show large biases and variances when the estimation is based on a small number of intervals. Therefore, a novel kernel density estimation (KDE) algorithm is implemented in the **smicd** package that overcomes the drawbacks of the previously mentioned methods (Walter, 2019, 2020). The algorithm bases the estimation of statistical indicators on pseudo samples that are drawn from a fitted non parametric distribution. The method automatically adapts to the shape of the true unknown distribution and provides reliable estimates for different interval-censoring scenarios. It can be applied via the function `kdeAlgo()`.

Similar to the direct estimation of statistical indicators from interval-censored data, there exists a variety of ad-hoc approaches and explicitly formulated mathematical methods for the estimation of linear regression models with an interval-censored dependent variable. The following methods and approaches are used for handling interval-censored dependent variables within linear regression models: Ordinary least squares (OLS) regression on the midpoints (Thompson and Nelson, 2003), ordered logit- or probit-regression (McCullagh, 1980), and regression methodology formulated for left-, right-, and interval-censored data (Tobin, 1958; Rosett and Nelson, 1975; Stewart, 1983). All of these methods are implemented in different R packages available on CRAN. OLS regression on the midpoints is applicable by using the `lm()` function from the **stats** package (R Core Team, 2020), ordered logit regression is implemented in the **MASS** package (Ripley, 2019; Venables and Ripley, 2002), and interval regression is implemented in the **survival** (Therneau, 2020; Therneau and Grambsch, 2000) package.

While OLS regression on the midpoints of the intervals is easily applied, it comes with the disadvantage of giving biased estimation results (Cameron, 1987). This approach disregards the uncertainty stemming from the unknown true distribution of the data within the intervals, and therefore, leads to biased parameter estimates. Its performance relies on the number of intervals, and estimation results are only comparable to more advanced methods when the number of intervals is very large (Fryer and Pethybridge, 1972). Conceptualizing the model as an ordered logit or probit regression is feasible by treating the dependent variable as an ordered factor variable (McCullagh, 1980). However, this approach also neglects the unknown distribution of the data within the intervals. Furthermore, the predicted values are not on a continuous scale but are in terms of the probability of belonging to a certain group. To overcome these disadvantages and obtain unbiased estimation results Stewart (1983) introduces regression methodology for models with an interval-censored dependent variable. Walter (2019) further develops his approach and introduces a novel stochastic expectation-maximization (SEM) algorithm for the estimation of linear regression models with an interval-censored dependent variable that is implemented in the **smicd** package. The model parameters are unbiasedly estimated as long as the model assumptions are fulfilled. The function `semLm()` provides the SEM algorithm and enables the use of fixed (logarithmic) and data-driven (Box-Cox) transformations (Box and Cox, 1964). The Box-Cox transformation automatically adapts to the shape of the data and transforms the dependent variable in order to meet the model assumption.

In order to analyze longitudinal or clustered data (e.g., students within schools), linear mixed regression models are applicable. These kinds of models control for the correlated structure of the data by including random effects in addition to the usual fixed effects. In order to deal with an interval-censored dependent variable in linear mixed regression models, there are several approaches described in the literature. Linear mixed regression models, just like linear regression models, can be estimated on the interval midpoints of the censored-dependent variable. Furthermore, conceptualizing the model as an ordered logit or probit regression model is feasible (Agresti, 2010). These approaches inherit the same advantages and disadvantages as previously discussed. Linear mixed regression on the midpoints can be applied by the **lme4** (Bates et al., 2020b, 2015) or **nlme** (Pinheiro et al., 2020) package and the ordered logit regression is implemented in the **ordinal** package (Christensen, 2019). To my knowledge, there are no R packages for the estimation of linear mixed regression models with an interval-censored dependent variable. Therefore, the package **smicd** contains the SEM algorithm proposed by Walter (2019) for the estimation of linear mixed regression models with an interval-censored dependent variable. If the model assumptions are fulfilled, the method gives unbiased estimation results. The function `semLme()` enables the estimation of the regression parameters, and it also allows for the usage of the logarithmic and Box-Cox transformation in order to fulfill the model assumptions (Gurka et al., 2006).

The paper is structured into two main sections. Section 2.2 deals with the direct estimation of statistical indicators from interval-censored data, whereas Section 2.3 introduces linear and linear mixed regression models with an interval-censored dependent variable. Both sections have been divided into three subsections: first, the statistical methodology is introduced, then the core functions of the **smicd** package are presented, and finally, illustrative examples with two different data sets are provided. In Section 2.4, the main results are summarized, and an outlook is given.

Direct estimation of statistical indicators

In the following three subsections, the methodology for the direct estimation of statistical indicators from interval-censored data is introduced, the core functionality of the function `kdeAlgo()` is presented, and statistical indicators are estimated using the synthetic EU-SILC (European Union Statistics on Income and Living Conditions) data set from Austria.

Methodology: Direct estimation of statistical indicators

In order to estimate statistical indicators from interval-censored data, the proposed algorithm generates metric pseudo samples of an interval-censored variable. These pseudo samples can be used to estimate any statistical indicator. They are drawn from a non parametrically estimated kernel density. Kernel density estimation was first introduced by (Rosenblatt, 1956) and (Parzen, 1962). By its application, the density $f(x)$ of a continuous independently and identically distributed random variable is estimated without assuming any distributional shape of the data. The estimator is defined as:

$$\hat{f}_h(x) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right), \qquad i = 1, \ldots, n,$$

where $K(\cdot)$ is a kernel function, $h > 0$ the bandwidth, and $x = \{x_1, x_2, \ldots, x_n\}$ denotes a sample of size n. The performance of the estimator is determined by the optimal choice of h. The selection of an optimal h is widely discussed in the literature; see Jones et al. (1996); Loader (1999); Zambom and Dias (2012). When working with interval-censored data, a standard KDE cannot be applied since x is not observed on a continuous scale. Nevertheless, its unobserved true distribution is of continuous form. As an ad hoc solution, the density $\hat{f}_h(x)$ can be estimated based on the interval midpoints. The resulting density estimate will be spiky unless the bandwidth is sufficiently large. A large bandwidth, however, leads to a loss of information (Wang and Wertelecki, 2013). Therefore, Walter (2019) proposes an iterative KDE algorithm for density estimation from interval-censored data. The approach is based on Groß et al. (2017), who introduce a similar KDE algorithm in a two-dimensional setting with an equidistant interval width. Walter (2019) shows that the algorithm can be adjusted to one-dimensional data with an arbitrary class width. For the estimation of linear and non-linear statistical indicators, the unknown distribution of x has to be reconstructed by using the observed interval $k = \{k_1, k_2, \ldots, k_n\}$ that an observation falls into. From Bayes' theorem (Bayes, 1763), it follows that the conditional distribution of $(x|k)$ is:

$$\pi(x|k) \propto \pi(k|x)\, \pi(x),$$

with $\pi(k|x)$ is defined by a product of a Dirac distribution $\pi(k|x) = \prod_{i=1}^{n} \pi(k_i|x_i)$ with

$$\pi(k_i|x_i) = \begin{cases} 1 & \text{if } A_{k_i-1} \leq x_i \leq A_{k_i}, \\ 0 & \text{else,} \end{cases}$$

for $i = 1, \ldots, n$. Since $\pi(x)$ is unknown, it is replaced by a kernel density estimate $\hat{f}_h(x)$.

Estimation and computational details

To fit the model, pseudosamples of x_i are drawn from the conditional distribution

$$\pi(x_i|k_i) \propto \mathbf{I}\left(A_{k_i-1} \leq x_i \leq A_{k_i}\right) f(x_i),$$

where $\mathbf{I}(\cdot)$ denotes the indicator function. The conditional distribution of $\pi(x_i|k_i)$ is given by the product of a uniform distribution and density $f(x_i)$. As the density is unknown, it is replaced by an estimate $\hat{f}_h(x)$, which is obtained by the KDE. In particular, x_i is repeatedly drawn from the given interval $\left(A_{k_i-1}, A_{k_i}\right)$ by using the current density estimate $\hat{f}_h(x)$ as a sampling weight. The explicit steps of the iterative algorithm as given in Walter (2019) are stated below:

1. Use the midpoints of the intervals as pseudo $\tilde{x}_i$ for the unknown x_i. Estimate a pilot estimate of $\hat{f}_h(x)$ by applying KDE. Note: Choose a sufficiently large bandwidth h in order to avoid rounding spikes.
2. Evaluate $\hat{f}_h(x)$ on an equal-spaced grid $G = \{g_1, \ldots, g_j\}$ with grid points $g_1, \ldots, g_j$. The width of the grid is denoted by δ_g. It is given by

$$\delta_g = \frac{|A_0 - A_{n_k}|}{j - 1},$$

and the grid is defined as:

$$G = \{g_1 = A_0, g_2 = A_0 + \delta_g, g_3 = A_0 + 2\delta_g, \ldots, g_{j-1} = A_0 + (j-2)\,\delta_g, g_j = A_{n_k}\}.$$

3. Sample from $\pi(x|k)$ by drawing randomly from $G_k = \{g_j | g_j \in (A_{k-1}, A_k)\}$ with sampling weights $\hat{f}_h(\tilde{x}_i)$ for $k = 1, \ldots, n_k$. The sample size for each interval is given by the number of observations within each interval. Obtain $\tilde{x}_i$ for $i = 1, \ldots, n$.
4. Estimate any statistical indicator of interest $\hat{I}$ using $\tilde{x}_i$.
5. Recompute the density $\hat{f}_h(x)$ using the pseudo samples $\tilde{x}_i$ obtained in iteration Step 3.
6. Repeat Steps 2-5, with $B^{(KDE)}$ burn-in and $M^{(KDE)}$ additional iterations.
7. Discard the $B^{(KDE)}$ burn-in iterations and estimate the final $\hat{I}$ by averaging the obtained $M^{(KDE)}$ estimates.

For open-ended intervals, e.g., $(15000, +\infty)$, the upper bound has to be replaced by a finite number. Walter (2019) shows through model-based simulations that a value of three times the value of the lower bound $(15000, 45000)$ gives appropriate estimation results when working with income data.

The variance of the statistical indicators is estimated by bootstrapping. Bootstrap methods were first introduced by Efron (1979). These methods serve as an estimation procedure when the variance cannot be stated as a closed-form solution (Shao and Tu, 1995). While bootstrapping avoids the problem of the non-availability of a closed-form solution, it comes with the disadvantage of long computational times. In the package, a non parametric bootstrap that accounts for the additional uncertainty coming from the interval-censored data is implemented. This non parametric bootstrap is introduced in Walter (2019).

Core functionality: Direct estimation of statistical indicators

The presented KDE algorithm is implemented in the function `kdeAlgo()` (see Table 1). The arguments and default settings of `kdeAlgo()` are briefly summarized in Table 2. The function gives back an S3 object of class `"kdeAlgo"`. A detailed explanation of all components of a `"kdeAlgo"` object can be found in the package documentation. The generic functions `plot()` and `print()` can be applied to `"kdeAlgo"` objects to output the main estimation results (see Table 1). In the next section, the function `kdeAlgo()` is used to estimate a variety of statistical indicators from interval-censored EU-SILC data, and its arguments are explained in more detail.

Table 1: Implemented functions for the direct estimation of statistical indicators.

Function Name	Description
`kdeAlgo()`	Estimates the statistical indicators and its standard errors from interval-censored data
`plot()`	Plots convergence of the estimated statistical indicators and estimated density of the pseudo $\tilde{x}_i$
`print()`	Prints the estimated statistical indicators and its standard errors

Example: Direct estimation of statistical indicators

To demonstrate the function `kdeAlgo()`, the equivalized household income and the corresponding household sample weight from the Austrian synthetic EU-SILC survey data set are used. The data set is included in the **laeken** package (Alfons et al., 2020; Alfons and Templ, 2013). Its synthetic nature makes it unusable for inferential statistics. However, the data set has the advantage over the scientific use file by being freely available which enables the easy reproducibility of the presented example. Since the total disposable household income is measured on a continuous scale, it is censored to 22 intervals for demonstration purposes. For a realistic censoring scheme, the interval bounds are chosen such that they closely follow the interval bounds used in the German Microcensus from 2013 (Statistisches Bundesamt, 2014). The German Microcensus is a representative household survey that covers 830,000 persons in 370,000 households (1% of the German population) in which income is only collected as an interval-censored variable (Statistisches Bundesamt, 2016).

In a first step, the variable total disposable household income called `hhincome_net` is interval-censored according to 22 intervals using the function `cut()`. The vector of interval bounds is called `intervals`, and the newly obtained interval-censored income variable is called `c.hhincome`.

Table 2: Arguments of function `kdeAlgo()`.

Argument	Description	Default
`xclass`	Interval-censored variable	
`classes`	Numeric vector of interval bounds	
`threshold`	Threshold used for poverty indicators (% of the median of the target variable)	`0.6`
`burnin`	Number of burn-in iterations $B^{(KDE)}$	`80`
`samples`	Number of additional iterations $M^{(KDE)}$	`400`
`bootstrap.se`	If TRUE, standard errors of the statistical indicators are estimated	`FALSE`
`b`	Number of bootstraps for the estimation of the standard errors	`100`
`bw`	Smoothing bandwidth used	`"nrd0"`
`evalpoints`	Number of evaluation grid points	`4000`
`adjust`	Bandwidth multiplier $bw = adjust * bw$	`1`
`custom_indicator`	A list of user-defined statistical indicators	`NULL`
`upper`	If upper bound of the upper interval is $+\infty$, e.g., $(15000, +\infty)$, then $+\infty$ is replaced by $15000 * upper$	`3`
`weights`	Survey weights	`NULL`
`oecd`	Household weights of equivalence scale	`NULL`

```
R> intervals <- c(
+    0, 150, 300, 500, 700, 900, 1100, 1300, 1500, 1700, 2000, 2300, 2600, 2900,
+    3200, 3600, 4000, 4500, 5000, 5500, 6000, 7500, Inf
+  )
R> c.hhincome <- cut(hhincome_net, breaks = intervals)
```

In order to get a descriptive overview of the distribution of the censored income data, the function `table()` is applied.

```
R> table(c.hhincome)
c.hhincome
          (0,150]         (150,300]         (300,500]
               66               113               280
        (500,700]         (700,900]     (900,1.1e+03]
              462              1137              1433
(1.1e+03,1.3e+03] (1.3e+03,1.5e+03] (1.5e+03,1.7e+03]
             2040              1811              1671
  (1.7e+03,2e+03]   (2e+03,2.3e+03] (2.3e+03,2.6e+03]
             2006              1383               849
(2.6e+03,2.9e+03] (2.9e+03,3.2e+03] (3.2e+03,3.6e+03]
              508               389               242
  (3.6e+03,4e+03]   (4e+03,4.5e+03]   (4.5e+03,5e+03]
              158               107                61
  (5e+03,5.5e+03]   (5.5e+03,6e+03]   (6e+03,7.5e+03]
               21                18                52
    (7.5e+03,Inf]
               17
```

Most incomes are in interval $(1100, 1300]$, and only 17 incomes are in the upper interval. For the estimation of the statistical indicators, the function `kdeAlgo()` of the **smicd** package is called with the following arguments.

```
R> Indicators <- kdeAlgo(
+    xclass = c.hhincome, classes = intervals,
+    bootstrap.se = TRUE, custom_indicator =
+     list(
+      quant05 = function(y, threshold, weights) {
+       wtd.quantile(y, probs = 0.05, weights)
```

```
+      },
+      quant95 = function(y, threshold, weights) {
+       wtd.quantile(y, probs = 0.95, weights)
+      }
+     ),
+    weights = hhweight
+  )
```

The variable `c.hhincome` is assigned to the argument `xclass`, and the vector of interval bounds `intervals` is assigned to the argument `classes`. The default settings of the arguments `burnin`, `samples`, `bw`, `evalpoints`, `adjust`, and `upper` are retained. Simulation results from Walter (2019) and Groß et al. (2017) show that these settings give good results when working with income data. Changing these arguments has an impact on the performance of the KDE algorithm. As default, the statistical indicators: mean, Gini coefficient, headcount ratio (HCR), the quantiles (10%, 25%, 50%, 75%, 90%), the poverty gap (PGAP), and the quintile share ratio (QSR) are estimated (Gini, 1912; Foster et al., 1984). The HCR and PGAP rely on a poverty threshold. The default choice of the `threshold` argument is 60% of the median of the target variable, as suggested by Eurostat (2014). Besides the mentioned indicators, any other statistical indicator can be estimated via the argument `custom_indicator`. In the example, the argument is assigned a list that holds functions to estimate the 5% and 95% quantile. The custom indicators must depend on the target variable, the threshold (even if it is not needed for the specified indicator), and optionally on the weights argument if the estimation of a weighted indicator is required. To estimate the standard errors of all indicators, `bootstrap.se = TRUE`, and the number of bootstrap samples is 100 (the default value as suggested in Walter (2019)). Lastly, the household weight (`hhweight`) is assigned to the argument `weights` in order to estimate weighted statistical indicators. It can also be controlled for households of different sizes by assigning `oecd` a variable with household equivalence weights. By applying the `print()` function to the `"kdeAlgo"` object, the estimated statistical indicators (default and custom indicators) as well as their standard errors are printed. For instance, in this example, the estimated mean is about 1,658 Euro and its standard error is 8.486.

```
R> print(Indicators)
Value:
    mean   gini    hcr  quant10   quant25   quant50
1658.329  0.265  0.145  802.227  1117.714  1507.947
 quant75   quant90   pgap     qsr  quant05   quant95
 2020.063 2654.707  0.040   3.920  630.326  3142.296

Standard error:
   mean   gini    hcr quant10 quant25 quant50
  8.486  0.002  0.002   5.839   5.977   6.605
quant75 quant90   pgap    qsr quant05 quant95
 10.548  21.622  0.001  0.044  10.327  24.401
```

For demonstration purposes, the statistical indicators are also estimated using the continuous household income variable from the synthetic EU-SILC data set (Table 3). The estimation results of the KDE algorithm using the interval-censored data are very close to those based on the continuous data. Slightly larger deviations are observable for the more extreme quantiles. This is due to the fact that these quantile estimates fall into intervals with a lower number of observations (compared to the other quantile estimates). Estimation results for these quantiles could potentially be further improved by increasing the number of `evalpoints` of the `kdeAlgo()`.

Table 3: Estimated weighted statistical indicators using the continuous household income variable from the synthetic EU-SILC data set.

mean	gini	hcr	quant10	quant25	quant50
1657.910	0.265	0.144	805.468	1114.028	1508.657
quant75	**quant90**	**pgap**	**qsr**	**quant05**	**quant95**
2017.585	2653.617	0.040	3.960	619.666	3153.425

In Walter (2019), the performance of the KDE algorithm is evaluated via detailed simulation studies. By applying the function `plot()`, `"kdeAlgo"` objects can be plotted. Thereby, convergence plots for all estimated statistical indicators and a plot of the estimated final density are obtained.

```
R> plot(Indicators)
```

Figure 1 shows convergence plots for two of the estimated indicators. Additionally, a plot of the estimated final density with a histogram of the observed data in the background is given in Figure 2. In Figure 1, the estimated statistical indicator (Gini, 10% quantile) for each iteration step of the KDE algorithm and the average over the estimates up to iteration step M (excluding the burn-in iterations) are plotted. A vertical line marks the end of the burn-in period. The horizontal line gives the value of the final estimate (average over the M iterations). All convergence plots indicate that the number of iterations is chosen sufficiently large for the estimates to converge.

If convergence were not achieved, the arguments `burnin` and `samples` should be increased. It is notable that the estimated 10% quantile has the same value for almost all iterations steps. This is the case because the quantile, as any other statistical indicator, is estimated using the pseudo samples that are drawn on 4,000 grid points G. Estimating a quantile on only 4,000 unique outcomes (pseudo values) leads to equal quantile estimates for numerous iteration steps of the KDE algorithm.

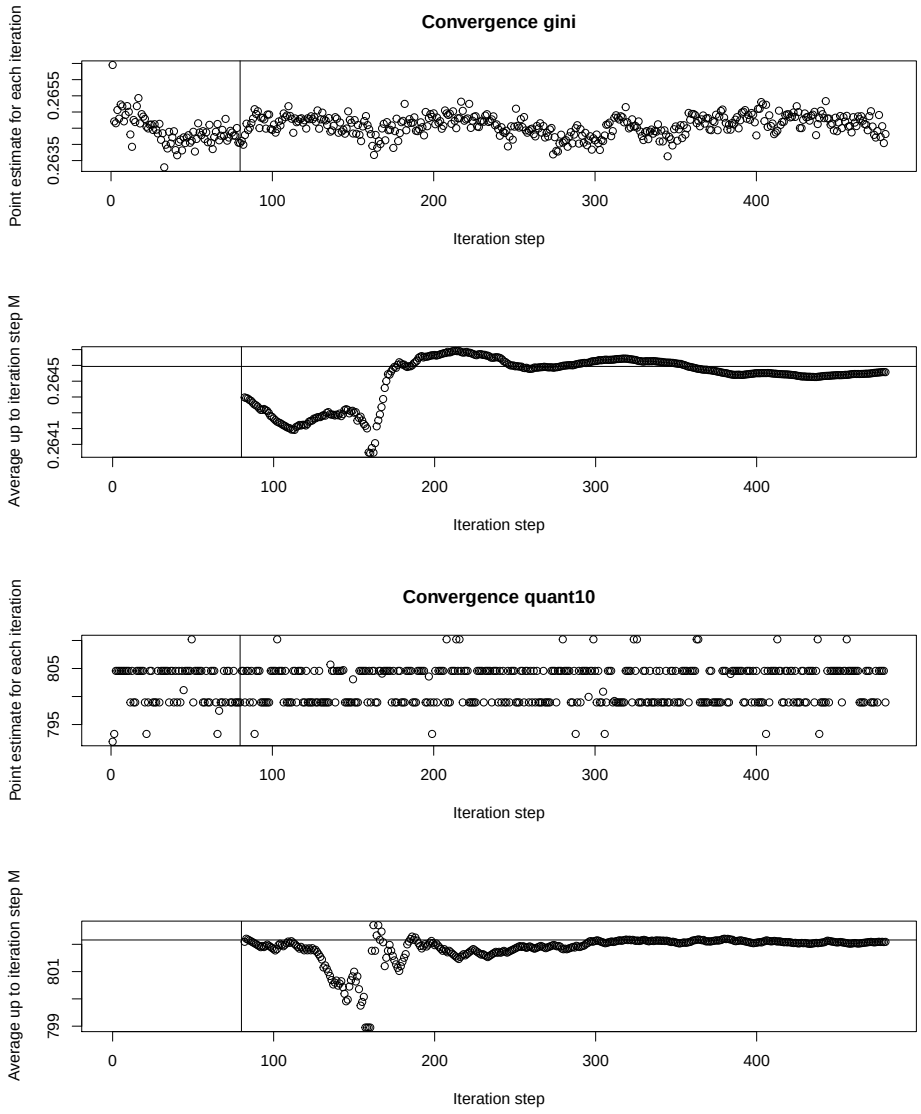


Figure 1: Convergence plots of the estimated indicators (Gini and 10% quantile).

Lastly, it should be mentioned that the computation time can be very long if the estimation of the standard errors is enabled. Hence, if the estimation of the standard errors is not required, the argument `bootstrap.se` should be set to `FALSE`. Furthermore, it should always be checked how many iterations are needed for the desired statistical indicator to converge. Reducing the number of required iterations (arguments `burnin` and `samples`) lowers the computation time significantly (with and without standard errors).

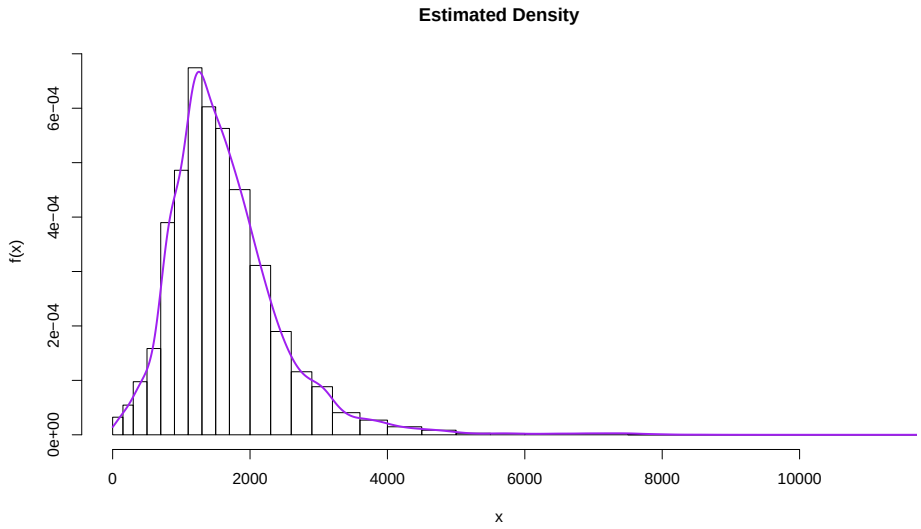


Figure 2: Estimated final density with a histogram of the observed distribution of the data in the background.

Regression analysis

In the following three subsections, the statistical methodology for linear and linear mixed regression models with an interval-censored dependent variable is introduced, the core functionality of the functions `semLM()` and `semLME()` is presented, and examination scores of students from schools in London are exemplary modeled.

Methodology: Regression analysis

The theoretical introduction of the new regression method, proposed by Walter (2019), is presented for linear mixed regression models. The theory for linear regression models can be obtained by simplifying the introduced method. In its standard form, the linear mixed regression model serves to analyze the linear relationship between a continuous dependent variable and some independent variables (Goldstein, 2010). Random parameters (random slopes and random intercepts) are included in the model to account for correlated data, e.g., students within schools. The model in matrix notation (Laird and Ware, 1983) is given by

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \mathbf{Z}\mathbf{v} + \mathbf{e}, \tag{1}$$

where $\mathbf{y}$ is an $n \times 1$ column vector of the dependent variable, n is the sample size, $\mathbf{X}$ is a $n \times p$ matrix where p is equal to the number of predictors, $\boldsymbol{\beta}$ is a column vector of the fixed effects regression parameters of size $p \times 1$, $\mathbf{Z}$ is the $n \times q$ design matrix with q random effects, $\mathbf{v}$ is a $q \times 1$ vector of random effects, and $\mathbf{e}$ is the residual vector of size $n \times 1$. The distribution of the random effects is given by

$$\mathbf{v} \sim N(\mathbf{0}, \mathbf{G}), \qquad \text{where } \mathbf{G} = \begin{bmatrix} \sigma_0^2 & \sigma_{01} & \dots & \sigma_{0q} \\ \sigma_{10} & \sigma_1^2 & \dots & \sigma_{1q} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{q0} & \sigma_{q1} & \dots & \sigma_q^2 \end{bmatrix},$$

and the distribution of the residuals is given by $\mathbf{e} \sim N(\mathbf{0}, \mathbf{R})$ with $\mathbf{R} = \mathbf{I}_n \sigma_e^2$, where $\mathbf{I}_n$ is the identity matrix, and σ_e^2 is the residual variance. The random effects $\mathbf{v}$ and the residuals $\mathbf{e}$ are assumed to be independent. For a more detailed introduction of mixed models, see Searle et al. (1992); McCulloch et al. (2008); Snijders and Bosker (2011). In the case of an interval-censored dependent variable, the parameters of Model (1) have to be estimated without observing $\mathbf{y}$ on a continuous scale. Instead, only the interval identifier $\mathbf{k}$, now defined as $n \times 1$ column vector, is observed. Open-ended interval bounds $A_0 = -\infty$ and $A_{n_k} = +\infty$ and unequal interval widths are allowed. Since the true distribution of $\mathbf{y}$ is unknown, the aim is to reconstruct the distribution of $\mathbf{y}$ using the known intervals $\mathbf{k}$ and the linear relationship stated in Model (1). As presented in Walter (2019), in order to reconstruct the unknown distribution of $f(\mathbf{y}|\mathbf{X}, \mathbf{Z}, \mathbf{v}, \mathbf{k}, \boldsymbol{\theta})$, where $\boldsymbol{\theta} = (\boldsymbol{\beta}, \mathbf{R}, \mathbf{G})$, the Bayes theorem (Bayes, 1763) is

applied. Hence,

$$f\left(\mathbf{y}|\mathbf{X},\mathbf{Z},\mathbf{v},\mathbf{k},\boldsymbol{\theta}\right) \propto f\left(\mathbf{k}|\mathbf{y},\mathbf{X},\mathbf{Z},\mathbf{v},\boldsymbol{\theta}\right) f\left(\mathbf{y}|\mathbf{X},\mathbf{Z},\mathbf{v},\boldsymbol{\theta}\right),$$

with $f\left(\mathbf{k}|\mathbf{y},\mathbf{X},\mathbf{Z},\mathbf{v},\boldsymbol{\theta}\right) = f\left(\mathbf{k}|\mathbf{y}\right)$ because the conditional distribution of the interval identifier $\mathbf{k}$ only depends on $\mathbf{y}$. It is given by $f\left(\mathbf{k}|\mathbf{y}\right) = \mathbf{r}$, with $\mathbf{r}$ being an $n \times 1$ column vector $\mathbf{r} = \left(r_1, r_2, \ldots, r_n\right)^T$, with

$$r_i = \begin{cases} 1 & \text{if } A_{k_i-1} \leq y_i \leq A_{k_i}, \\ 0 & \text{else,} \end{cases}$$

for $i = 1, \ldots, n$ and

$$f\left(\mathbf{y}|\mathbf{X},\mathbf{Z},\mathbf{v},\boldsymbol{\theta}\right) \sim \quad N\left(\mathbf{X}\boldsymbol{\beta} + \mathbf{Z}\mathbf{v}, \mathbf{R}\right). \tag{2}$$

The relationship in Equation (2) follows from the linear mixed model assumptions (Model (1)). The unknown parameters $\boldsymbol{\theta} = \left(\boldsymbol{\beta}, \mathbf{R}, \mathbf{G}\right)$ are estimated based on pseudo samples $\tilde{\mathbf{y}}$ (since $\mathbf{y}$ is unknown) that are iteratively drawn from $f\left(\mathbf{y}|\mathbf{X},\mathbf{Z},\mathbf{v},\mathbf{k},\boldsymbol{\theta}\right)$. The next subsection states the computational details of the SEM algorithm.

Estimation and computational details

To fit Model (1), the parameter vector $\hat{\boldsymbol{\theta}} = \left(\widehat{\boldsymbol{\beta}}, \widehat{\mathbf{R}}, \widehat{\mathbf{G}}\right)$ is estimated, and pseudo samples of the unknown $\mathbf{y}$ are iteratively generated by the following SEM algorithm. The pseudo samples $\tilde{\mathbf{y}}$ are drawn from the conditional distribution

$$f\left(\mathbf{y}|\mathbf{X},\mathbf{Z},\mathbf{v},\mathbf{k},\boldsymbol{\theta}\right) \propto \mathbf{I}\left(A_{\mathbf{k}-1} \leq \mathbf{y} \leq A_{\mathbf{k}}\right) \times N\left(\mathbf{X}\boldsymbol{\beta} + \mathbf{Z}\mathbf{v}, \mathbf{R}\right),$$

where $\mathbf{I}\left(\cdot\right)$ denotes the indicator function. Hence, for $\mathbf{y}$ with explanatory variables $\mathbf{X}$, the corresponding $\tilde{\mathbf{y}}$ is drawn from $N\left(\mathbf{X}\boldsymbol{\beta} + \mathbf{Z}\mathbf{v}, \mathbf{R}\right)$ conditional on the given interval $\left(A_{\mathbf{k}-1} \leq \mathbf{y} \leq A_{\mathbf{k}}\right)$. If $\hat{\boldsymbol{\theta}}$ is estimated, the conditional distribution $f\left(\mathbf{y}|\mathbf{X},\mathbf{Z},\mathbf{v},\mathbf{k},\boldsymbol{\theta}\right)$ follows a two-sided truncated normal distribution. Its probability density function equals

$$\hat{f}\left(\mathbf{y}|\mathbf{X},\mathbf{Z},\hat{\mathbf{v}},\mathbf{k},\hat{\boldsymbol{\theta}}\right) = \frac{\phi\left(\frac{\mathbf{y}-\hat{\boldsymbol{\mu}}}{\hat{\sigma}_e}\right)}{\hat{\sigma}_e\left(\Phi\left(\frac{A_{\mathbf{k}}-\hat{\boldsymbol{\mu}}}{\hat{\sigma}_e}\right) - \Phi\left(\frac{A_{\mathbf{k}-1}-\hat{\boldsymbol{\mu}}}{\hat{\sigma}_e}\right)\right)}, \tag{3}$$

with $\hat{\boldsymbol{\mu}} = \mathbf{X}\widehat{\boldsymbol{\beta}} + \mathbf{Z}\hat{\mathbf{v}}$. $\phi\left(\cdot\right)$ denotes the probability density function of the standard normal distribution, and $\Phi(\cdot)$ denotes its cumulative distribution function. From its definition, it follows that $\Phi\left(\frac{A_{\mathbf{k}}-\hat{\boldsymbol{\mu}}}{\hat{\sigma}_e}\right) = 1$ if $A_{\mathbf{k}} = +\infty$, and $\Phi\left(\frac{A_{\mathbf{k}-1}-\hat{\boldsymbol{\mu}}}{\hat{\sigma}_e}\right) = 0$ if $A_{\mathbf{k}-1} = -\infty$. The steps of the SEM algorithm as described in Walter (2019) are:

1. Estimate $\hat{\boldsymbol{\theta}} = \left(\widehat{\boldsymbol{\beta}}, \widehat{\mathbf{R}}, \widehat{\mathbf{G}}\right)$ from Model (1) using the midpoints of the intervals as substitutes for the unknown $\mathbf{y}$. The parameters are estimated by restricted maximum likelihood theory (REML) (Thompson, 1962).
2. **Stochastic step:** For $i = 1, \ldots, n$, draw randomly from $N\left(\mathbf{X}\widehat{\boldsymbol{\beta}} + \mathbf{Z}\hat{\mathbf{v}}, \widehat{\mathbf{R}}\right)$ within the given interval $\left(A_{\mathbf{k}-1} \leq \mathbf{y} \leq A_{\mathbf{k}}\right)$ (the two-sided truncated normal distribution given in Equation (3)) obtaining $\left(\tilde{\mathbf{y}}, \mathbf{X}, \mathbf{Z}\right)$. The drawn pseudo $\tilde{\mathbf{y}}$ are used as replacements for the unobserved $\mathbf{y}$.
3. **Maximization step:** Re-estimate the parameter vector $\hat{\boldsymbol{\theta}}$ from Model (1) by using the pseudo samples $\left(\tilde{\mathbf{y}}, \mathbf{X}, \mathbf{Z}\right)$ from Step 2. Again, parameter estimation is carried out by REML.
4. Iterate Steps 2-3 $B^{(SEM)} + M^{(SEM)}$ times, with $B^{(SEM)}$ burn-in iterations and $M^{(SEM)}$ additional iterations.
5. Discard the burn-in iterations $B^{(SEM)}$ and estimate $\hat{\boldsymbol{\theta}}$ by averaging the obtained $M^{(SEM)}$ estimates.

If open-ended intervals $A_0 = -\infty$ and $A_{n_k} = +\infty$ are present, the midpoints M_1 and M_{n_k} of these intervals in iteration Step 1 are computed as follows:

$$M_1 = \left(A_1 - \overline{D}\right)/2,$$
$$M_{n_k} = \left(A_{n_k-1} + \overline{D}\right)/2,$$

where

$$\overline{D} = \frac{1}{(n_k - 2)} \sum_{k=2}^{n_k-1} |A_{k-1} - A_k|.$$

These midpoints serve as proxies for the unknown interval midpoints in Step 1 of the algorithm. The SEM algorithm for the linear regression model is obtained by simplifying the conditional distribution $f(\mathbf{y}|\mathbf{X},\mathbf{Z},\mathbf{v},\boldsymbol{\theta}) \sim N(\mathbf{X}\boldsymbol{\beta}+\mathbf{Z}\mathbf{v},\mathbf{R})$ to $f(\mathbf{y}|\mathbf{X},\boldsymbol{\beta},\sigma_e^2) \sim N(\mathbf{X}\boldsymbol{\beta},\sigma_e^2)$ according to the model assumptions of a linear regression model. In the SEM algorithm for linear models, it is then drawn from $N(\mathbf{X}\boldsymbol{\beta},\sigma_e^2)$ within the given interval.

The standard errors of the regression parameters are estimated using bootstrap methods. For the linear regression model, a non parametric bootstrap (Efron and Stein, 1981; Efron, 1982; Efron and Tibshirani, 1986, 1993) and for the linear mixed regression model, a parametric bootstrap (Wang et al., 2006; Thai et al., 2013) is used to estimate the standard errors. The non parametric, as well as the parametric bootstrap, are further developed to account for the additional uncertainty that is due to the interval-censored dependent variable. Both newly proposed bootstraps are available in the **smicd** package, and the theory is explained in (Walter, 2019).

To assure that the model assumptions are fulfilled, the logarithmic and the Box-Cox transformations are incorporated into the function `semLm()` and `semLme()`.

Core functionality: Regression analysis

The introduced SEM algorithm is implemented in the functions described in Table 4. The arguments and default settings of the estimation functions `semLm()` and `semLme()` are summarized in Table 5. Both functions return an S3 object of class `"sem"`, `"lm"` or `"sem"`, `"lme"`. A detailed explanation of all the components of these objects can be found in the **smicd** package documentation. The generic functions `plot()`, `print()`, and `summary()` can be applied to objects of class `"sem"`,`"lm"` and `"sem"`,`"lme"` in order to summarize the main estimation results. In the next section, the functionality of `semLm()` and `semLme()` is demonstrated based on an illustrative example.

Table 4: Implemented functions for the estimation of linear and linear mixed regression models.

Function Name	Description
`semLm()`	Estimates linear regression models with an interval-censored dependent variable
`semLme()`	Estimates linear mixed regression models with an interval-censored dependent variable
`plot()`	Plots convergence of the estimated parameters and estimated density of the pseudo $\tilde{\mathbf{y}}$ from the last iteration step
`print()`	Prints basic information of the estimated linear and linear mixed regression models
`summary()`	Summary of the estimated linear and linear mixed regression models

Table 5: Arguments of functions `semLm()` and `semLme()`.

Argument	Description	Default
`formula`	A two-sided linear formula object	
`data`	A data frame containing the variables of the model	
`classes`	Numeric vector of interval bounds	
`burnin`	Burn-in iterations	`40`
`samples`	Additional iterations	`200`
`trafo`	Transformation of the dependent variable: None, logarithmic, or Box-Cox transformation	`"None"`
`adjust`	Extends the number of iterations for the estimation of the Box-Cox transformation parameter: $(burnin + samples) * adjust$	`2`
`bootstrap.se`	If TRUE, standard errors and confidence intervals of the regression parameters are estimated	`FALSE`
`b`	Number of bootstraps for the estimation of the standard errors	`100`

Example: Regression analysis

To demonstrate the functions `semLm()` and `semLme()`, the famous London school data set that is analyzed in Goldstein et al. (1993) is used. The data set contains the examination results of 4,059 students from 65 schools in six Inner London Education Authorities. The data set is available in the R package **mlmRev** (Bates et al., 2020a) and also included in the package **smicd**. The variables used in the following example are: general certificate of secondary examination scores (`examsc`), the standardized London reading test scores at the age of 11 years (`standLRT`), the sex of the student (`sex`), and the school identifier (`school`). In the original data set, the variable `examsc` is measured on a continuous scale. In order to demonstrate the functionality of the functions `semLm()` and `semLme()`, the variable is arbitrarily censored to nine intervals. As before, the censoring is carried out by the function `cut()`, and the vector of interval bounds is called `intervals`.

```
R> intervals <- c(1, 1.5, 2.5, 3.5, 4.5, 5.5, 6.5, 7.7, 8.5, Inf)
R> Exam$examsc.class <- cut(Exam$examsc, intervals)
```

The newly created interval-censored variable is called `examsc.class`. The distribution is visualized by applying the function `table()`.

```
R> table(Exam$examsc.class)
  (1,1.5] (1.5,2.5] (2.5,3.5] (3.5,4.5] (4.5,5.5] (5.5,6.5]
        1        32       249       937      1606       951
(6.5,7.7] (7.7,8.5] (8.5,Inf]
      267        15         1
```

It can be seen that most examination scores are concentrated in the center intervals. To fit the linear regression model, the function `semLM()` is called.

```
R> LM <- semLm(
+    formula = examsc.class ~ standLRT + sex, data = Exam,
+    classes = intervals,  bootstrap.se = TRUE
+  )
```

The formula argument is assigned the model equation, where `examsc.class` is regressed on `standLRT` and `sex`. The argument `data` is assigned the name of the data set `Exam`, and the vector of interval bounds `intervals` is assigned to the `classes` argument. The arguments `burnin` and `samples` are left as defaults. The specified number of default iterations is sufficiently large for most regression models. However the convergence of the parameters has to be checked by plotting the estimation results with the function `plot()` after the estimation. No transformation is specified for the interval-censored dependent variable and therefore, `trafo` is assigned its default value. The argument `adjust` is only relevant if the Box-Cox transformation `trafo="bc"` is chosen. In this case, the number of iterations for the estimation of the Box-Cox transformation parameter λ can be specified by this argument. The convergence of the transformation parameter λ has to be checked using the function `plot()`. More information on the Box-Cox transformation and on the estimation of the transformation parameter is given in Walter (2019). For the estimation of the standard errors of the regression parameters, the argument `bootstrap.se` is set to `TRUE`. The number of bootstrap samples `b` is 100, its default value, which again is reasonable for most settings. A summary of the estimation results is obtained by the application of the function `summary()`.

```
R> summary(LM)
Call:
semLm(formula = examsc.class ~ standLRT + sex, data = Exam,
     classes = intervals, bootstrap.se = TRUE)

Fixed effects:
             Estimate  Std.Error Lower 95%-level Upper 95%-level
(Intercept) 5.0702791 0.01816102       5.0326739       5.1033905
standLRT    0.5908015 0.01349275       0.5614197       0.6163845
sexM       -0.1715966 0.03093346      -0.2308930      -0.1010877

Multiple R-squared: 0.3501  Adjusted R-squared: 0.3498
Variable examsc.class is divided into 9 intervals.
```

The output shows the function call, the estimated regression coefficients, the bootstrapped standard errors, and the confidence intervals, as well as the R-squared and the adjusted R-squared. Furthermore, the output reminds the user that the dependent variable is censored to nine intervals. All estimates are

interpreted as in a linear regression model with a continuous dependent variable. Hence, if `standLRT` increases by one unit and all other parameters are kept constant, `examsc.class` increases by 0.59 on average. The bootstrapped confidence intervals indicate that all regressors have a significant effect on the dependent variable.

By using the generic function `plot()` on an object of class `"sem"` and `"lm"`, convergence plots of each estimated regression parameter and of the estimated residual variance are obtained. Furthermore, the density of the generated pseudo $\tilde{\mathbf{y}}$ variable from the last iteration step is plotted with a histogram of the observed distribution of the interval-censored variable `examsc.class` in the background.

```
R> plot(LM)
```

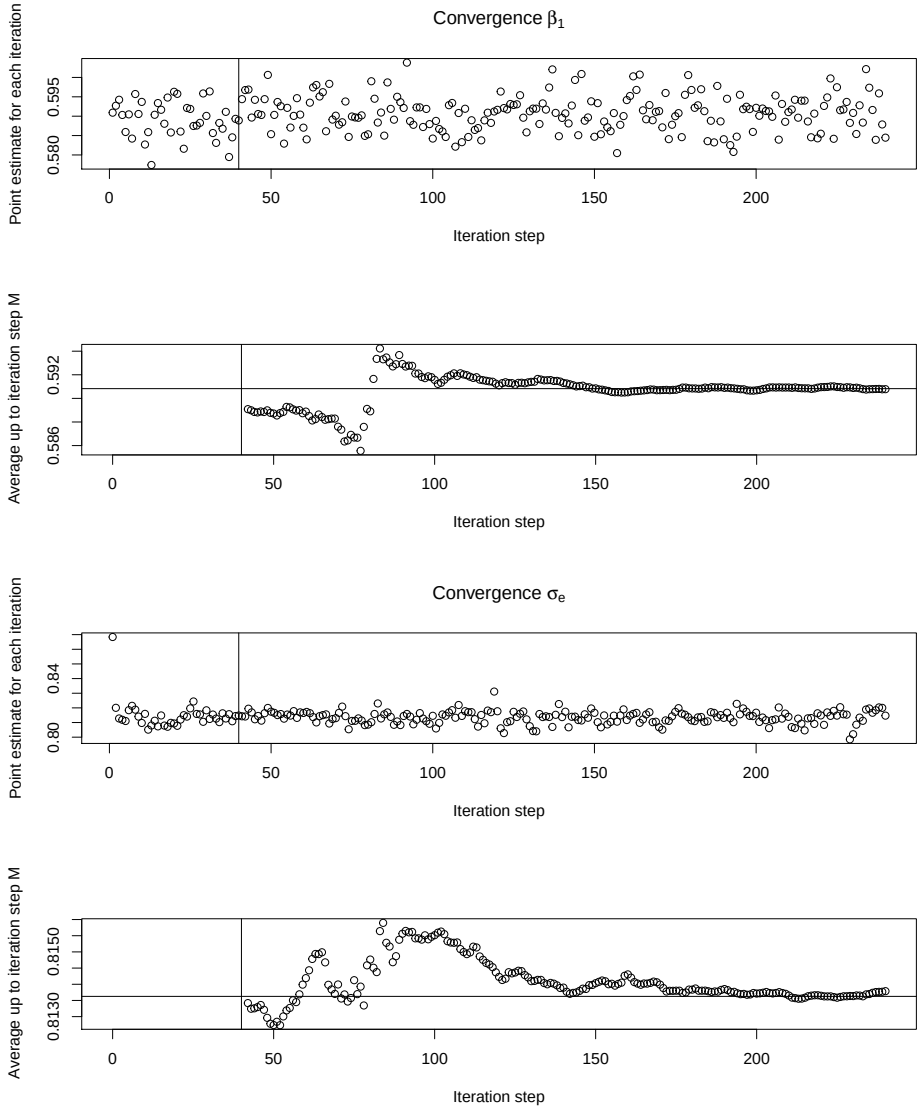


Figure 3: Convergence plots of estimated model parameters (β_1 and σ_e).

In Figure 3, a selection of convergence plots is given, and in Figure 4, the density of the pseudo $\tilde{\mathbf{y}}$ from the last iteration step of the SEM algorithm is plotted. In the convergence plots, the estimated parameter and the average up to iteration step M (excluding B) are plotted for each iteration step of the SEM algorithm. A vertical line indicates the end of the burn-in period (40 iterations). The final parameter estimate marked by the horizontal line is obtained by averaging the $M^{(SEM)}$ additional iterations (200). The selected 240 iterations are enough to obtain reliable estimates in this example because the estimates have converged.

As already mentioned, the **smicd** package also enables the estimation of linear mixed regression models by the function `semLme()`. In the London school data set, students are nested within schools, and therefore, it is necessary to control for the correlation within-schools. In order to do that, the

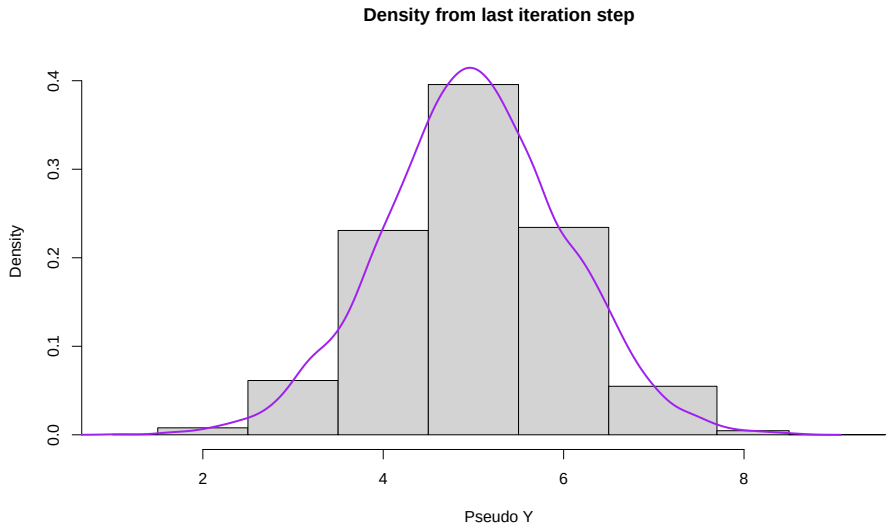


Figure 4: Estimated final density with a histogram of the observed distribution of the data in the background.

variable `school` is specified as a random intercept. If necessary, a random slope parameter could also be included in the model. Again, the variable `sex` is included as an additional regressor. Hence, the `formula` argument is assigned the following model equation `examsc.class` $\sim$ `standLRT + sex + (1|school)`. So far, the function `semLme()` enables the estimation of linear mixed models with a maximum of one random slope and one random intercept parameter. Regarding all other arguments, the same specifications as before are made.

```
R> LME <- semLme(
+    formula = examsc.class ~ standLRT + sex + (1|school),
+    data = Exam, classes = intervals, bootstrap.se = TRUE
+  )
```

By using the generic function `summary()`, the estimation results are printed. In addition to the fixed effects, the estimated random effects are obtained as in the **lme4** and **nlme** packages. Since the R-squared and the adjusted R-squared are not defined for mixed models, the `summary()` function prints the marginal R-squared and conditional R-squared (Nakagawa and Schielzeth, 2013; Johnson, 2014).

```
> summary(LME)
Call:
semLme(formula = examsc.class ~ standLRT + sex + (1 | school),
    data = Exam, classes = intervals, bootstrap.se = TRUE)

Random effects:
    Groups        Name   Variance  Std.Dev.
    school (Intercept) 0.08755431 0.2958958
  Residual             0.58417586 0.7643140

Fixed effects:
             Estimate    Std.Error Lower 95%-level Upper 95%-level
(Intercept) 5.0777581 0.0005188930       5.0769749       5.0791095
standLRT    0.5605049 0.0003665976       0.5599456       0.5613711
sexM       -0.1711065 0.0008159909      -0.1724193      -0.1692369

Marginal R-squared: 0.324 Conditional R-squared: 0.4121
Variable examsc.class is divided into 9 intervals.
```

Again, interpretation is the same as in linear mixed models with a continuous dependent variable. By applying the generic function `plot()` to a `"sem" "lme"` object, the same plots as for the linear regression model are plotted.

Discussion and outlook

Asking for interval-censored data can lead to lower item non-response rates and increased data quality. While item non-response is potentially avoided, applying traditional statistical methods becomes infeasible because the true distribution of the data within each interval is unknown. The functions of the **smicd** package enable researchers to easily analyze this kind of data. The paper briefly introduces the new statistical methodology and presents, in detail, the core functions of the package:

- `kdeAlgo()` for the direct estimation of any statistical indicator,
- `semLm()` to estimate linear models with an interval-censored dependent variable,
- `semLme()` to estimate linear mixed models with an interval-censored dependent variable.

The functions are applied in order to estimate statistical indicators from interval-censored synthetic EU-SILC income data and to analyze interval-censored examination scores of students from London with linear and linear mixed regression models.

Further developments of the **smicd** package will include the possibility to estimate the bootstrapped standard errors in parallel computing environments. Additionally, it is planned to allow for the use of survey weights in the linear (mixed) regression models.

Bibliography

A. Agresti. *Analysis of Ordinal Categorical Data*. Wiley, New Jersey, 2010. URL https://doi.org/10.1002/9780470594001. [p397]

A. Alfons and M. Templ. Estimation of social exclusion indicators from complex surveys: The R package laeken. *Journal of Statistical Software*, 54(15):1–25, 2013. URL http://www.jstatsoft.org/v54/i15/. [p399]

A. Alfons, J. Holzer, and M. Templ. *laeken: Estimation of Indicators on Social Exclusion and Poverty*, 2020. URL https://CRAN.R-project.org/package=laeken. R package version 0.5.1. [p399]

Australian Bureau of Statistics. Census household form, 2011. URL https://unstats.un.org/unsd/demographic/sources/census/quest/AUS2011en.pdf. Accessed: 2020-07-18. [p396]

R. Bandourian, J. McDonald, and R. S. Turley. A comparison of parametric models of income distribution across countries and over time. Technical report, Luxembourg Income Study, 2002. [p396]

D. Bates, M. Mächler, B. Bolker, and S. Walker. Fitting linear mixed-effects models using lme4. *Journal of Statistical Software*, 67(1):1–48, 2015. URL https://doi.org/10.18637/jss.v067.i01. [p397]

D. Bates, M. Maechler, and B. Bolker. *mlmRev: Examples from Multilevel Modelling Software Review*, 2020a. URL https://CRAN.R-project.org/package=mlmRev. R package version 1.0-8. [p406]

D. Bates, M. Maechler, B. Bolker, and S. Walker. *lme4: Linear Mixed-Effects Models using 'Eigen' and S4*, 2020b. URL https://CRAN.R-project.org/package=lme4. R package version 1.1-23. [p397]

T. Bayes. An essay towards solving a problem in the doctrine of chances. By the late Rev. Mr. Bayes, F. R. S. communicated by Mr. Price, in a letter to John Canton, A. M. F. R. S. *Philosophical Transactions*, 53:370–418, 1763. URL https://doi.org/10.1098/rstl.1763.0053. [p398, 403]

Y. G. Berger and E. L. Escobar. Variance estimation of imputed estimators of change for repeated rotating surveys. *International Statistical Review*, 85(3):421–438, 2016. URL https://doi.org/10.1111/insr.12197. [p396]

G. E. P. Box and D. R. Cox. An analysis of transformations. *Journal of the Royal Statistical Society: Series B*, 26(2):211–252, 1964. URL http://www.jstor.org/stable/2984418. [p397]

T. A. Cameron. The impact of grouping coarseness in alternative grouped-data regression models. *Journal of Econometrics*, 35(1):37–57, 1987. URL https://doi.org/10.1016/0304-4076(87)90080-7. [p397]

R. H. B. Christensen. *ordinal: Regression Models for Ordinal Data*, 2019. URL https://CRAN.R-project.org/package=ordinal. R package version 2019.12-10. [p397]

C. Dagum. *A New Model of Personal Income Distribution: Specification and Estimation*, pages 3–25. Springer New York, New York, NY, 2008. URL https://doi.org/10.1007/978-0-387-72796-7_1. [p396]

M. L. Delignette-Muller and C. Dutang. fitdistrplus: An R package for fitting distributions. *Journal of Statistical Software*, 64(4):1–34, 2015. URL http://www.jstatsoft.org/v64/i04/. [p396]

M.-L. Delignette-Muller, C. Dutang, and A. Siberchicot. *fitdistrplus: Help to Fit of a Parametric Distribution to Non-Censored or Censored Data*, 2020. URL https://CRAN.R-project.org/package=fitdistrplus. R package version 1.1-1. [p396]

Departamento Administrativo Nacional De Estadística. Censo general 2005, 2005. URL https://www.dane.gov.co/files/censos/libroCenso2005nacional.pdf?&. Accessed: 2020-07-18. [p396]

C. Dutang, V. Goulet, and M. Pigeon. actuar: An r package for actuarial science. *Journal of Statistical Software*, 25(7):38, 2008. URL http://www.jstatsoft.org/v25/i07. [p396]

B. Efron. Bootstrap methods: another look at the jackknife. *The Annals of Statistics*, 7(1):1–26, 1979. URL https://www.jstor.org/stable/2958830. [p399]

B. Efron. *The Jackknife, the Bootstrap and Other Resampling Plans*. Stanford University, Philadelphia, 1982. URL https://doi.org/10.1137/1.9781611970319. [p405]

B. Efron and C. Stein. The jackknife estimate of variance. *The Annals of Statistics*, 9(3):586–596, 1981. URL https://doi.org/10.1214/aos/1176345462. [p405]

B. Efron and R. Tibshirani. Bootstrap methods for standard errors, confidence intervals, and other measures of statistical accuracy. *Statistical Science*, 1(1):54–75, 1986. URL https://doi.org/10.1214/ss/1177013815. [p405]

B. Efron and R. Tibshirani. *An Introduction to the Bootstrap*. Chapman & Hall, New York, 1993. [p405]

Eurostat. Statistics explained: at-risk-of-poverty rate, 2014. URL http://ec.europa.eu/eurostat/statistics-explained/index.php/Glossary:At-risk-of-poverty_rate. Accessed: 2020-07-11. [p401]

L. Fahrmeir, R. Künstler, I. Pigeot, and G. Tutz. *Statistik - Der Weg zur Datenanalyse*. Springer, Berlin, 2016. URL https://doi.org/10.1007/978-3-662-50372-0. [p396]

J. Foster, J. Greer, and E. Thorbecke. A class of decomposable poverty measures. *Econometrica*, 52(3): 761–766, 1984. URL https://doi.org/10.2307/1913475. [p401]

J. G. Fryer and R. J. Pethybridge. Maximum likelihood estimation of a linear regression function with grouped data. *Journal of the Royal Statistical Society: Series C*, 21(2):142–154, 1972. URL https://doi.org/10.2307/2346486. [p397]

C. Gini. *Variabilità e mutabilità: contributo allo studio delle distribuzioni e delle relazioni statistiche*. Studi economico-giuridici pubblicati per cura della facoltà di Giurisprudenza della R. Università di Cagliari. Tipogr. di P. Cuppini, Bologna, 1912. URL https://books.google.de/books?id=fqjaBPMxB9kC. [p401]

H. Goldstein. *Multilevel Statistical Models*. Wiley, New York, 2010. URL https://doi.org/10.1002/9780470973394. [p403]

H. Goldstein, J. Rasbash, M. Yang, G. Woodhouse, H. Pan, D. Nuttall, and S. Thomas. A multilevel analysis of school examination results. *Oxford Review of Education*, 19(4):425–433, 1993. URL https://doi.org/10.1080/0305498930190401. [p406]

V. Goulet, C. Dutang, M. Pigeon, J. A. Ryan, R. Gentleman, R. Ihaka, R Core Team, and R Foundation. *actuar: Actuarial Functions and Heavy Tailed Distributions*, 2020. URL https://CRAN.R-project.org/package=actuar. R package version 3.0-0. [p396]

M. Groß, U. Rendtel, T. Schmid, S. Schmon, and N. Tzavidis. Estimating the density of ethnic minorities and aged people in Berlin: multivariate kernel density estimation applied to sensitive georeferenced administrative data protected via measurement error. *Journal of the Royal Statistical Society: Series A*, 180(1):161–183, 2017. URL https://doi.org/10.1111/rssa.12179. [p398, 401]

M. J. Gurka, L. J. Edwards, K. E. Muller, and L. Kupper. Extending the Box-Cox transformation to the linear mixed model. *Journal of the Royal Statistical Society: Series A*, 169(2):273–288, 2006. URL https://doi.org/10.1111/j.1467-985X.2005.00391.x. [p397]

A. Hagenaars and K. D. Vos. The definition and measurement of poverty. *Journal of Human Resources*, 23(2):211–221, 1988. URL https://doi.org/10.2307/145776. [p396]

P. Johnson. Extension of Nakagawa & Schielzeth's R^2_{GLMM} to random slopes models. *Methods in Ecology and Evolution*, 5(9):944–946, 2014. URL https://doi.org/10.1111/2041-210X.12225. [p408]

M. C. Jones, J. S. Marron, and S. J. Sheather. A brief survey of bandwidth selection for density estimation. *Journal of the American Statistical Association*, 91(433):401–407, 1996. URL https://doi.org/10.2307/2291420. [p398]

M. N. Laird and J. H. Ware. Random-effects models for longitudinal data. *Biometrics*, 38(4):963–74, 1983. URL https://doi.org/10.2307/2529876. [p403]

S. Lenau and R. Münnich. Estimating income poverty and inequality from income classes. Technical report, InGRID Integrating Expertise in Inclusive Growth: Case Studies, 2016. [p396, 397]

C. R. Loader. Bandwidth selection: classical or plug-in? *Annals of Statistics*, 27(2):415–438, 1999. URL https://doi.org/10.1214/aos/1018031201. [p398]

P. McCullagh. Regression models for ordinal data. *Journal of the Royal Statistical Society: Series B*, 42(2): 109–142, 1980. URL http://www.jstor.org/stable/2984952. [p397]

C. E. McCulloch, S. R. Searle, and J. M. Neuhaus. *Generalized, Linear, and Mixed Models*. Wiley, New Jersey, 2008. [p403]

J. B. McDonald. Some generalized functions for the size distribution of income. *Econometrica*, 52(3): 647–663, 1984. URL https://doi.org/10.2307/1913469. [p396]

J. C. Moore and E. J. Welniak. Income measurement error in surveys: a review. *Journal of Official Statistics*, 16(4):331–361, 2000. [p396]

S. Nakagawa and H. Schielzeth. A general and simple method for obtaining R^2 from generalized linear mixed-effects models. *Methods in Ecology and Evolution*, 4(2):133–142, 2013. URL https://doi.org/10.1111/j.2041-210x.2012.00261.x. [p408]

E. Parzen. On estimation of a probability density function and mode. *The Annals of Mathematical Statistics*, 33(3):1065–1076, 1962. URL https://doi.org/10.1214/aoms/1177704472. [p398]

J. Pinheiro, D. Bates, and R-core. *nlme: Linear and Nonlinear Mixed Effects Models*, 2020. URL https://CRAN.R-project.org/package=nlme. R package version 3.1-148. [p397]

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2020. URL https://www.R-project.org/. [p397]

B. Ripley. *MASS: Support Functions and Datasets for Venables and Ripley's MASS*, 2019. URL https://CRAN.R-project.org/package=MASS. R package version 7.3-51.5. [p397]

M. Rosenblatt. Remarks on some nonparametric estimates of a density function. *The Annals of Mathematical Statistics*, 27(3):832–837, 1956. URL https://www.jstor.org/stable/2237390. [p398]

R. N. Rosett and F. D. Nelson. Estimation of the two-limit probit regression model. *Econometrica*, 43(1): 141–146, 1975. URL https://doi.org/10.2307/1913419. [p397]

S. R. Searle, G. Casella, and C. E. McCulloch. *Variance Components*. Wiley, New York, 1992. [p403]

J. Shao and D. Tu. *The Jackknife and Bootstrap*. Springer, New York, 1995. URL https://doi.org/10.1007/978-1-4612-0795-5. [p399]

T. Snijders and R. Bosker. *Multilevel Analysis: An Introduction to Basic and Advanced Multilevel Modeling*. Sage, London, 2011. [p403]

Statistisches Bundesamt. Codebook microsensus 2014, 2014. URL https://www.forschungsdatenzentrum.de/sites/default/files/mz_2014_suf_dhb.pdf. Accessed: 2020-07-18. [p399]

Statistisches Bundesamt. Data supply: microcensus, 2016. URL https://www.forschungsdatenzentrum.de/de/haushalte/mikrozensus. Accessed: 2020-07-18. [p399]

Statistisches Bundesamt. Datenhandbuch zum Mikrozensus Scientific-Use-File 2012. http://www.forschungsdatenzentrum.de/bestand/mikrozensus/suf/2012/fdz_mz_suf_2012_schluesselverzeichnis.pdf, 2017. Accessed: 2020-07-18. [p396]

M. Stewart. On least square estimation when the dependent varaible is grouped. *The Review of Economic Studies*, 50(4):737–753, 1983. URL https://doi.org/10.2307/2297773. [p397]

H. Thai, F. Mentre, N. Holford, C. Veyrat-Follet, and E. Comets. A comparison of bootstrap approaches for estimating uncertainty of parameters in linear mixed-effects models. *Pharmaceutical Statistics*, 12 (3):129–140, 2013. URL https://doi.org/10.1002/pst.1561. [p405]

T. M. Therneau. *survival: Survival Analysis*, 2020. URL https://CRAN.R-project.org/package=survival. R package version 3.2-3. [p397]

T. M. Therneau and P. M. Grambsch. *Modeling Survival Data: Extending the Cox Model*. Springer, New York, 2000. [p397]

J. W. A. Thompson. The problem of negative estimates of variance components. *Annals of Mathematical Statistics*, 33(1):273–289, 1962. URL https://doi.org/10.1214/aoms/1177704731. [p404]

M. L. Thompson and K. Nelson. Linear regression with type I interval- and left-censored response data. *Environmental and Ecological Statistics*, 10(2):221–230, 2003. URL https://doi.org/10.1023/A:1023630425376. [p397]

J. Tobin. Estimation of relationships for limited dependent variables. *Econometrica*, 26(1):24–36, 1958. URL https://doi.org/10.2307/1907382. [p397]

W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S*. Springer, New York, 2002. URL https://doi.org/10.1007/978-0-387-21706-2. [p397]

P. Walter. *A Selection of Statistical Methods for Interval-Censored Data with Applications to the German Microcensus*. PhD thesis, Freie Universität Berlin, 2019. URL https://doi.org/10.17169/refubium-1621. [p397, 398, 399, 401, 403, 404, 405, 406]

P. Walter. *smicd: Statistical Methods for Interval Censored Data*, 2020. URL https://CRAN.R-project.org/package=smicd. R package version 1.1.1. [p397]

B. Wang and M. Wertelecki. Density estimation for data with rounding errors. *Computational Statistics & Data Analysis*, 65:4–12, 2013. URL https://doi.org/10.1016/j.csda.2012.02.016. [p398]

J. Wang, J. R. Carpenter, and M. A. Kepler. Using SAS to conduct nonparametric residual bootstrap multilevel modeling with a small number of groups. *Computer Methods and Programs in Biomedicine*, 82(2):130–143, 2006. URL https://doi.org/10.1016/j.cmpb.2006.02.006. [p405]

A. Z. Zambom and R. Dias. A review of kernel density estimation with applications to econometrics. *International Econometric Review*, 5(1):20–42, 2012. URL https://EconPapers.repec.org/RePEc:erh:journl:v:5:y:2013:i:1:p:20-42. [p398]

Paul Walter
Freie Universität Berlin
Garystraße 21, 14195 Berlin
Germany
paul.walter@fu-berlin.de

krippendorffsalpha: An R Package for Measuring Agreement Using Krippendorff's Alpha Coefficient

by John Hughes

Abstract R package **krippendorffsalpha** provides tools for measuring agreement using Krippendorff's α coefficient, a well-known nonparametric measure of agreement (also called inter-rater reliability and various other names). This article first develops Krippendorff's α in a natural way and situates α among statistical procedures. Then, the usage of package **krippendorffsalpha** is illustrated via analyses of two datasets, the latter of which was collected during an imaging study of hip cartilage. The package permits users to apply the α methodology using built-in distance functions for the nominal, ordinal, interval, or ratio levels of measurement. User-defined distance functions are also supported. The fitting function can accommodate any number of units, any number of coders, and missingness. Bootstrap inference is supported, and the bootstrap computation can be carried out in parallel.

Introduction

Krippendorff's α (Hayes and Krippendorff, 2007) is a well-known nonparametric measure of agreement (i.e., consistency of scoring among two or more raters for the same units of analysis (Gwet, 2014)). In R (Ihaka and Gentleman, 1996), Krippendorff's α can be applied using function `kripp.alpha` of package **irr** (Gamer et al., 2012), function `kripp.boot` of package **kripp.boot** (Proutskova and Gruszczynski, 2020), function `krippalpha` of package **icr** (Staudt and L'Ecuyer, 2020), and functions `krippen.alpha.raw` and `krippen.alpha.dist` of package **irrCAC** (Gwet, 2019). However, these packages fail to provide a number of useful features. In this article we present package **krippendorffsalpha**, which improves upon the above mentioned packages in (at least) the following ways. Package **krippendorffsalpha**

- offers commonly used built-in distance functions for the nominal, ordinal, interval, and ratio levels of measurement and also supports user-defined distance functions;
- conforms to the R idiom by providing S3 methods `confint`, `influence`, `plot`, and `summary`;
- supports embarrassingly parallel bootstrap computation; and
- supports verbose communication with the user, including the display of a progress bar during the production of the bootstrap sample.

The remainder of this article is organized as follows. In Section 2.2, we locate Krippendorff's α among statistical procedures. In Section 2.2.1, we first develop a special case of Krippendorff's α (call it α_{SED}) in a well-known parametric setting, and then we present α in its most general (i.e., nonparametric) form. In Section 2.2.2, we show that α is a type of multiresponse permutation procedure. In Section 2.2.3, we generalize α_{SED} in a fully parametric fashion, arriving at Sklar's ω. In Section 2.3, we describe our package's bootstrap inference for α and compare the performance of our procedure to that of two alternative approaches. In Section 2.4, we briefly discuss robustness and influence. In Section 2.5, we provide a thorough demonstration of **krippendorffsalpha**'s usage before concluding in Section 2.6.

Situating Krippendorff's Alpha among statistical procedures

Since Krippendorff's α is defined in terms of discrepancies (Krippendorff, 2013), at first glance, one might conclude, erroneously, that α is a measure of *dis*-agreement, and so answers the wrong question. In Sections 2.2.1–2.2.3, we will show, by examining Krippendorff's α's place among statistical procedures, that α is, in fact, a sensible measure of agreement. Also, establishing a context for α may help practitioners make educated decisions regarding α's use.

The UML class diagram (Fowler et al., 2004) shown below in Figure 1 provides a conceptual roadmap for our development. Briefly, a special case of α (which we denote as Alpha(SED) or α_{SED}) arises naturally in the context of the one-way mixed-effects ANOVA model. Alpha(SED) can then be generalized in a nonparametric fashion to arrive at Krippendorff's α as it has been presented by Hayes and Krippendorff (see Gwet (2015) for development of nonparametric α in terms of agreement rather

than discrepancies); this nonparametric form of α is a (slightly modified) multiresponse permutation procedure. Alternatively, α_{SED} can be generalized in a parametric fashion to arrive at Sklar's ω, a Gaussian copula-based methodology for measuring agreement.

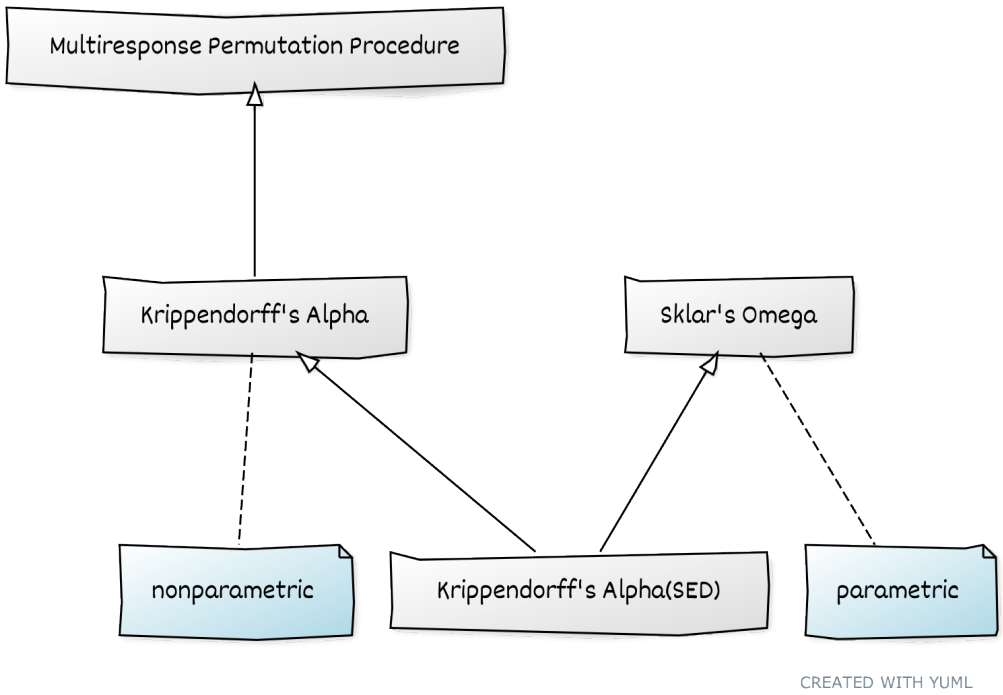


Figure 1: A UML class diagram that shows the relationships between Krippendorff's α and other statistical procedures.

Parametric genesis of Krippendorff's Alpha coefficient

In this section, we develop Krippendorff's α (Hayes and Krippendorff, 2007) in an intuitive and bottom-up fashion. Our starting point is a fully parametric model, namely, the classic one-way mixed-effects ANOVA model (Ravishanker and Dey, 2001). To ease exposition, we will consider only a balanced version of the model. We have, for n_u units and n_c coders, scores

$$Y_{ij} = \mu + \tau_i + \varepsilon_{ij}, \qquad (i = 1, \ldots, n_u;\ j = 1, \ldots, n_c),$$

where

- μ (the population mean) is a fixed real number,
- the τ_i are independent $\mathcal{N}(0, \sigma_\tau^2)$ random variables,
- the ε_{ij} are independent $\mathcal{N}(0, \sigma_\varepsilon^2)$ random variables, and
- the τ_i are independent of the ε_{ij}.

In this setup, we have n_c Gaussian codes $Y_{i1}, \ldots, Y_{in_c}$ for unit $i \in \{1, \ldots, n_u\}$. Conditional on τ_i, said codes are $\mathcal{N}(\mu + \tau_i, \sigma_\varepsilon^2)$ random variables. Since the variables share the "unit effect" τ_i, the variables are correlated. The correlation, which is usually called the *intraclass correlation*, is given by

$$\alpha = \frac{\sigma_\tau^2}{\sigma_\tau^2 + \sigma_\varepsilon^2} = 1 - \frac{\sigma_\varepsilon^2}{\sigma_\tau^2 + \sigma_\varepsilon^2}.$$

We use α to denote this quantity precisely because Krippendorff's α is the intraclass correlation for codes that conform to this model. That is, for the one-way mixed-effects ANOVA model, Krippendorff's α is the intraclass correlation. The reader may recall that the estimator of α for this model is

$$\hat{\alpha} = 1 - \frac{\widehat{\sigma_\varepsilon^2}}{\widehat{\sigma_\tau^2 + \sigma_\varepsilon^2}} = 1 - \frac{\frac{1}{n_u(n_c-1)} \sum_{i=1}^{n_u} \sum_{j=1}^{n_c} (Y_{ij} - \bar{Y}_{i\bullet})^2}{\frac{1}{n_u n_c - 1} \sum_{i=1}^{n_u} \sum_{j=1}^{n_c} (Y_{ij} - \bar{Y}_{\bullet\bullet})^2}, \tag{1}$$

where $\bar{Y}_{i\bullet}$ and $\bar{Y}_{\bullet\bullet}$ denote the arithmetic means for the ith unit and for the entire sample, respectively. The form of this estimator is not surprising, of course, since it is well-known that assuming Gaussianity leads to variance estimators involving sums of weighted squared deviations from sample arithmetic means.

We can eliminate the arithmetic means in (1) by employing the identity

$$\sum_{i=1}^{n}(x_i - \bar{x}_\bullet)^2 = \frac{1}{2n}\sum_{i=1}^{n}\sum_{j=1}^{n}(x_i - x_j)^2.$$

This gives

$$\hat{\alpha} = 1 - \frac{\frac{1}{2n_u n_c(n_c-1)}\sum_{i=1}^{n_u}\sum_{j=1}^{n_c}\sum_{k=1}^{n_c}(Y_{ij} - Y_{ik})^2}{\frac{1}{2n_u n_c(n_u n_c-1)}\sum_{i=1}^{n_u}\sum_{j=1}^{n_c}\sum_{k=1}^{n_u}\sum_{l=1}^{n_c}(Y_{ij} - Y_{kl})^2}. \tag{2}$$

Now, let $d^2(x,y) = (x-y)^2$, and rewrite (2) as

$$\hat{\alpha} = 1 - \frac{D_o}{D_e} = 1 - \frac{\frac{1}{2n_u n_c(n_c-1)}\sum_{i=1}^{n_u}\sum_{j=1}^{n_c}\sum_{k=1}^{n_c} d^2(Y_{ij}, Y_{ik})}{\frac{1}{2n_u n_c(n_u n_c-1)}\sum_{i=1}^{n_u}\sum_{j=1}^{n_c}\sum_{k=1}^{n_u}\sum_{l=1}^{n_c} d^2(Y_{ij}, Y_{kl})}, \tag{3}$$

where D_o and D_e denote observed and expected disagreement, respectively. This is Krippendorff's α for the squared Euclidean distance (which is not a metric but a Bregman divergence (Bregman, 1967)). We will henceforth refer to this version of α as Alpha(SED) or α_{SED}. As we mentioned above, this form of Krippendorff's α arises quite naturally when the data at hand conform to the one-way mixed-effects ANOVA model, for which agreement corresponds to a positive correlation. More generally, Krippendorff recommends this form of α for the interval level of measurement. For other levels of measurement, Krippendorff presents other distance functions d^2 (several possibilities are shown in Table 1). Note that package **krippendorffsalpha** supports user-defined distance functions as well as the interval, nominal, and ratio distance functions shown in the table.

Level of Measurement	Distance Function
interval	$d^2(x,y) = (x-y)^2$
nominal	$d^2(x,y) = 1\{x \neq y\}$
ratio	$d^2(x,y) = \left(\frac{x-y}{x+y}\right)^2$
bipolar	$d^2(x,y) = \frac{(x-y)^2}{(x+y-2x_{\min})(2x_{\max}-x-y)}$
circular	$d^2(x,y) = \left\{\sin\left(\pi\frac{x-y}{I}\right)\right\}^2$ (I = number of equal intervals on circle)
ordinal	$d^2(x,y) = (x-y)^2$ (adjacent ranks are equidistant)

Table 1: Several distance functions that may be appropriate for use in Krippendorff's α.

Alpha as a multiresponse permutation procedure

In any case, (3) is nonparametric for arbitrary d^2 since then the estimator $\hat{\alpha}$ does not usually correspond to a well-defined population parameter α. This more general form of Krippendorff's α is, in fact, a special case of the so-called multiresponse permutation procedure (MRPP). The MRPPs form a class of permutation methods for discerning differences among two or more groups in one or more dimensions (Mielke and Berry, 2007). Note, however, that although α can be viewed as an MRPP (as we are about to show), α has been modified for the purpose of measuring agreement rather than discerning differences.

To show that Krippendorff's α is an MRPP, we first present the general form of the MRPP. Following Mielke and Berry, let $\Omega = \{\omega_1, \ldots, \omega_n\}$ be a finite sample that is representative of some population of interest, let $S_1, \ldots, S_{a+1}$ denote a partition of Ω into $a+1$ disjoint groups, and let ρ be a metric that makes sense for the objects of Ω. (Strictly speaking, ρ need not be a metric; a symmetric distance function will suffice.) To ease notation a bit, let $\rho_{jk} \equiv \rho(\omega_j, \omega_k)$. Then, the MRPP statistic can be

written as

$$\delta = \sum_{i=1}^{a} C_i \theta_i,$$

where $C_i > 0$ are group weights such that $\sum_i C_i = 1$;

$$\theta_i = \frac{1}{\binom{n_i}{2}} \sum_{j<k} \rho_{jk} 1_i\{\omega_j\} 1_i\{\omega_k\}$$

is the average distance between distinct pairs of objects in group S_i; $n_i \geq 2$ is the number of objects in group i; $l = \sum_{i=1}^{a} n_i$; $n_{a+1} = n - l \geq 0$ is the number of remaining unclassified objects in group S_{a+1}; and 1_i is the indicator function for membership in group i.

Note that this formulation is quite general since the objects in Ω can be scalars, vectors, or more exotic objects, and we are free to choose the metric and the weights. In the case of Krippendorff's α, we can produce $\delta = D_o$ by letting $\rho = d^2$ for some appropriately chosen distance function d^2 and choosing weights $C_i = 1/n_u$.

A parametric generalization of Alpha(SED)

In the preceding sections, we generalized α_{SED} in a nonparametric fashion by substituting other notions of distance for the squared Euclidean distance. Now, we will present a fully parametric generalization of α_{SED}, namely, Sklar's ω (Hughes, 2018).

The statistical model underpinning Sklar's ω is a Gaussian copula model (Xue-Kun Song, 2000). The most general form of the model is given by

$$\begin{aligned} \boldsymbol{Z} = (Z_1, \ldots, Z_n)' &\sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{\Omega}) \\ U_i = \Phi(Z_i) &\sim \mathcal{U}(0,1) \qquad (i = 1, \ldots, n) \\ Y_i = F_i^{-1}(U_i) &\sim F_i, \end{aligned} \tag{4}$$

where $\boldsymbol{\Omega}$ is a correlation matrix, Φ is the standard Gaussian cdf, and F_i is the cdf for the ith outcome Y_i. Note that $\boldsymbol{U} = (U_1, \ldots, U_n)'$ is a realization of the Gaussian copula, which is to say that the U_i are marginally standard uniform and exhibit the Gaussian correlation structure defined by $\boldsymbol{\Omega}$. Since U_i is standard uniform, applying the inverse probability integral transform to U_i produces outcome Y_i having the desired marginal distribution F_i.

To see that the one-way mixed-effects ANOVA model (and hence α_{SED}) is a special case of Sklar's ω, let the copula correlation matrix $\boldsymbol{\Omega}$ be block diagonal, where the ith block corresponds to the ith unit $(i = 1, \ldots, n_u)$ and has a compound symmetry structure. That is,

$$\boldsymbol{\Omega} = \operatorname{diag}(\boldsymbol{\Omega}_i),$$

where

$$\boldsymbol{\Omega}_i = \begin{array}{c} \\ c_1 \\ c_2 \\ \vdots \\ c_{n_c} \end{array} \begin{array}{c} \begin{array}{cccc} c_1 & c_2 & \ldots & c_{n_c} \end{array} \\ \begin{pmatrix} 1 & \omega & \ldots & \omega \\ \omega & 1 & \ldots & \omega \\ \vdots & \vdots & \ddots & \vdots \\ \omega & \omega & \ldots & 1 \end{pmatrix} \end{array}.$$

Complete the specification by letting F_{ij} $(i = 1, \ldots, n_u;\ j = 1, \ldots, n_c)$ be the cdf for the Gaussian distribution with mean μ and variance σ^2. Then $\omega = \alpha$, the intraclass correlation coefficient.

Inference for Krippendorff's Alpha

Mielke and Berry describe hypothesis testing for MRPPs. Specifically, they discuss three approaches: permutation, Monte Carlo resampling, and Pearson type III moment approximation. The latter has significant advantages. For Krippendorff's α, though, we are interested not in hypothesis testing but in interval estimation. This can be done straightforwardly and efficiently using Monte Carlo resampling. Since D_e is invariant to permutation of the scores, our resampling procedure focuses on D_o only. The algorithm proceeds as follows.

1. Collect the scores in an $n_u \times n_c$ matrix, $\mathbf{A}$, where each row corresponds to a unit.

2. For $i \in \{1, \ldots, n_b\}$, form matrix $\mathbf{A}_i$ by sampling, with replacement, n_u rows from $\mathbf{A}$.
3. For each $\mathbf{A}_i$, compute $D_o^{(i)}$ using the same distance function d^2 that was used to compute $\hat{\alpha}$.
4. For each $D_o^{(i)}$, compute $\hat{\alpha}_i = 1 - D_o^{(i)} / D_e$.

The resulting collection $\{\hat{\alpha}_1, \ldots, \hat{\alpha}_{n_b}\}$ is a bootstrap sample for $\hat{\alpha}$, sample quantiles of which are estimated confidence limits for α.

We carried out a number of realistic simulation experiments and found that this approach to interval estimation performs well in a wide variety of circumstances. When the true distribution of $\hat{\alpha}$ is (at least approximately) symmetric, Gwet's closed-form expression for $\hat{\mathbb{V}}(\hat{\alpha})$, which is implemented (for categorical data only) in package **irrCAC**, also performs well. By contrast, we found that the bootstrapping procedure recommended by Krippendorff (2016), which is implemented in packages **kripp.boot** and **icr**, generally performs rather poorly, producing intervals that are far too narrow (e.g., 95% intervals achieve 74% coverage).

Robustness and interpretation

For some levels of measurement, one may, in the interest of robustness, be tempted to replace squares with absolute values (in the distance function d^2). This would be advantageous if one aimed to do hypothesis testing. But for Krippendorff's α, using absolute values instead of squares proves disastrous, for the resulting estimator $\hat{\alpha}$ is substantially negatively biased and tends to lead to erroneous inference regarding agreement. All is not lost, however, since package **krippendorffsalpha** provides a means of investigating the influence on $\hat{\alpha}$ of any unit or coder (see the next section for examples).

Illustrations

Here we illustrate the use of **krippendorffsalpha** by applying Krippendorff's α to a couple of datasets. We will interpret the results according to the ranges given in Table 2, but we suggest—as do Krippendorff and others (Artstein and Poesio, 2008; Landis and Koch, 1977)—that an appropriate reliability threshold may be context-dependent.

Range of Agreement	Interpretation
$\alpha \leq 0.2$	Slight Agreement
$0.2 < \alpha \leq 0.4$	Fair Agreement
$0.4 < \alpha \leq 0.6$	Moderate Agreement
$0.6 < \alpha \leq 0.8$	Substantial Agreement
$\alpha > 0.8$	Near-Perfect Agreement

Table 2: Guidelines for interpreting values of an agreement coefficient.

Nominal data analyzed previously by Krippendorff

Consider the following data, which appear in (Krippendorff, 2013). These are nominal values (in $\{1, \ldots, 5\}$) for twelve units and four coders. The dots represent missing values.

Note that the scores for all units except the sixth are constant or nearly so. This suggests near-perfect agreement, and so we should expect $\hat{\alpha}$ to be greater than 0.8.

To apply Krippendorff's α to these data, first we load package **krippendorffsalpha**.

```
R> library(krippendorffsalpha)

krippendorffsalpha: Measuring Agreement Using Krippendorff's Alpha Coefficient
Version 1.1 created on 2021-01-13.
copyright (c) 2020-2021, John Hughes
For citation information, type citation("krippendorffsalpha").
Type help(package = krippendorffsalpha) to get started.
```

	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_{10}	u_{11}	u_{12}
c_1	1	2	3	3	2	1	4	1	2	•	•	•
c_2	1	2	3	3	2	2	4	1	2	5	•	3
c_3	•	3	3	3	2	3	4	2	2	5	1	•
c_4	1	2	3	3	2	4	4	1	2	5	1	•

Figure 2: Some example nominal outcomes for twelve units and four coders, with seven missing values.

Now, we create the dataset as a matrix such that each row corresponds to a unit and each column corresponds to a coder.

```
R> nominal = matrix(c(1,2,3,3,2,1,4,1,2,NA,NA,NA,
+                     1,2,3,3,2,2,4,1,2,5,NA,3,
+                     NA,3,3,3,2,3,4,2,2,5,1,NA,
+                     1,2,3,3,2,4,4,1,2,5,1,NA), 12, 4)
R> nominal

      [,1] [,2] [,3] [,4]
 [1,]    1    1   NA    1
 [2,]    2    2    3    2
 [3,]    3    3    3    3
 [4,]    3    3    3    3
 [5,]    2    2    2    2
 [6,]    1    2    3    4
 [7,]    4    4    4    4
 [8,]    1    1    2    1
 [9,]    2    2    2    2
[10,]   NA    5    5    5
[11,]   NA   NA    1    1
[12,]   NA    3   NA   NA
```

Next, we apply Krippendorff's α for the nominal level of measurement. If argument `level` is set to `"nominal"`, the discrete metric $d^2(x, y) = 1\{x \neq y\}$ is used by default. We do a bootstrap with sample size $n_b = 1{,}000$ (argument `confint` defaults to `TRUE`, and control parameter `bootit` defaults to 1,000). We set control parameter `parallel` equal to `FALSE` because the dataset is too small to warrant parallelization of the bootstrap computation. Finally, we set argument `verbose` equal to `TRUE` so that a progress bar is shown during the bootstrap computation. The computation took less than one second.

```
R> set.seed(42)
R> fit.full = krippendorffs.alpha(nominal, level = "nominal", control = list(parallel = FALSE),
+                                 verbose = TRUE)

  |++++++++++++++++++++++++++++++++++++++++++++++++++| 100% elapsed=00s
```

As is customary in R, one can view a summary by passing the fit object to `summary.krippendorffsalpha`, an S3 method. If `krippendorffs.alpha` was called with `confint = TRUE`, `summary` displays a 95% confidence interval by default. The confidence level can be specified using argument `conf.level`. In any case, the quantile method (Davison and Hinkley, 1997) is used to estimate the confidence limits. Any arguments passed to `summary.krippendorffsalpha` via `...` are passed on to R's `quantile` function. This allows the user to control, for example, how the sample quantiles are computed.

```
R> summary(fit.full)

Krippendorff's Alpha

Data: 12 units x 4 coders

Call:
```

```
krippendorffs.alpha(data = nominal, level = "nominal", verbose = TRUE,
    control = list(parallel = FALSE))

Control parameters:

parallel FALSE
bootit   1000

Results:

      Estimate  Lower Upper
alpha   0.7429 0.4644     1
```

We see that $\hat{\alpha} = 0.74$ and $\alpha \in (0.46, 1.00)$. This point estimate indicates only substantial agreement, which is not what we expected. At least the interval is consistent with near-perfect agreement, but we should not take this interval too seriously since the interval is rather wide (owing to the small size of the dataset).

Perhaps the substantial disagreement for the sixth unit was influential enough to yield $\hat{\alpha} \leq 0.8$. We can use `influence.krippendorffsalpha`, another S3 method, to investigate. This function, like other R versions of `influence` (e.g., `influence.lm`, `influence.glm`), computes DFBETA statistics (Young, 2017), as illustrated below.

```
R> (inf.6 = influence(fit.full, units = 6))

$dfbeta.units
         6
-0.1141961
```

Leaving out the sixth unit yields a DFBETA statistic of -0.11, which implies that $\hat{\alpha}$ would have been 0.86. This is consistent with our initial hypothesis.

```
R> fit.full$alpha.hat - inf.6$dfbeta.units

    alpha
0.8571429
```

Let us call `krippendorffs.alpha` again to get a new interval.

```
R> fit.sub = krippendorffs.alpha(nominal[-6, ], level = "nominal",
+                                control = list(parallel = FALSE))
confint(fit.sub)

    0.025     0.975
0.6616541 1.0000000
```

We see that excluding the sixth unit leads to $\alpha \in (0.66, 1.00)$. The new 95% interval was returned by S3 method `confint.krippendorffsalpha`, whose `level` argument defaults to 0.95, in keeping with R's other `confint` methods. Note that `confint.krippendorffsalpha`, like `summary.krippendorffsalpha`, passes any `...` arguments on to the `quantile` function.

We conclude this example by producing a visual display of our results (Figure 3). The figure was produced via a call to S3 method `plot.krippendorffsalpha`, which in turn calls `hist` and `abline`, and does not show a kernel density estimate. Function `plot.krippendorffsalpha` is capable of producing highly customized plots; see the package documentation for details. Since $\hat{\alpha}$ is close to 1 and the dataset is small, the bootstrap distribution is substantially skewed to the left. Thus, these data provide a textbook example of the importance of bootstrapping.

```
R> plot(fit.sub, xlim = c(0, 1), xlab = "Bootstrap Estimates", main = "Nominal Data",
+       density = FALSE)
```

Since the dataset used in this example has missing values, we take this opportunity to explain how the package handles missingness. First, the scores for a given unit of analysis are included in the computation only if two or more scores are present for that unit. Otherwise, the unit's row of the data matrix is simply ignored. Second, if two or more scores are present for a given unit, each `NA` for that unit is ignored in the computations for that row. This is handled both by the loop (adjusted denominator) and by the distance function, which should return 0 if either of its arguments is `NA`. In the next example, we illustrate this by way of a user-defined distance function, and of course, the package's built-in distance functions take the same approach.

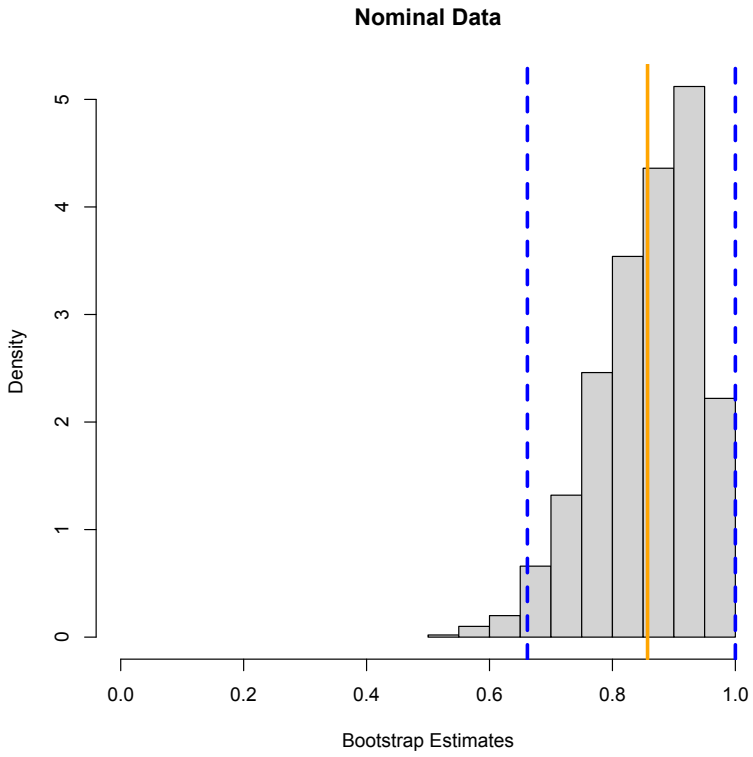


Figure 3: A plot of the results from our analysis of the nominal data. The histogram shows the bootstrap sample, the solid orange line marks the value of $\hat{\alpha}$, and the dashed blue lines mark the 95% confidence limits.

Interval data from an imaging study of hip cartilage

The data for this example, some of which appear in Figure 4, are 323 pairs of T2* relaxation times (a magnetic resonance quantity) for femoral cartilage (Nissi et al., 2015) in patients with femoroacetabular impingement (Figure 5), a hip condition that can lead to osteoarthritis. One measurement was taken when a contrast agent was present in the tissue, and the other measurement was taken in the absence of the agent. The aim of the study was to determine whether raw and contrast-enhanced T2* measurements agree closely enough to be interchangeable for the purpose of quantitatively assessing cartilage health.

	u_1	u_2	u_3	u_4	u_5	...	u_{319}	u_{320}	u_{321}	u_{322}	u_{323}
c_1	27.3	28.5	29.1	31.2	33.0	...	19.7	21.9	17.7	22.0	19.5
c_2	27.8	25.9	19.5	27.8	26.6	...	18.3	23.1	18.0	25.7	21.7

Figure 4: Raw and contrast-enhanced T2* values for femoral cartilage.

First, we load the cartilage data, which are included in the package. The cartilage data are stored in a data frame; we convert the data frame to a matrix, which is the format required by `krippendorffs.alpha`.

```
R> data(cartilage)
R> cartilage = as.matrix(cartilage)
```

Now, we compute $\hat{\alpha}$ for the interval level of measurement, i.e., squared Euclidean distance. We also produce a bootstrap sample of size 10,000. Since this dataset is much larger than the dataset analyzed in the preceding section, we parallelize the bootstrap computation. We use three CPU cores (of the four available on the author's computer). Setting argument `verbose` to `TRUE` causes the fitting function to display a progress bar once again. The computation took five seconds to complete.

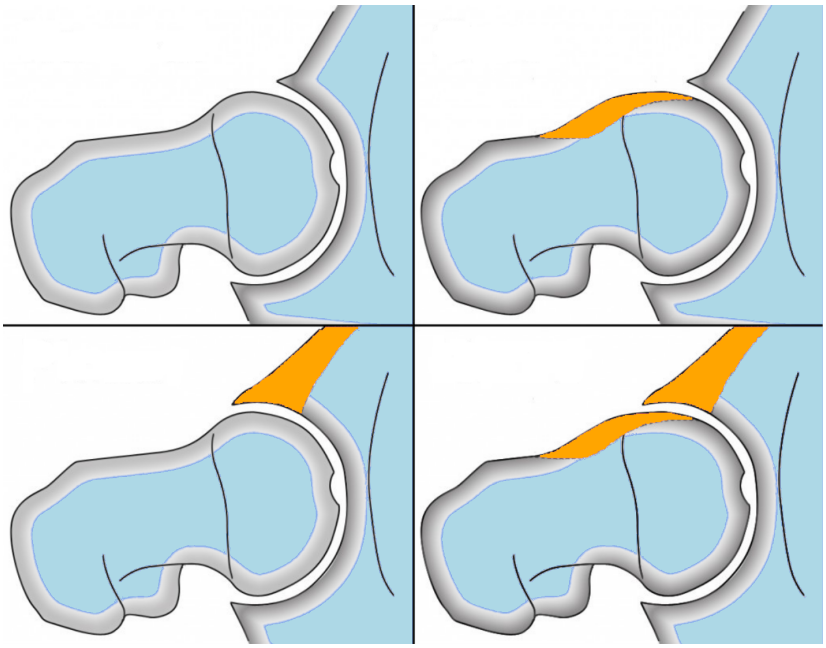


Figure 5: An illustration of femoroacetabular impingement (FAI). Top left: normal hip joint. Top right: cam type FAI (deformed femoral head). Bottom left: pincer type FAI (deformed acetabulum). Bottom right: mixed type (both deformities present).

```
R> set.seed(12)
R> fit.sed = krippendorffs.alpha(cartilage, level = "interval", verbose = TRUE,
+                                control = list(bootit = 10000, parallel = TRUE,
+                                nodes = 3))

Control parameter 'type' must be "SOCK", "PVM", "MPI", or "NWS". Setting it to "SOCK".

  |++++++++++++++++++++++++++++++++++++++++++++++++++| 100% elapsed=05s
```

A call of function `summary.krippendorffsalpha` produced the output shown below.

```
R> summary(fit.sed)

Krippendorff's Alpha

Data: 323 units x 2 coders

Call:

krippendorffs.alpha(data = cartilage, level = "interval", verbose = TRUE,
    control = list(bootit = 10000, parallel = TRUE, nodes = 3))

Control parameters:

bootit   10000
parallel TRUE
nodes    3
type     SOCK

Results:

      Estimate Lower  Upper
alpha   0.8369 0.808 0.8648
```

We see that $\hat{\alpha} = 0.84$ and $\alpha \in (0.81, 0.86)$. Thus these data suggest that raw T2* measurements agree almost perfectly with contrast-enhanced T2* measurements, perhaps rendering gadolinium-based contrast agents (GBCAs) unnecessary in T2*-based cartilage assessment. This finding could

have clinical significance since the use of GBCAs is not free of risk to patients, especially pregnant women and patients with impaired kidney function. For much additional information regarding the potential risks associated with the use of GBCAs, we refer the interested reader to the University of California, San Francisco's policy on MRI with contrast: https://radiology.ucsf.edu/patient-care/patient-safety/contrast/mri-with-contrast-gadolinium-policy.

Figure 6 provides a visual display of the cartilage results. The histogram and kernel density estimate show the expected large-sample behavior of $\hat{\alpha}$, i.e., the estimator is approximately Gaussian-distributed and has a small variance.

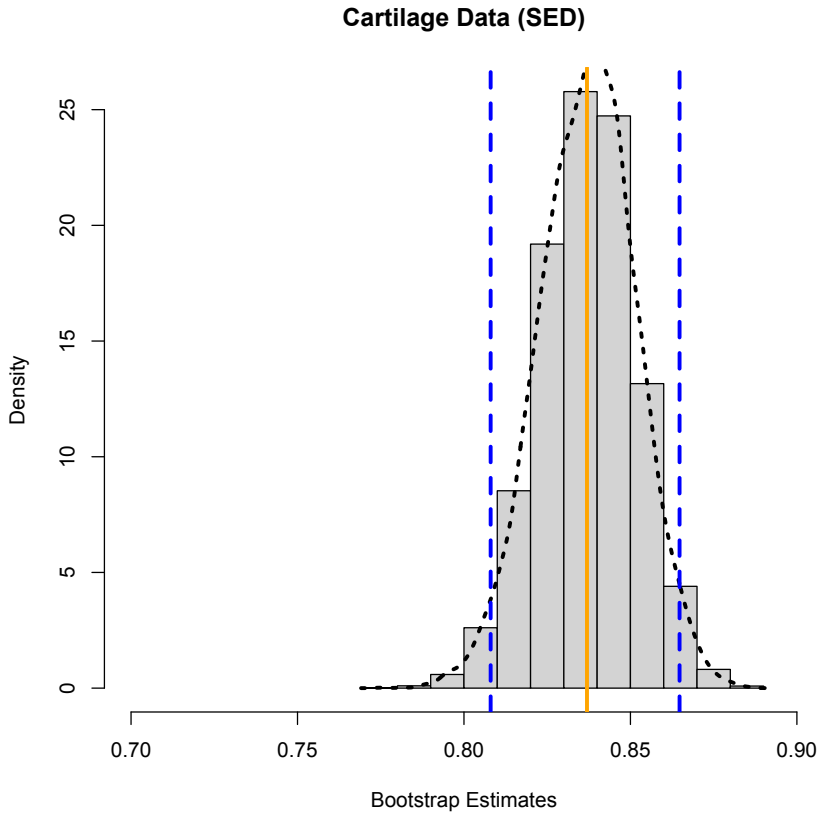


Figure 6: A plot of the results from our analysis of the cartilage data. The histogram and kernel density estimate (dotted black curve) show the bootstrap sample, the solid orange line marks the value of $\hat{\alpha}$, and the dashed blue lines mark the 95% confidence limits.

We mentioned above that attempting to robustify Krippendorff's α by using absolute values in place of squares may prove problematic. This is evident for the cartilage data, as we now demonstrate.

First, define a new distance function as follows. Note that any user-defined distance function must deal explicitly with NAs if the data at hand exhibit missingness. There are no missing values in the cartilage data, but we illustrate the handling of NA anyway.

```
R> L1.dist = function(x, y)
+ {
+    d = abs(x - y)
+    if (is.na(d))
+        d = 0
+    d
+ }
```

Now we call `krippendorffs.alpha`, supplying our new distance function via the `level` argument.

```
R> fit.L1 = krippendorffs.alpha(cartilage, level = L1.dist, verbose = TRUE,
+                               control = list(bootit = 10000, parallel = TRUE,
+                               nodes = 3))

Control parameter 'type' must be "SOCK", "PVM", "MPI", or "NWS". Setting it to "SOCK".
```

```
  |++++++++++++++++++++++++++++++++++++++++++++++++++| 100% elapsed=05s
```

The results are summarized below. These results strongly suggest that only moderate to substantial agreement exists between raw T2* measurements and contrast-enhanced T2* measurements. This contradicts not only our α_{SED} analysis but also a Sklar's ω analysis that assumed a non-central t marginal distribution to accommodate slight asymmetry.

```
R> summary(fit.L1)

Krippendorff's Alpha

Data: 323 units x 2 coders

Call:

krippendorffs.alpha(data = cartilage, level = L1.dist, verbose = TRUE,
    control = list(bootit = 10000, parallel = TRUE, nodes = 3))

Control parameters:

bootit   10000
parallel TRUE
nodes    3
type     SOCK

Results:

      Estimate  Lower Upper
alpha   0.6125 0.5761 0.648
```

Summary and discussion

In this article, we described Krippendorff's α methodology for measuring agreement and illustrated the use of R package **krippendorffsalpha**. We first established α's context among statistical procedures. Specifically, the one-way mixed-effects ANOVA model provides a natural, intuitive genesis for α as the intraclass correlation coefficient. This form of α can be generalized in a parametric fashion to arrive at Sklar's ω, or in a nonparametric fashion to arrive at the form of α presented by Krippendorff, which is a special case of the multiresponse permutation procedure.

We demonstrated the use of **krippendorffsalpha** version 1.1 by analyzing two datasets: a nominal dataset previously analyzed by Krippendorff, and a sample of raw and contrast-enhanced T2* values from an MRI study of hip cartilage. These analyses highlighted the benefits of the package, which include the use of S3 methods, parallel bootstrap computation, support for user-defined distance functions, and a means of identifying influential units and/or coders.

Computational details

The results in this paper were obtained using R 4.0.3 for macOS and the **pbapply** 1.4-2 package. R itself and all packages used (save **kripp.boot**) are available from the Comprehensive R Archive Network (CRAN) at http://CRAN.R-project.org. Package **krippendorffsalpha** may be downloaded from CRAN or from the author's GitHub repository, which can be found at https://github.com/drjphughesjr/krippendorffsalpha. Information about the author's other R packages can be found at http://www.johnhughes.org/software.html.

John Hughes
Department of Statistics
The Pennsylvania State University
University Park, PA
USA
drjphughesjr@gmail.com

Bibliography

R. Artstein and M. Poesio. Inter-coder agreement for computational linguistics. *Computational Linguistics*, 34(4):555–596, 2008. [p417]

L. M. Bregman. The relaxation method of finding the common points of convex sets and its application to the solution of problems in convex programming. *USSR Computational Mathematics and Mathematical Physics*, 7(3):200–217, 1967. [p415]

A. C. Davison and D. V. Hinkley. *Bootstrap Methods and their Application*, volume 1. Cambridge University Press, 1997. [p418]

M. Fowler, C. Kobryn, and K. Scott. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Object Technology Series. Addison-Wesley, 2004. [p413]

M. Gamer, J. Lemon, and I. F. P. Singh. *irr: Various Coefficients of Interrater Reliability and Agreement*, 2012. URL https://CRAN.R-project.org/package=irr. R package version 0.84. [p413]

K. L. Gwet. *Handbook of Inter-Rater Reliability: The Definitive Guide to Measuring the Extent of Agreement Among Raters*. Advanced Analytics, LLC, Gaithersburg, MD, 4th edition, 2014. [p413]

K. L. Gwet. On Krippendorff's alpha coefficient. October 2015. [p413, 417]

K. L. Gwet. *irrCAC: Computing Chance-Corrected Agreement Coefficients (CAC)*, 2019. URL https://CRAN.R-project.org/package=irrCAC. R package version 1.0. [p413]

A. F. Hayes and K. Krippendorff. Answering the call for a standard reliability measure for coding data. *Communication Methods and Measures*, 1(1):77–89, 2007. [p413, 414]

J. Hughes. Sklar's Omega: A Gaussian copula-based framework for assessing agreement. *arXiv: Methodology*, 2018. [p416]

R. Ihaka and R. Gentleman. R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, 5:299–314, 1996. [p413]

K. Krippendorff. Computing Krippendorff's alpha-reliability. Technical report, University of Pennsylvania, 2013. [p413, 417]

K. Krippendorff. Bootstrapping distributions for Krippendorff's alpha. Technical report, The University of Pennsylvania, 2016. [p417]

J. R. Landis and G. G. Koch. The measurement of observer agreement for categorical data. *Biometrics*, pages 159–174, 1977. [p417]

P. W. Mielke and K. J. Berry. *Permutation Methods: A Distance Function Approach*. Springer Series in Statistics. Springer, New York, 2nd edition, 2007. [p415, 416]

M. J. Nissi, S. Mortazavi, J. Hughes, P. Morgan, and J. Ellermann. T2* relaxation time of acetabular and femoral cartilage with and without intra-articular Gd-DTPA2 in patients with femoroacetabular impingement. *American Journal of Roentgenology*, 204(6):W695, 2015. [p420]

P. Proutskova and M. Gruszczynski. *kripp.boot: Bootstrap Krippendorff's Alpha Intercoder Reliability Statistic*, 2020. URL https://github.com/MikeGruz/kripp.boot. R package version 1.0.0. [p413]

N. Ravishanker and D. Dey. *A First Course in Linear Model Theory*. Chapman & Hall/CRC Texts in Statistical Science. Taylor & Francis, 2001. [p414]

A. Staudt and P. L'Ecuyer. *icr: Compute Krippendorff's Alpha*, 2020. URL https://CRAN.R-project.org/package=icr. R package version 0.6.2. [p413]

P. Xue-Kun Song. Multivariate dispersion models generated from Gaussian copula. *Scandinavian Journal of Statistics*, 27(2):305–320, 2000. [p416]

D. S. Young. *Handbook of Regression Methods*. CRC Press, 2017. [p419]

Working with CRSP/COMPUSTAT in R: Reproducible Empirical Asset Pricing

by Majeed Simaan

Abstract It is common to come across SAS or Stata manuals while working on academic empirical finance research. Nonetheless, given the popularity of open-source programming languages such as R, there are fewer resources in R covering popular databases such as CRSP and COMPUSTAT. The aim of this article is to bridge the gap and illustrate how to leverage R in working with both datasets. As an application, we illustrate how to form size-value portfolios with respect to Fama and French (1993) and study the sensitivity of the results with respect to different inputs. Ultimately, the purpose of the article is to advocate reproducible finance research and contribute to the recent idea of "Open Source Cross-Sectional Asset Pricing", proposed by Chen and Zimmermann (2020).

Overview

Typically the CRSP and COMPUSTAT databases are viewed as the cornerstones of academic empirical finance research. The former corresponds to security-related information for publicly listed companies, such as closing prices and returns. The latter covers financial statements disclosed by public firms, such as income statement, balance sheet, and cash-flow related items. We begin our discussion by demonstrating how to clean, manipulate, and merge both datasets. After doing so, we conduct the main analysis from the perspective of empirical asset pricing research based on Fama and French (1993).

The undertaken analysis constitutes a typical portfolio formation procedure to investigate how investors are compensated for taking certain types of risk/style. By grouping firms (stocks) with respect to pre-specified attributes, the researcher can study the implications of these characteristics (risks) in association with future returns. Such analysis is known as the cross-section of expected return (see, e.g., Harvey et al. (2016)) since the relationship is investigated on the firm/portfolio level. Consistent with Harvey et al. (2016), we find that the results of the cross-section of expected returns are sensitive to the research design. In particular, the exclusion of small stocks in the sample has a significant economic impact. Small stocks tend to trade less frequently and to be less liquid. Hence, in order to better understand the cross-section of expected returns, one needs to take into consideration the underlying limits of arbitrage facing investors (Li et al., 2014).

We hope this article would further contribute to reproducible finance research and to our understandings of the cross-sectional of expected returns. The article proceeds as follows. In Section 2.2, we discuss how to load each dataset along with the pre-analysis needed in order to merge the data altogether. Section 2.3 is devoted to the replication of Fama and French (1993)'s size and value premiums. The analysis is conducted using raw and risk-adjusted returns. Section 2.4 then proceeds to investigate the sensitivity of the cross-section of returns with respect to the research design. Finally, Section 2.6 concludes.

Data

The discussion assumes that the user has already downloaded the CRSP and the COMPUSTAT datasets in two separate files, both in csv formats.[1] The article will rely on different libraries to perform the analysis. The core packages of interest are **data.table** (Dowle and Srinivasan, 2019) and **lubridate** (Grolemund and Wickham, 2011). We also refer to **ggplot2** (Wickham, 2016) to produce figures and **parallel** (R Core Team, 2020) to perform parallel processing. In a few cases, we refer to **plyr** (Wickham, 2011) and **dplyr** (Wickham et al., 2020) for additional data manipulation. Nonetheless, the main analysis is conducted in the **data.table** environment.

```
library(data.table)
library(lubridate)
library(ggplot2)
library(plyr)
library(parallel)
rm(list = ls())
```

[1] Note that users can work with the database directly using the WRDS API.

CRSP

While there are a lot of pros to working **data.table**, one good functionality is the option that allows the user to easily specify a subset of variables while reading the whole data. Mostly, after downloading the full dataset, we focus on a subset of variables of interest. We refer to its main function `fread`, rather than the base command `read.csv`. The `fread` is similar to the base command; however, it provides faster and more convenient data manipulation. It is highly relevant when it comes to large data. Additionally, it allows users to easily utilize multi-threads using the `nThread` argument.

```
file.i <- "CRSP_1960_2019.csv"
select.var <- c("PERMCO","date","COMNAM","SHROUT","SHRCD","DLRET",
                "DLSTCD","DLPDT","EXCHCD","RET","PRC","CUSIP")
DT  <- fread(file.i,select = select.var)
```

The above commands load the monthly CRSP dataset with pre-specified variables. Those are the permanent identifier of the security (`PERMCO`), date, shares outstanding in thousands (`SHROUT`), share code (`SHRCD`), exchange code (`EXCHCD`), security return `RET`, price `PRC`, and the `CUSIP` identifier. Note that the CUSIP is the key link between the CRSP and COMPUSTAT data. Additionally, we consider delisting-related variables denoted by `DL`, which are discussed later.

Filters and Cleaning

After loading the data, we perform a few filters and cleaning procedures. In particular, we keep common shares, those with 10 or 11 codes. We drop missing values for prices. There are also certain flags for prices denoted by -44, -55, -66, -77, -88, and -99. We drop these from the data as well. Additionally, we keep securities listed on major exchanges (NYSE, AMEX, or NASDAQ). Given these filters, we compute the market cap for each stock-month in the data.

```
DT <- DT[DT$SHRCD %in% 10:11,]
DT <- DT[!is.na(DT$PRC),]
DT <- DT[!DT$PRC %in% (-(4:9)*11),]
DT$RET <- as.numeric(DT$RET)
DT <- DT[!is.na(DT$RET),]
DT$PRC <- abs(DT$PRC)
DT$MKTCAP <- DT$PRC*DT$SHROUT
DT <- DT[DT$EXCHCD %in% 1:3,]
```

There may be duplicates in the data depending on the identifier of interest. To control for this, consider the following commands:

```
DT <- unique(DT)
DT <- DT[order(DT$PERMCO,DT$date),]
DT[ , `:=`( duplicate_N = .N   ) , by= list(PERMCO,date)]
table(DT$duplicate_N)/nrow(DT)*100
```

```
#>            1            2            3            4            6            7
#> 98.014830993  1.902150108  0.044272559  0.018337003  0.010738451  0.009670885
```

Note that the `:=` command creates a new variable to the already existing **data.table**, where `.N` denotes the data length. By grouping the data based on the stock identifier and month, we can count whether there are multiple observations within each case. As we observe from the above table, about 2% of the observations are duplicates, i.e., multiple observations for the same stock-month. For instance, the table below illustrates the case in which we have 7 duplicates:

```
head(DT[DT$duplicate_N == 7,],7)
```

```
#>    PERMCO     date                      COMNAM SHROUT SHRCD DLRET DLSTCD DLPDT
#> 1:  54311 20160531 LIBERTY MEDIA CORP 3RD NEW  25569    11            NA    NA
#> 2:  54311 20160531 LIBERTY MEDIA CORP 3RD NEW  55684    11            NA    NA
#> 3:  54311 20160531 LIBERTY MEDIA CORP 3RD NEW  10228    11            NA    NA
#> 4:  54311 20160531 LIBERTY MEDIA CORP 3RD NEW  22284    11            NA    NA
#> 5:  54311 20160531 LIBERTY MEDIA CORP 3RD NEW 102277    11            NA    NA
#> 6:  54311 20160531 LIBERTY MEDIA CORP 3RD NEW   9871    11            NA    NA
#> 7:  54311 20160531 LIBERTY MEDIA CORP 3RD NEW 222735    11            NA    NA
#>    EXCHCD       RET   PRC    CUSIP    MKTCAP duplicate_N
#> 1:      3  0.064481 19.48 53122987  498084.1           7
```

```
#> 2:      3  0.052778 18.95 53122985 1055211.8          7
#> 3:      3  0.081873 15.56 53122970  159147.7          7
#> 4:      3  0.095723 15.00 53122988  334260.0          7
#> 5:      3 -0.026854 31.89 53122940 3261613.5          7
#> 6:      3 -0.026292 32.59 53122950  321695.9          7
#> 7:      3 -0.017801 31.45 53122960 7005015.8          7
```

Note that the duplicates arise due to different CUSIP identifiers. Since we are planning to link the data with the COMPUSTAT using CUSIP, we check whether there are duplicates for the same CUSIP-month:

```
DT[ , `:=`( duplicate_N = .N   ) , by= list(CUSIP,date)]
table(DT$duplicate_N)/nrow(DT)*100

#>   1
#> 100

DT$duplicate_N <- NULL
```

In all cases, observe that there is a unique CUSIP-date observation. However, if one is working with CRSP alone, it is common to aggregate duplicates by value-weighting the observations based on market cap to yield a unique PERMCO-month observation. Additionally, note that the first 6 CUSIP characters result in the same unique identifier.

In terms of date formatting, we utilize the **lubridate** library to manipulate dates:

```
DT$date <- ymd(DT$date)
DT$date <- ceiling_date(DT$date,"m") - 1
```

It is common to require a minimum history of each security in the data. For instance, a researcher may need to estimate the market beta on a rolling window using an initial sample of 2 years. For the sake of illustration, we require that each security should have at least two months of data (2 observations). Nonetheless, dropping observations in the following manner could have major implications in terms data-snooping and survivor-ship bias. In Section 2.4, we discuss the sensitivity of the portfolio results to this input.

```
crsp_keep <- 2
DT[ , `:=`( N_obs = .N   ) , by= list(CUSIP)]
DT <- DT[DT$N_obs >= crsp_keep,]
DT$N_obs <- NULL
```

Finally, we have 24,581 unique securities, 720 months, and a total of 3,184,762 security-month observations.

Delisted Returns

An important characteristic of the CRSP database is that it includes historical companies that were delisted in the past. Not controlling for such delisting creates a survivor-ship bias, especially when it comes to researching the cross-section of stock returns - see, e.g., Beaver et al. (2007) for further information. To control for delisted returns, we follow the methodology recommended by Bali et al. (2016). We demonstrate these steps below. Before we do so, we take a quick look at the summary statistics of the current monthly returns:

```
summary(DT$RET)

#>     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
#> -0.99360 -0.06667  0.00000  0.01155  0.07143 24.00000

DT$DLRET <- as.numeric(DT$DLRET)
DT <- DT[order(DT$CUSIP,DT$date),]
DT[,`:=` (last_date = date[.N]),by = list(CUSIP) ]
DT[DT$DLSTCD %in% 100,"DLSTCD"] <- NA
cusip_delist <- unique(DT[!is.na(DT$DLSTCD),])$CUSIP

DT_delist <- DT[DT$CUSIP %in% cusip_delist,]
DT_delist_last <- DT_delist[, .SD[.N],  by= list(CUSIP)]
rm(DT_delist)
DT_delist_last$date <- ceiling_date(DT_delist_last$date + 1,"m")- 1
```

```
DT_delist_last$RET <- DT_delist_last$DLRET

na_dlret_index <- is.na(DT_delist_last$RET)
DT_delist_last_na <- DT_delist_last[na_dlret_index,]
DT_delist_last <-DT_delist_last[!na_dlret_index,]

select_code <- c(500,520:551,573,574,580,584)
DT_delist_last_na[DT_delist_last_na$DLSTCD %in% select_code,"RET"] <- -0.3
DT_delist_last_na[!DT_delist_last_na$DLSTCD %in% select_code,"RET"] <- -1

DT_delist_last <- rbind(DT_delist_last,DT_delist_last_na)
DT_delist_last$MKTCAP <- DT_delist_last$MKTCAP*(1 + DT_delist_last$RET)
DT_delist_last$PRC <- DT_delist_last$PRC*(1 + DT_delist_last$RET)

DT <- rbind(DT,DT_delist_last)
DT <- DT[order(DT$CUSIP,DT$date),]
summary(DT$RET) %$

#>     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
#> -1.00000 -0.06667  0.00000  0.01151  0.07143 24.00000
```

The above steps adjust for delisted returns. When available, it takes into consideration the delisting returns provided by CRSP. Otherwise, we use an arbitrary return according to the suggestion by Bali et al. (2016). We can see that the minimum return becomes -100%. However, at the same time, we observe that the mean return stays roughly the same due to the large sample size. It is also worth mentioning that dropping stocks below a certain price level potentially eliminates outliers and penny stocks that are more likely to get delisted.

Simple Portfolio Formation

Before we merge CRSP with COMPUSTAT, let us perform some basic analysis. For instance, we can easily aggregate the security return on the monthly level to create either a value-weighted or equally weighted portfolios. To do so, consider the following commands:

```
PORT_RET <- DT[,list(EW_RET = lapply(.SD,mean,na.rm = TRUE)),
               by = list(date), .SDcols = "RET"]
PORT_RET2 <- DT[,list(VW_RET = lapply(.SD,
                                      function(x) sum(x*MKTCAP/sum(MKTCAP,na.rm = TRUE),na.rm = TRUE))),
                by = list(date), .SDcols = "RET"]
PORT_RET <- merge(PORT_RET,PORT_RET2)
rm(PORT_RET2)
PORT_RET <- PORT_RET[order(PORT_RET$date),]
```

To summarize the returns over time, consider the time series of cumulative returns for each portfolio. We refer to the **ggplot2** library and the National Bureau of Economic Research (NBER) recession periods to do so:

```
bar.col <- "gray"
ggplot_recession0 <- geom_rect(fill = bar.col,col = bar.col,
                               aes(xmin=date("1973-11-30"),
                                   xmax=date("1975-03-31"),
                                   ymin=-Inf, ymax=Inf))
ggplot_recession1 <- geom_rect(fill = bar.col,col = bar.col,
                               aes(xmin=date("1980-01-31"),
                                   xmax=date("1980-07-31"),
                                   ymin=-Inf, ymax=Inf))
ggplot_recession2 <- geom_rect(fill = bar.col,col = bar.col,
                               aes(xmin=date("1981-07-31"),
                                   xmax=date("1982-11-30"),
                                   ymin=-Inf, ymax=Inf))
ggplot_recession3 <- geom_rect(fill = bar.col,col = bar.col,
                               aes(xmin=date("1990-07-31"),
                                   xmax=date("1991-03-31"),
                                   ymin=-Inf, ymax=Inf))
ggplot_recession4 <- geom_rect(fill = bar.col,col = bar.col,
```

Figure 1: This figure demonstrates the cumulative return of equally-weighted (red line) and value-weighted (blue line) portfolios over time. The gray bar denote the recession periods according to the National Bureau of Economic Research (NBER).


```
                              aes(xmin=date("2001-03-31"),
                                  xmax=date("2001-11-30"),
                                  ymin=-Inf, ymax=Inf))
ggplot_recession5 <- geom_rect(fill = bar.col,col = bar.col,
                              aes(xmin=date("2007-12-31"),
                                  xmax=date("2009-06-30"),
                                  ymin=-Inf, ymax=Inf))

ds.plot1 <- data.frame(date = PORT_RET$date, CumRet =  cumsum(PORT_RET$EW_RET),
                       Type = "Equally-Weighted" )
ds.plot2 <- data.frame(date = PORT_RET$date, CumRet =  cumsum(PORT_RET$VW_RET),
                       Type = "Value-Weighted" )
ds.plot <- rbind(ds.plot1,ds.plot2)
ds.plot$Type <- as.factor(ds.plot$Type)

p <- ggplot(ds.plot, aes(date, CumRet,colour = Type))
p <- p +  ggplot_recession0 + ggplot_recession1 +
  ggplot_recession2 + ggplot_recession3 +
  ggplot_recession4 + ggplot_recession5
p <- p + geom_line(alpha = 0.4)
p <- p + xlab("Date") + ylab("Cumulative Return")
p <- p + geom_abline(intercept = 0, slope = 0, color="black",  linetype="dashed", size=0.2)
p
```

We note that, overall, the equally-weighted portfolio under-performs the value-weighted one. The equally-weighted portfolio attributes greater weight to small-cap stocks. Contrary to Fama and French (1993), interestingly, we observe that the value-weighted stocks outperform the small-cap stocks. Nonetheless, the pattern became evident beginning in the late 80s. To take a closer look at the above, we group securities into size portfolios. At each month, we group securities into 5 groups based on the market cap cut-off.[2] To perform the cut-off, we refer to the `ntile` function from the **dplyr** library. In the following analysis, we proceed with equal-weighting for the sake of brevity.

```
library(dplyr)
cut.n <- 5
DT <- DT[,`:=` (Group_Size = ntile(MKTCAP,cut.n)), by = list(date)]
```

[2]Note that in empirical asset pricing studies, when forming portfolios based on a single characteristic (i.e., single-sort), the common practice is to choose $G = 10$. On the other hand, for double-sorting it is common to choose $G = 5$. This, however, depends on data availability. In our case, we choose $G = 5$ for both brevity and consistency (see, e.g., Fama and French (1993)).

```
N_G <- DT[,.N, by = list(date,Group_Size) ]
N_G[,mean(N), by = list(Group_Size)]
```

```
#>    Group_Size       V1
#> 1:          2 885.3694
#> 2:          1 885.7667
#> 3:          3 885.3875
#> 4:          4 885.3694
#> 5:          5 884.9500
```

We observe that, on average, each group contains 885 firms over the sample period. In addition to the group size, we consider the next one month return on each security. We use the `shift` function from **data.table** and apply it to each security as follows:

```
DT <- DT[order(DT$CUSIP,DT$date),]
DT <- DT[,`:=` (RET_1 = shift(RET,-1)), by = list(CUSIP)]
```

Given the above, we report summary statistics on the size group level:

```
DT_size <- DT[,lapply(.SD,mean,na.rm = TRUE),by = list(Group_Size),
              .SDcols = c("RET","RET_1","PRC","SHROUT","MKTCAP","EXCHCD")]
DT_size <- DT_size[order(DT_size$Group_Size),]
DT_size$RET_1 <- DT_size$RET_1*12
DT_size$RET <- DT_size$RET*12
DT_size
```

```
#>    Group_Size        RET     RET_1        PRC     SHROUT     MKTCAP   EXCHCD
#> 1:          1 -0.1047989 0.2177438   4.265575   7569.757   13946.99 2.704329
#> 2:          2  0.1463509 0.1069973   9.280118  11101.177   56211.45 2.592039
#> 3:          3  0.2047528 0.1200868  15.410881  15797.729  168506.54 2.396027
#> 4:          4  0.2313902 0.1266713  25.020412  26814.684  538252.87 2.039177
#> 5:          5  0.2132456 0.1225963 105.854839 167534.402 7505924.35 1.457429
```

Clearly, securities ranked in the 5th largest size group have a higher market-cap, which is associated with higher prices and shares outstanding. Additionally, note that the large stocks are more likely to be listed on NYSE (EXCHCD = 1), whereas the small-cap stocks are listed on NASDAQ (EXCHCD = 3). In terms of returns, the results are sensitive with respect to whether we consider an in-sample return or next month's return. In the in-sample, we observe that large-cap outperform small-cap. Nonetheless, the more relevant case in practice is the out-of-sample return. In the latter case, we observe that small-cap stocks outperform large-cap stocks. In annual terms, the small-cap stocks return 10% higher mean return than large-cap stocks. Ignoring transaction cost and assuming that investors rebalance their portfolios on a monthly basis, this evidence is consistent with Fama and French (1993)'s size premium. That is, investors expect a higher return for investing in small-cap stocks.

COMPUSTAT

The above discussion relates to the CRSP data alone. In the following, we focus on the COMPUSTAT data, which contains accounting-related information for public firms. Similar to the above, we focus on a subset of variables. For identification, we look into the CUSIP and the CIK number. The latter is relevant to identify firms via the SEC EDGAR system - for those interested in merging the data with SEC filings and perform textual analysis. The `FYEARQ` and `FQTR` are the fiscal year and quarter of the data point. For accounting variables, we consider total assets (`atq`), net income (`niq`), and common equity `ceqq`.

```
file.j <- "COMPUSTAT_1960_2020.csv"
select.var2 <- tolower(c("FYEARQ","FQTR","cusip","cik","sic","niq","atq","ceqq"))
DT2  <- unique(fread(file.j,select = select.var2))
```

Filters and Cleaning

To link between COMPUSTAT and CRSP, we need to make a small adjustment for the CUSIP identifiers. In CRSP, the number of characters is 8, whereas in COMPUSTAT, it is 9.

```
table(nchar(DT$CUSIP))
```

```
#>
#>       8
#> 3187327

table(nchar(DT2$cusip))

#>
#>       0       9
#>     130 1806228
```

Note that there a few cases in which the CUSIP is unavailable in COMPUSTAT. The adjustments are described below:

```
DT2 <- DT2[!nchar(DT2$cusip) == 0,]
DT2$CUSIP <- substr(DT2$cusip,0,8)
DT2$cusip <- NULL
DT2 <- unique(DT2[DT2$CUSIP %in% DT$CUSIP,])
```

In order to merge with the CRSP data, which corresponds to calendar dates, we adjust the fiscal dates in COMPUSTAST. It is common to use 6 months lags to allow the financial disclosures to become publicly available. To do so, we perform the following adjustments:

```
DT2$date <- ymd(DT2$fyearq*10000 + DT2$fqtr*3*100 + 1)
DT2$date <- DT2$date + months(6)
DT2$date <- ceiling_date(DT2$date,"q") - 1
DT2 <- DT2[month(DT2$date) == 6,]
DT2$fyearq <- DT2$fqtr <- NULL
```

Additionally, we keep the annual data rather than the quarterly one and consider portfolio formation on an annual basis.[3] In particular, we keep the June data for the portfolio formation process.

Same as before, let us check for duplicates:

```
DT2 <- unique(DT2)
DT2[ , `:=`( duplicate_N = .N   ) , by= list(CUSIP,date)]
table(DT2$duplicate_N)

#>
#>      1
#> 247903

DT2$duplicate_N <- NULL
```

The COMPUSTAT dataset have unique CUSIP-date observations.

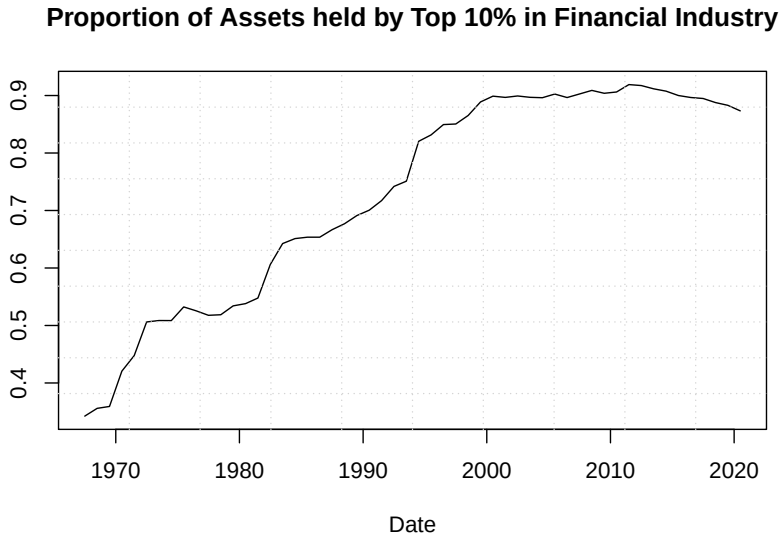
Summary Statistics

Given the final COMPUSTAT data, we consider a few summary statistics for each industry, which is defined using the 4th digit of the SIC code.

```
DT2 <- na.omit(DT2)
DT2$ROA <- DT2$niq/DT2$atq
DT2$ROE <- DT2$niq/DT2$ceqq
DT2$BL <- DT2$atq/DT2$ceqq
DT2$Industry <- floor(DT2$sic/1000)

DT2_sum <- DT2[,lapply(.SD,median,na.rm = TRUE), by = list(Industry),
               .SDcols = c("atq","ROA","ROE","BL")]
DT2_sum <- DT2_sum[order(DT2_sum$Industry),]
DT2_sum

#>     Industry      atq          ROA          ROE       BL
#>  1:        0  97.9455 -0.005973224 -0.007163557 2.005072
#>  2:        1 152.8265  0.003908058  0.015030814 2.031069
#>  3:        2 112.5905  0.006157001  0.023091766 1.761200
#>  4:        3  85.0685  0.010216054  0.024022369 1.736357
```

[3] Clearly, it all depends on the purpose of the investigation.

Figure 2: This figure below demonstrates the concentration in the financial industry. It illustrates the book value of assets held by the top 10% banks relative to the total assets held by the financial industry.




```
#>  5:        4 540.3000  0.006908974  0.024877318 2.914309
#>  6:        5 148.0990  0.011941449  0.030180501 2.155898
#>  7:        6 893.6370  0.002491525  0.026317412 8.965721
#>  8:        7  88.5455  0.004830920  0.018396345 1.736342
#>  9:        8  73.5180  0.006937877  0.020926220 1.904278
#> 10:        9  37.3550 -0.003438435  0.006888589 1.562943
```

We observe that financial firms are associated with the highest book leverage, given the business nature of financial institutions. Also, we note that the financial industry is associated with the largest total assets. This is not surprising given the concentration of the financial industry over time, in which a few entities hold the majority of assets. To relate to the last point, consider the total assets of the top 10% firms with respect to the total assets in the industry as a whole.

```
DT2_Fin <- DT2[DT2$Industry == 6,]
DT2_Fin <- DT2_Fin[,`:=` (Group_Size = ntile(atq,10)), by = list(date)]
DT2_Fin <- DT2_Fin[,`:=` (Total_Assets = sum(atq)), by = list(date)]
DT2_Fin <- DT2_Fin[,lapply(.SD, function(x) sum(x/Total_Assets)),
                   by = list(date,Group_Size), .SDcols = "atq"]
DT2_Fin_Top <- DT2_Fin[DT2_Fin$Group_Size == 10,]
DT2_Fin_Top <- DT2_Fin_Top[order(DT2_Fin_Top$date),]
plot(atq~date,data = DT2_Fin_Top,type = "l",
     main = "Proportion of Assets held by Top 10% in Financial Industry",
     ylab = "", xlab = "Date")
grid(10)
```

According to Figure 2, we discern that the top 10% increased their share of total assets from 34% in the late 60s up to 90% more recently.

CRSP-COMPUSTAT

Note that the COMPUSTAT dataset is annual, while the CRSP is monthly. If we seek to form portfolios based on book-to-market (BM) ratio, such as the case for Fama-French, we need to create an annual BM variable in the COMPUSTAT dataset. Before we merge the data altogether, let us add the market value of equity (ME) to the COMPUSTAT dataset. Additionally, we add the stock prices which would be useful for small-cap stocks from the portfolio formation later on.

```
BM <- DT[,c("date","CUSIP","MKTCAP","PRC")]
DT2 <- merge(DT2,BM, by = c("CUSIP","date"), all = F)
DT2$BM_ratio <- DT2$ceqq/(DT2$MKTCAP/1000)
```

```
DT2$ME <- DT2$MKTCAP
DT2$PRC_LAST <- DT2$PRC
DT2$MKTCAP <-  DT2$PRC <- NULL
DT2 <- DT2[order(DT2$CUSIP,DT2$date),]
DT2[,`:=` (N_years = 1:.N), by = list(CUSIP)]
rm(BM); gc()
```

```
#>            used  (Mb) gc trigger   (Mb)  max used   (Mb)
#> Ncells  1024743  54.8    2730065  145.9   2730065  145.9
#> Vcells 63598437 485.3  149041274 1137.1 148737297 1134.8
```

The above commands link the COMPUSTAT and the CRSP datasets to determine the market equity of the firms and, hence, the book-to-market ratio. Also, we denote the stock price as `PRC_LAST` to refer to the recent stock price in the annual data.

Before we finally merge the data altogether, we need to do one small trick with the CRSP data. The ultimate goal of this illustration to attribute the next 12 months' returns with respect to the BM ratio from the previous June. To do so, consider the following:

```
prev_june_f <- function(x) floor_date(floor_date(x,"m") + months(6),"y") - months(6) - 1
DT$date_june <- prev_june_f(DT$date)
DT2$date_june <- DT2$date
DT2$date <- NULL
```

While the `prev_june_f` seems obscure, one should break the function into different components in order to understand how it works. Consider the special case where we have:

```
x <- DT$date[1:13]
x
```

```
#>  [1] "1970-12-31" "1971-01-31" "1971-02-28" "1971-03-31" "1971-04-30"
#>  [6] "1971-05-31" "1971-06-30" "1971-07-31" "1971-08-31" "1971-09-30"
#> [11] "1971-10-31" "1971-11-30" "1971-12-31"
```

The first command of the function shifts the dates six months ahead:

```
floor_date(x,"m") + months(6)
```

```
#>  [1] "1971-06-01" "1971-07-01" "1971-08-01" "1971-09-01" "1971-10-01"
#>  [6] "1971-11-01" "1971-12-01" "1972-01-01" "1972-02-01" "1972-03-01"
#> [11] "1972-04-01" "1972-05-01" "1972-06-01"
```

The second step identifies the floor of each date on the annual level:

```
floor_date(floor_date(x,"m") + months(6),"y")
```

```
#>  [1] "1971-01-01" "1971-01-01" "1971-01-01" "1971-01-01" "1971-01-01"
#>  [6] "1971-01-01" "1971-01-01" "1972-01-01" "1972-01-01" "1972-01-01"
#> [11] "1972-01-01" "1972-01-01" "1972-01-01"
```

The final step subtracts 6 months to identify the June of the previous year. The minus one is added to retrieve June 30th rather than July 1st, such that:

```
floor_date(floor_date(x,"m") + months(6),"y") - months(6) - 1
```

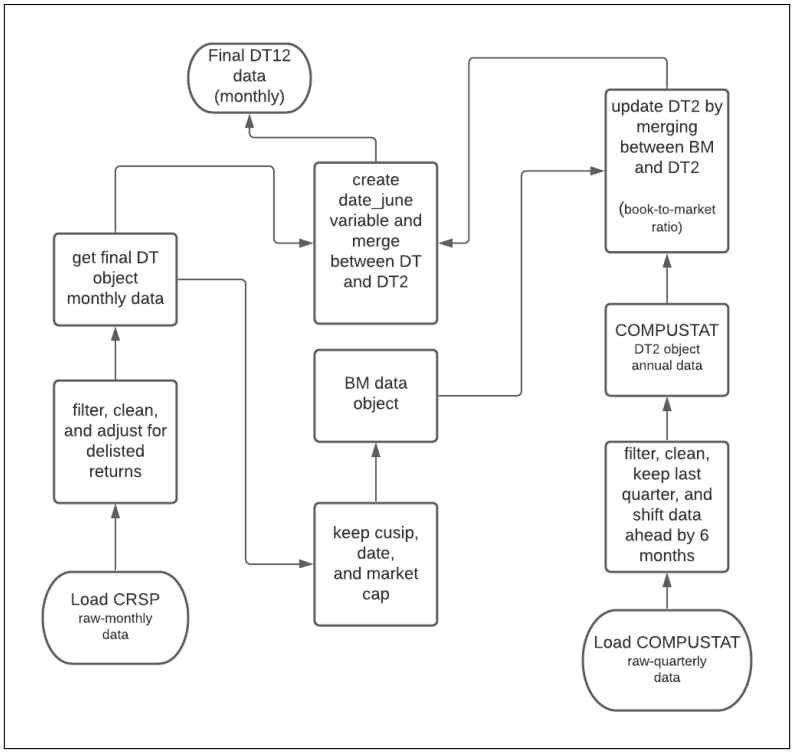
```
#>  [1] "1970-06-30" "1970-06-30" "1970-06-30" "1970-06-30" "1970-06-30"
#>  [6] "1970-06-30" "1970-06-30" "1971-06-30" "1971-06-30" "1971-06-30"
#> [11] "1971-06-30" "1971-06-30" "1971-06-30"
```

Looking at returns between Dec 1970 and Mar 1971, the function traces these observations back to June 1970 for the first 7 returns. When the next June data shows up, the `date_june` variable adjusts accordingly. Finally, the merged dataset is given by:

```
DT12 <- merge(DT,DT2,by = c("CUSIP","date_june"))
```

Note: The above steps provide details on how to merge between the two data sets. In Figure 3, we provide a flow chart summarizing the steps/processes undertaken to develop the final merged data set `DT12`.

Figure 3: The figure below illustrates a flow chart of the data merging process. It demonstrates the steps taken altogether to come up with the final CRSP-COMPUSTAT merged data set.



Forming Size and BM Portfolios

The DT12 object contains the final CRSP-COMPUSTAT data. The merging is conducted to allow for portfolio formation at the end of June each year, as mentioned above. Such a setting allows a convenient way to form portfolios without relying on loops. Nonetheless, it would also impose that portfolio managers balance their portfolios on a monthly basis according to the market cap from the previous June.

Given the merged dataset, we form groups at the end of June-year based on market equity (size) and book-to-market ratio (value):

```
DT12 <- DT12[,`:=` (Group_ME = ntile(ME,5)), by = list(date_june)]
DT12 <- DT12[,`:=` (Group_BM = ntile(BM_ratio,5)), by = list(date_june,Group_ME)]
```

The above two commands perform dependent portfolio sorting, in which firms are first sorted based on size and then on BM ratio. We consider value-weighting to compute the future returns on each size-value portfolio. This results in 25 value-weighted portfolios.

```
PORT_RET <- DT12[,list(VW_RET = sum(RET*ME/sum(ME))  ),
                 by = list(date,Group_ME,Group_BM)]
PORT_RET <- PORT_RET[order(PORT_RET$date,PORT_RET$Group_BM,PORT_RET$Group_ME),]
PORT_RET$VW_RET <- as.numeric(PORT_RET$VW_RET)*100

ret_matrix <- PORT_RET[,lapply(.SD,mean,na.rm = TRUE),
                       by = list(Group_ME,Group_BM), .SDcols = "VW_RET"]
ret_matrix <- ret_matrix[order(ret_matrix$Group_BM,ret_matrix$Group_ME),]
ret_matrix <- matrix(as.numeric(ret_matrix$VW_RET),5)
rownames(ret_matrix) <- 1:5
colnames(ret_matrix) <- paste("BM",1:5,sep = "_")
rownames(ret_matrix)[1] <- "Small"
rownames(ret_matrix)[5] <- "Big"
colnames(ret_matrix)[1] <- "Low"
colnames(ret_matrix)[5] <- "High"
round(data.frame(ret_matrix*12),2)
```

```
#>         Low  BM_2  BM_3  BM_4  High
#> Small 12.93 14.73 18.05 20.14 23.40
#> 2      8.45 12.55 14.37 16.72 16.17
#> 3      8.04 11.79 14.30 14.24 15.99
#> 4      9.89 11.53 11.92 14.47 14.77
#> Big    9.84 11.70 10.41 13.13 12.78
```

For each column above, we observe the stock returns (raw) decrease, on average, with size. This is commonly known as the size effect. At the same time, we observe that within each size group, the mean return increases with the BM ratio. The latter denotes what is known as the value effect. In other words, investors expect higher returns from small enterprises and undervalued stocks (trading below book value). Nonetheless, recent discussions debate whether this is the case. For further information on this, see this article.

Risk-Adjusted Returns using Fama-French's Risk Factors

The above portfolio results compute the raw returns. In order to price these portfolios, we decompose the returns into (1) systematic components (risk-premiums) and (2) non-systematic. The latter denotes the risk-adjusted returns. Additionally, we consider the excess return on each portfolio, i.e., the portfolio return minus the 1-month Treasury yield.

The Fama-French's risk factors are obtained easily using the Kenneth French public library according to the commands below. Note that we focus on the three factors: market, size, and value:

```
FF_file <-
  "https://mba.tuck.dartmouth.edu/pages/faculty/ken.french/ftp/F-F_Research_Data_Factors_CSV.zip"
temp <- tempfile()
download.file(FF_file,temp)
unz_files <- unzip(temp)
ds <- read.csv(unz_files,skip = 3)
flag_obs <- grep("Annual",ds[,1],ignore.case = TRUE)
ds <- ds[1:(flag_obs-1),]
names(ds)[1] <- "date"
ds <- data.frame(apply(ds, 2, as.numeric))
ds$date <- ceiling_date(ymd(ds$date*100+ 01),"m")-1
tail(ds)
```

```
#>            date Mkt.RF   SMB    HML   RF
#> 1123 2020-01-31  -0.11 -3.11  -6.27 0.13
#> 1124 2020-02-29  -8.13  0.96  -4.01 0.12
#> 1125 2020-03-31 -13.39 -5.16 -14.12 0.12
#> 1126 2020-04-30  13.65  2.78  -1.27 0.00
#> 1127 2020-05-31   5.58  2.47  -4.95 0.01
#> 1128 2020-06-30   2.45  2.56  -2.03 0.01
```

We merge the risk factors data with the portfolios time series and regress the excess returns of each of the 25 portfolios on the three factors. To do so, we leverage some functional programming using the `lapply` base function along with the `dlply` function from the **plyr** library as follows:

```
PORT_RET_RA <- merge(PORT_RET,ds)
PORT_RET_RA$RAR <- PORT_RET_RA$VW_RET - PORT_RET_RA$RF

lm_list <- dlply(PORT_RET_RA,c("Group_ME","Group_BM"),
                 function(x)  lm( RAR ~ Mkt.RF +  SMB +   HML, data = x   )    )
rar_matrix <- lapply(lm_list, coef)
rar_matrix <- sapply(rar_matrix,function(x) x[[1]])
rar_matrix <- matrix(rar_matrix,5,5,byrow = TRUE)

rownames(rar_matrix) <- 1:5
colnames(rar_matrix) <- paste("BM",1:5,sep = "_")
rownames(rar_matrix)[1] <- "Small"
rownames(rar_matrix)[5] <- "Big"
colnames(rar_matrix)[1] <- "Low"
colnames(rar_matrix)[5] <- "High"
```

```
rar_matrix <- round(data.frame(rar_matrix*12),2)
rar_matrix
```

```
#>         Low  BM_2  BM_3  BM_4  High
#> Small -1.08  2.03  5.55  6.49  8.90
#> 2     -5.77 -0.33  0.78  2.47  1.01
#> 3     -6.51 -1.66  0.96 -0.15 -0.18
#> 4     -2.90 -1.41 -0.90  0.08 -0.10
#> Big    0.35  1.47 -0.47  0.32 -1.29
```

Note that after controlling for the three risk factors, the high-small portfolio still yields an annual return of 9%. A potential argument may be that such alpha is attributed to other risk factors that the three-factor model does not fully price. More recently, Fama-French suggest five factors model to better representation of systematic components (Fama and French, 2015).

Rendering Results

The above results are subjected to certain filtering, the time period, and the number of months each stock should include in the data. One major challenge is whether the securities in the analysis are tradable. For instance, it is common to consider stocks with a price larger than $5. The major issue with doing so is that such a filter could cause a forward-looking bias. By the time the decision maker allocates his/her portfolio, he/she cannot ascertain whether the stocks would be above or less than $5 in the future - this is less of an issue for large-cap stocks. Hence, to keep stocks that satisfy a minimum price level, it should be done on a recurring basis. Specifically, rather than dropping all observations in which the price is less than $5, one should consider only the stocks that were qualified by the time of the portfolio formation. The same issue applies to the window length for which the data is available and other filters one may be interested in controlling for.

The following is a generalized portfolio formation function that takes four arguments. The first two correspond to the minimum price and number of years that the stock should have to be considered investable in June each year. The other two arguments set the time period during which the researcher is interested in computed the mean returns. The function returns a list containing the raw and risk-adjusted returns of the 25 portfolios along with the average number of stocks within each portfolio.

```
FF3_anomaly <- function(min_price,year_keep = 1,year1 = 1965,year2 =  2019) {
  DT12_sub <- DT12
  keep_cusip_date <- unique(DT12_sub[,list(CUSIP,date_june,N_years,PRC_LAST)])
  keep_cusip_date <- keep_cusip_date[keep_cusip_date$N_years >= year_keep,]
  keep_cusip_date <- keep_cusip_date[keep_cusip_date$PRC_LAST >= min_price ,]

  keep_cusip_date <- unique(keep_cusip_date[,list(CUSIP,date_june)])
  DT12_sub <- merge(DT12_sub,keep_cusip_date)

  DT12_sub <- DT12_sub[,`:=` (Group_ME = ntile(ME,5)), by = list(date_june)]
  DT12_sub <- DT12_sub[,`:=` (Group_BM = ntile(BM_ratio,5)), by = list(date_june,Group_ME)]
  N_G <- DT12_sub[,.N, by = list(date,Group_ME,Group_BM) ]
  N_G <- (N_G[,mean(N), by = list(Group_ME,Group_BM) ])
  N_G <- N_G[order(N_G$Group_ME,N_G$Group_BM),]
  N_G <- matrix(N_G$V1,5,5,byrow = TRUE)
  rownames(N_G) <- 1:5
  colnames(N_G) <- paste("BM",1:5,sep = "_")
  rownames(N_G)[1] <- "Small"
  rownames(N_G)[5] <- "Big"
  colnames(N_G)[1] <- "Low"
  colnames(N_G)[5] <- "High"
  N_G <- round((N_G))

  PORT_RET <- DT12_sub[,list(VW_RET = sum(RET*ME/sum(ME))  ),
                       by = list(date,Group_ME,Group_BM)]
  PORT_RET <- PORT_RET[order(PORT_RET$date,PORT_RET$Group_BM,PORT_RET$Group_ME),]
  PORT_RET$VW_RET <- as.numeric(PORT_RET$VW_RET)*100

  PORT_RET_RA <- merge(PORT_RET,ds)
  PORT_RET_RA$RAR <- PORT_RET_RA$VW_RET - PORT_RET_RA$RF
```

```
  PORT_RET_RA_sub <- PORT_RET_RA
  PORT_RET_RA_sub <- PORT_RET_RA_sub[year(PORT_RET_RA_sub$date) >= year1,]
  PORT_RET_RA_sub <- PORT_RET_RA_sub[year(PORT_RET_RA_sub$date) <= year2 ,]

  ret_matrix <- PORT_RET_RA_sub[,lapply(.SD,mean,na.rm = TRUE),
                                by = list(Group_ME,Group_BM), .SDcols = "VW_RET"]
  ret_matrix <- ret_matrix[order(ret_matrix$Group_BM,ret_matrix$Group_ME),]
  ret_matrix <- matrix(as.numeric(ret_matrix$VW_RET),5)
  rownames(ret_matrix) <- 1:5
  colnames(ret_matrix) <- paste("BM",1:5,sep = "_")
  rownames(ret_matrix)[1] <- "Small"
  rownames(ret_matrix)[5] <- "Big"
  colnames(ret_matrix)[1] <- "Low"
  colnames(ret_matrix)[5] <- "High"
  ret_matrix <- round(data.frame(ret_matrix*12),2)


  lm_list <- dlply(PORT_RET_RA_sub,c("Group_ME","Group_BM"),
                   function(x)  lm( RAR ~ Mkt.RF +  SMB +   HML, data = x   )    )
  rar_matrix <- lapply(lm_list, coef)
  rar_matrix <- sapply(rar_matrix,function(x) x[[1]])
  rar_matrix <- matrix(rar_matrix,5,5,byrow = TRUE)
  rownames(rar_matrix) <- 1:5
  colnames(rar_matrix) <- paste("BM",1:5,sep = "_")
  rownames(rar_matrix)[1] <- "Small"
  rownames(rar_matrix)[5] <- "Big"
  colnames(rar_matrix)[1] <- "Low"
  colnames(rar_matrix)[5] <- "High"
  rar_matrix <- round(data.frame(rar_matrix*12),2)


  list(N_G,ret_matrix,rar_matrix)

}
```

Control for Minimum Price

Given the above function, we test the sensitivity of the size/value premiums by including stocks with pre-specified minimum price. In particular, we run this for a sequence of prices ranging between 0 and \$50. With the **parallel** library, we can easily perform parallel computing on four cores. It takes a few seconds to run the following commands[4]:

```
p_seq <- 0:50
list_port_price <- mclapply(p_seq, function(p) FF3_anomaly(p),mc.cores = 4)
```

Given the list `list_port_price`, we extract the result of interest. In this case, we focus on the raw returns. For instance, the first item of this list corresponds to the same results from Section 2.3.

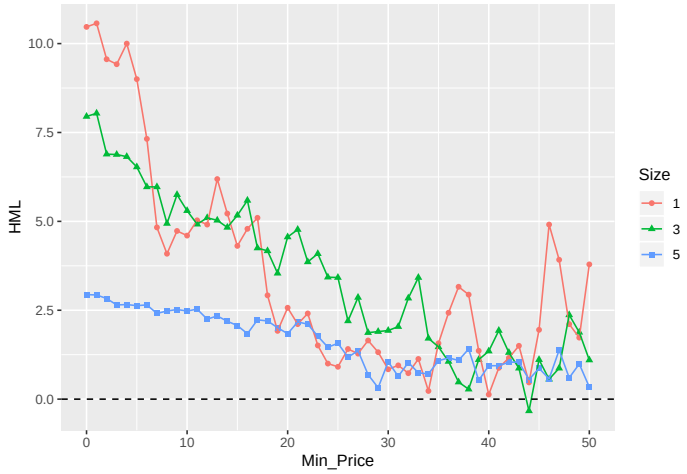
```
list_port_price[[1]][[2]]

#>         Low  BM_2  BM_3  BM_4  High
#> Small 12.93 14.73 18.05 20.14 23.40
#> 2      8.45 12.55 14.37 16.72 16.17
#> 3      8.04 11.79 14.30 14.24 15.99
#> 4      9.89 11.53 11.92 14.47 14.77
#> Big    9.84 11.70 10.41 13.13 12.78
```

On the other hand, if we consider \$5 minimum price, we see that the results are tentative depending on the level of entry:

```
list_port_price[[6]][[2]]

#>         Low  BM_2  BM_3  BM_4  High
#> Small  8.86 12.19 14.64 15.31 17.86
```

[4]Note that the `mclapply` can be easily leveraged on Linux machines.

Figure 4: This figure summarizes the results of the double-sorting portfolio based on the book-to-market (BM) ratio and size. Stocks are first sorted into five quintiles with respect to the BM ratio. Then, within each BM quintile, stocks are sorted based on size. The double-sorting results in 25 value-weighted portfolios. The y-axis corresponds to the value premium, which is the difference between the top and the bottom BM quintiles within a given size group. The x-axis controls for the stock's minimum price to be included within each one of the 25 sorted portfolios. The red, green, and blue lines correspond to the size groups of small-cap (1), medium-cap (2), and large-cap (5), respectively. Overall, the plot demonstrates the sensitivity of the value premium as a function of the minimum price.




```
#> 2     10.35 11.94 13.51 16.18 15.08
#> 3      8.54 12.39 13.52 14.41 15.07
#> 4     10.16 11.09 11.95 14.20 15.17
#> Big    9.79 11.66 10.62 12.95 12.42
```

To capture this for each matrix of raw returns, we compute the high-minus-low (HML), i.e., column 5 minus column 1 for each size level. This results in 5 HML premiums for each size level. We summarize the results in Figure 4.

```
RAR_list <- lapply(list_port_price,function(x) x[[2]])
RAR_list_Value <- t(sapply(RAR_list, function(x) x[,5] - x[,1]  ))
colnames(RAR_list_Value) <- rownames(RAR_list[[1]])
RAR_list_Size <- t(sapply(RAR_list, function(x) x[1,] - x[5,]  ))

ds.plot <- lapply(1:5, function(i) data.frame(HML =  RAR_list_Value[,i],
                                              Min_Price = p_seq, Size = i))
ds.plot <- Reduce(rbind,ds.plot)
ds.plot <- ds.plot[order(ds.plot$Size,ds.plot$Min_Price),]
ds.plot <- ds.plot[ds.plot$Size %in% c(1,3,5),]
ds.plot$Size <- as.factor(ds.plot$Size)
p <- ggplot(ds.plot,aes(x = Min_Price, y = HML, colour = Size, shape = Size))
p <- p + geom_point() + geom_line()
p <- p + geom_abline(intercept = 0, slope = 0, color="black",  linetype="dashed")
p
```

We observe that the HML is more evident for small-cap stocks. However, at the same time, we note that the premium declines as we increase the minimum price entry. One argument is the following. As we increase the minimum price, the size effect is mitigated and, hence, the value premium. On the other hand, this could also be attributed to whether investors can utilize such alphas for small-cap that tend to be less liquid.

We repeat the same plot for the size premium. In particular, we compute the small-minus-big (SMB), i.e., row 1 minus row 5 for each BM level. This results in 5 SMB premiums for each BM group. Similar to Figure 4, Figure 5 demonstrates the sensitivity of the results with respect to the minimum price:

```
ds.plot <- lapply(1:5, function(i) data.frame(SMB =  unlist(RAR_list_Size[,i]),
```

Figure 5: This figure summarizes the results of the double-sorting portfolio based on the book-to-market (BM) ratio and size. Stocks are first sorted into five quintiles with respect to the BM ratio. Then, within each BM quintile, stocks are sorted based on size. The double-sorting results in 25 value-weighted portfolios. The y-axis corresponds to the size premium, which is the difference between the top and the bottom size quintiles withing a given BM group. The x-axis controls for the stock's minimum price to be included within each one of the 25 sorted portfolios. The red, green, and blue lines correspond to different BM groups of low (1), medium (2), and high (5), respectively. Overall, the plot demonstrates the sensitivity of the size premium as a function of the minimum price.


```
                                        Min_Price = p_seq, BM = i))
ds.plot <- Reduce(rbind,ds.plot)
ds.plot <- ds.plot[order(ds.plot$BM,ds.plot$Min_Price),]
ds.plot <- ds.plot[ds.plot$BM %in% c(1,3,5),]
ds.plot$BM <- as.factor(ds.plot$BM)
p <- ggplot(ds.plot,aes(x = Min_Price, y = SMB, colour = BM, shape = BM))
p <- p + geom_point() + geom_line()
p <- p + geom_abline(intercept = 0, slope = 0, color="black",  linetype="dashed")
p
```

Consistent with the case for the HML premium, the SMB premium seems to shrink as we increase the minimum price entry. While different forces could be attributed to this, it is of great relevance for researchers/investors to understand the mechanisms behind which. Interested readers may find the work by Li et al. (2014) very relevant concerning the impact of liquidity and limits of arbitrage when it comes to the inclusion/exclusion of small-cap stocks.

Additional Results

We conduct one additional test related to the capital asset pricing model (CAPM) by Sharpe (1964) and Lintner (1965). The following analysis depends only on the CRSP database rather than the merged CRSP-COMPUSTAT data. Hence, the code is executed based on the `DT` data object only.

The CAPM postulates a positive linear relationship between the market beta (systematic risk) and the asset's mean return. Empirically, a large body of research shows that the relationship is flatter than the one predicted by the model (Jensen et al., 1972) or even negative (Frazzini and Pedersen, 2014). Consistent with the literature (Bali et al., 2016), we sort stocks into 10 portfolios based on their monthly beta. In particular, we use M months to estimate the market beta on a rolling window using the **rollRegres** (Christoffersen, 2019) library. We set M to be either 36, 60, or 120 months. For each sample size M, we estimate the market beta on a rolling basis. At the end of month t, we sort stocks into 10 groups based on their market beta and compute the value-weighted return over the next month at $t+1$. For the next month, we repeat the same procedure until the last month of our data sample.

```
BETA <- DT[,c("date","CUSIP","RET")]
BETA <- merge(BETA,ds[,c("date","Mkt.RF","RF")], by = "date")
BETA$E_RET <- (BETA$RET - BETA$RF/100)*100
```

```
BETA$RF <- NULL
# keep stocks with at least M months
BETA[, count := (.N), by = CUSIP]
BETA <- BETA[order(BETA$CUSIP,BETA$date),]
BETA <- BETA[BETA$count   > est_window,]

BETA <- BETA %>%
  group_by(CUSIP) %>%
  do(.,mutate(.,Beta = roll_regres(E_RET ~ Mkt.RF,data = .,est_window)$coef[,2]))

BETA <- data.table(BETA)
BETA <- na.omit(BETA[,list(date,CUSIP,Beta)])

DT_capm <- merge(DT,BETA, by = c("CUSIP","date"), all = F)
DT_capm <- DT_capm[order(DT_capm$CUSIP,DT_capm$date),]
DT_capm <- DT_capm[,`:=` (Group_Beta = ntile(Beta,10)), by = list(date)]
```

Note that we execute the `roll_regres` command using the `do` command in the tidy environment. This approach is the fastest solution to execute a rolling regression for panel data to the best of our knowledge. For instance, when $M = 120$, the `DT_capm` data object contains 1,287,611 observations with 9,149 unique securities. The execution takes about 10 seconds. Given the data object `DT_capm`, we compute the next month's portfolio value-weighted return as follows:

```
PORT_RET <- DT_capm[,list(VW_RET = 12*100*sum(RET_1*MKTCAP/sum(MKTCAP),na.rm = T),
                          VW_BETA = sum(Beta*MKTCAP/sum(MKTCAP),na.rm = T)),
                    by = list(date,Group_Beta)]
PORT_RET <- PORT_RET[order(PORT_RET$Group_Beta,PORT_RET$date),]
```

Finally, we summarize the results using **ggplot2** in Figure 6. Note that the dashed line denotes the relationship implied by the CAPM. Figure 6 is consistent with Figure 2 from Fama and French (2004), where the slope (intercept) denotes the average annual market excess (risk-free) return. We compute the slope and intercept of the CAPM line based on the sub-sample of the Fama-French data set `ds` that corresponds to the same dates of the portfolio returns.

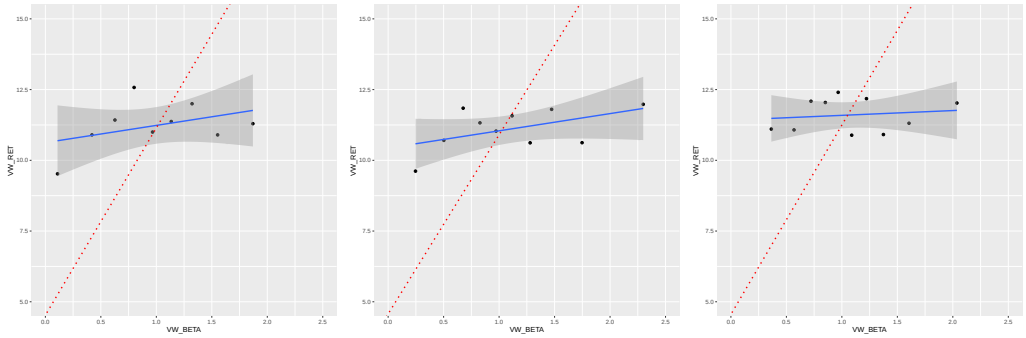
```
CAPM_result <- PORT_RET[,lapply(.SD,mean,na.rm = T),
                        by = list(Group_Beta),
                        .SDcols = c("EW_RET","VW_RET","EW_BETA","VW_BETA")]
CAPM_result <- CAPM_result[order(CAPM_result$Group_Beta),]
CAPM_result

ds_sub <- ds[ds$date %in% PORT_RET$date,]

p2 <- ggplot(CAPM_result, aes(VW_BETA, VW_RET)) +
  geom_point()
p2 <- p2 + geom_smooth(method = "lm")
p2 <- p2 +ylim(c(5,15)) + xlim(c(0,2.5))
p2 <- p2 + geom_abline(intercept = mean(ds_sub$RF)*12,
                       slope = mean(ds_sub$Mkt.RF)*12,
                       color="red", linetype="dotted", size=1)
```

Consistent with Fama and French (2004), we note that the portfolio results denote a flatter line than the one suggested by the CAPM. At the same time, we note that the results are noisy and sensitive to the sample size choice. In unreported results, we find that the negative relationship suggested by Frazzini and Pedersen (2014) is only evident when the betas are estimated using daily returns and portfolios are equally weighted.

Concluding Remarks

This article provides a brief illustration of merging the CRSP (security prices/returns) data with the COMPUSTAT (financial) data. The illustration is conducted for portfolio formation using book

Figure 6: Capital Asset Pricing Model
The dots below report the average annualized monthly return (y-axis) versus market beta (x-axis) for the value-weighted portfolios. The solid line is smoothed using linear regression. The dashed line corresponds to the one predicted by the CAPM over the same sample period, where the slope (intercept) denotes the average annual market excess (risk-free) return. Each panel corresponds to a different estimation window needed to estimate the market beta and form beta portfolios on a rolling basis.



(a) 36 Months Window (b) 60 Months Window (c) 120 Months Window

data. The final result is combining monthly market data with annual accounting data. However, researchers interested in performing panel analysis or applying predictive models using machine learning may prefer merging the data altogether using the same frequency. We leave this for future research. Nonetheless, we hope this article would encourage further reproducible empirical asset pricing research while also helping bridge the gap between the data science community and the empirical finance literature.

Notes

An html vignette is found on https://rpubs.com/simaan84/CRSP_COMP. The Rmd source code can be retrieved using the link.

Bibliography

T. G. Bali, R. F. Engle, and S. Murray. *Empirical asset pricing: The cross section of stock returns*. John Wiley & Sons, 2016. [p428, 429, 440]

W. Beaver, M. McNichols, and R. Price. Delisting returns and their effect on accounting-based market anomalies. *Journal of Accounting and Economics*, 43(2-3):341–368, 2007. [p428]

A. Y. Chen and T. Zimmermann. Open source cross-sectional asset pricing. *Available at SSRN*, 2020. [p426]

B. Christoffersen. *rollRegres: Fast Rolling and Expanding Window Linear Regression*, 2019. URL https://CRAN.R-project.org/package=rollRegres. R package version 0.1.3. [p440]

COMPUSTAT. Compustat north america. URL http://datalib.edina.ac.uk/catalogue/compustat. [p426]

CRSP. Center for research in security prices. URL http://www.crsp.org. [p426]

M. Dowle and A. Srinivasan. *data.table: Extension of 'data.frame'*, 2019. URL https://CRAN.R-project.org/package=data.table. R package version 1.12.8. [p426]

E. F. Fama and K. R. French. Common risk factors in the returns on stocks and bonds. *Journal of Finance*, 1993. [p426, 430, 431]

E. F. Fama and K. R. French. The capital asset pricing model: Theory and evidence. *Journal of economic perspectives*, 18(3):25–46, 2004. [p441]

E. F. Fama and K. R. French. A five-factor asset pricing model. *Journal of financial economics*, 116(1): 1–22, 2015. [p437]

A. Frazzini and L. H. Pedersen. Betting against beta. *Journal of Financial Economics*, 111(1):1–25, 2014. [p440, 441]

G. Grolemund and H. Wickham. Dates and times made easy with lubridate. *Journal of Statistical Software*, 40(3):1–25, 2011. URL http://www.jstatsoft.org/v40/i03/. [p426]

C. R. Harvey, Y. Liu, and H. Zhu. ... and the cross-section of expected returns. *The Review of Financial Studies*, 29(1):5–68, 2016. [p426]

M. C. Jensen, F. Black, and M. S. Scholes. The capital asset pricing model: Some empirical tests. 1972. [p440]

X. Li, R. N. Sullivan, and L. Garcia-Feijóo. The limits to arbitrage and the low-volatility anomaly. *Financial Analysts Journal*, 70(1):52–63, 2014. [p426, 440]

J. Lintner. The valuation of risk assets and the selection of risky investments in stock portfolios and capital budgets. *Review of Economics and Statistics*, 47(1):13–37, 1965. [p440]

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2020. URL https://www.R-project.org/. [p426]

W. F. Sharpe. Capital asset prices: A theory of market equilibrium under conditions of risk. *The journal of finance*, 19(3):425–442, 1964. [p440]

H. Wickham. The split-apply-combine strategy for data analysis. *Journal of Statistical Software*, 40(1): 1–29, 2011. URL http://www.jstatsoft.org/v40/i01/. [p426]

H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. ISBN 978-3-319-24277-4. URL https://ggplot2.tidyverse.org. [p426]

H. Wickham, R. François, L. Henry, and K. Müller. *dplyr: A Grammar of Data Manipulation*, 2020. URL https://CRAN.R-project.org/package=dplyr. R package version 0.8.5. [p426]

Majeed Simaan
School of Business, Stevens Institute of Technology
1 Castle Point Terrace, Babbio Center, Hoboken, NJ 07030, USA.
msimaan@stevens.edu

distr6: R6 Object-Oriented Probability Distributions Interface in R

by Raphael Sonabend and Franz J. Király

Abstract **distr6** is an object-oriented (OO) probability distributions interface leveraging the extensibility and scalability of R6 and the speed and efficiency of **Rcpp**. Over 50 probability distributions are currently implemented in the package with 'core' methods, including density, distribution, and generating functions, and more 'exotic' ones, including hazards and distribution function anti-derivatives. In addition to simple distributions, **distr6** supports compositions such as truncation, mixtures, and product distributions. This paper presents the core functionality of the package and demonstrates examples for key use-cases. In addition, this paper provides a critical review of the object-oriented programming paradigms in R and describes some novel implementations for design patterns and core object-oriented features introduced by the package for supporting **distr6** components.

Introduction

Probability distributions are an essential part of data science, underpinning models, simulations, and inference. Hence, they are central to computational data science. With the advent of modern machine learning and AI, it has become increasingly common to adopt a conceptual model where distributions are considered objects in their own right, as opposed to primarily represented through distribution-defining functions (e.g., cdf, pdf) or random samples.

An important distinction to keep in mind is between random variables (that can be sampled from) and probability distributions. **distr6** is an interface for probability distributions and supports construction, manipulation, composition, and querying of parameterized simple and composite distributions. **distr6** is not an interface for random variables, and therefore, procedures such as sampling and inference are out of scope.

The conceptual model: probability distributions and random variables

We continue by explaining our conceptual model of probability distributions underpinning the design of **distr6** and delineate it from the common conceptualization of random variables. A full mathematical definition of the conceptual model is given in the next section. This section contains an intuitive introduction.

First, we invite the reader to recall some common mathematical objects and recognize that these are related but conceptually distinct:

- a random variable, distributed according to a certain distribution, e.g., $X \sim \text{Normal}(0, 1)$
- the cdf of that random variable X, usually denoted by F_X, a function $F_X : \mathbb{R} \to [0, 1]$
- the pdf of that random variable X, often denoted by f_X, a function $f_X : \mathbb{R} \to [0, \infty)$
- the distribution according to which X is distributed - often called 'the law of' X. This can be represented by multiple mathematical objects, such as the cdf F_X or the pdf f_X. We will call this distribution d. Note that d is not identical to either these representation functions.

Critically, we highlight that random variables and distributions are neither identical objects nor concepts. A random variable X *has* distribution d, and multiple random variables may be distributed according to d. Further, random variables are sampled from, while the distribution is only a description of probabilities for X. Thus, X and d are not identical objects. Figure 1 visually summarizes these differences.

As a possible logical consequence of the above, we adopt the conceptual model that distribution is an abstract object, which:

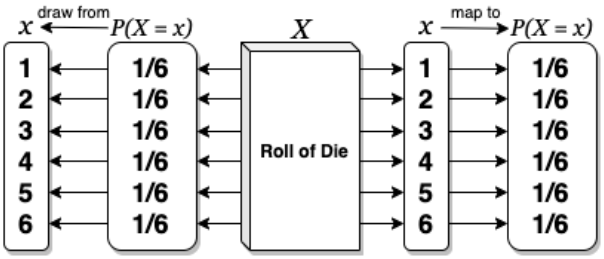
- Has multiple defining representations, for example, through cdf and possibly through pdf, but is not identical with any of these representations
- Possesses traits, such as being absolutely continuous over the Reals, and properties, such as skewness and symmetry.
- Can be used to define sampling laws of random variables but is not conceptually identical with a random variable.

Abstracting distributions as objects from multiple, non-identical, representations (random variables), introduces major consequences for the conceptual model:

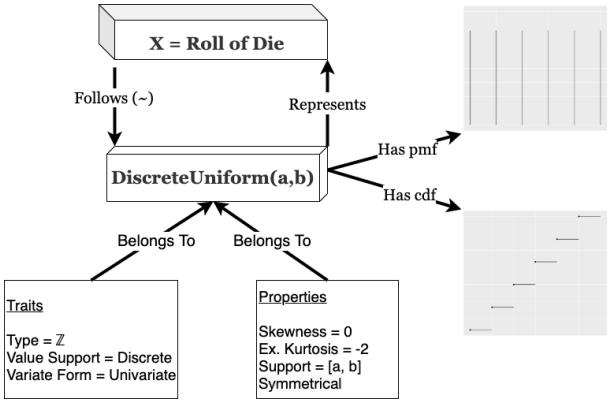
(i) It lends itself naturally to a class-object representation in the computer scientific sense of object-oriented programming. Abstract distributions become classes, concrete distributions are objects, and distribution defining functions are methods of these classes. Random variables are a separate type of object.

(ii) It strongly suggests adoption of mathematical conceptualization and notation which cleanly separates distributions from random variables and distribution defining functions - in contrast to common convention, where random variables or random sampling takes conceptual primacy above all.

(iii) It allows clean formulation of algorithmic manipulations involving distributions, especially higher-order constructs (truncation, huberization, etc.), as well as clean mathematical definitions.



(a) Discrete Uniform Random Variable



(b) Discrete Uniform Probability Distribution

Figure 1: Schematic representation of concepts "probability distribution" and "random variable". (a) Example of a random variable following a Discrete Uniform distribution. A random variable is an object in its own right, modeling the process of a random experiment. It *has* a probability distribution describing the nature of the experiment (in the case of uniform: the masses on the 6 outcomes) and can be sampled from, to obtain realizations. (b) Example of a probability distribution, the Discrete Uniform distribution. A probability distribution is an object in its own right and can serve as a template for random experiments, represented as random variables. Properties and traits such as value support, mean, variance, and functions such as pmf, cdf, are properties and traits of the probability distribution, not (direct) properties of random variables, even if random variables can follow a given distribution.

Distributions as software objects and mathematical objects

In **distr6**, distributions are first-class objects subject to an object-oriented class-object representation. For example, a discrete uniform distribution (fig. 1b) is a 'class' with traits such as type (Naturals) and variate form (univariate). With a given parameterization, this becomes an 'object' with properties including symmetry and support. An alternative definition to the conceptual model of distributions is now provided.

On the mathematical level, we again consider distributions as objects in their own right, not being identical with a cdf, pdf, or measure, but instead 'having' these as properties.

For a set $\mathcal{Y}$ (endowed with suitable topology), we define $\mathrm{Distr}(\mathcal{Y})$ as a set containing formal objects d which are in bijection to (but not identical with) probability measures over $\mathcal{Y}$. Elements of $\mathrm{Distr}(\mathcal{Y})$ are called distributions over $\mathcal{Y}$. We further define formal symbols which, in case of existence, denote 'aspects' that such elements have, in the following way: the symbol $d.F$, for example, denotes the cdf of

d, which is to be read as the 'F' of d, with F, in this case, to be read as a modifier to a standard symbol d, rather than a fixed, bound, or variable symbol. In this way, we can define:

(i) $d.F$ for the cdf of d. This typically exists if $\mathcal{Y} \subseteq \mathbb{R}^n$ for some n, in which case $d.F$ is a function of type $d.F : \mathbb{R}^n \to [0, 1]$.
(ii) $d.f$ for the pdf of d. This exists if $\mathcal{Y} \subseteq \mathbb{R}^n$, and the distribution d is absolutely continuous over $\mathcal{Y}$. In this case, $d.f$ is a function of type $d.f : \mathbb{R}^n \to [0, \infty)$.
(iii) $d.P$ for the probability measure that is in bijection with d. This is a function $d.P : \mathcal{F} \to [0, 1]$, where $\mathcal{F}$ is the set of measurable sub-sets of $\mathcal{Y}$.

We would like to point out that the above is indeed a full formal mathematical definition of our notion of distribution. While distributions, defined this way, are not identical with any of the conventional mathematical objects that define them (cdf, pdf, measures), they are conceptually, formally, and notationally well-defined. Similarly, the aspects ($d.F$, $d.f$, etc.) are also well-defined since they refer to one of the conventional mathematical objects which are well-specified in the dependence of the distribution (in case of existence).

This notation provides a more natural and clearer separation of distribution and random variables and allows us to talk about and denote concepts such as 'the cdf of any random variable following the distribution d' with ease ($d.F$), unlike classical notation that would see one define $X \sim d$ and then write F_X. Our notation more clearly follows the software implementation of distributions.

For example, in **distr6**, the code counterpart to defining a distribution d which is Gaussian with mean 1 and variance 2 is

```
> d <- Normal$new(1, 2)
```

The pdf and cdf of this Gaussian distribution evaluated at 2 are obtained in code as

```
> d$pdf(2)
> d$cdf(2)
```

which evaluates to 'numerics' that represent the real numbers $d.f(2)$ and $d.F(2)$.

The consideration of distributions as objects, and their conceptual distinction from random variables as objects, notably differs from conceptualization in R **stats**, which implements both distribution and random variable methods by the 'dpqr' functions. Whilst this may allow a very fast generation of probabilities and values, there is no support for querying and inspection of distributions as objects. By instead treating the dpqr functions as methods that belong to a distribution object, **distr6** encapsulates all the information in R **stats** as well as distribution properties, traits, and other important mathematical methods. The object orientation principle that defines the architecture of **distr6** is further discussed throughout this manuscript.

Treating distributions as objects is not unique to this package. Possibly the first instance of the object-oriented conceptualization is the **distr** (Ruckdeschel et al., 2006) family of packages. **distr6** was designed alongside the authors of **distr** in order to port some of their functionality from S4 to R6.

distr6 is the first such package to use the 'class' object-oriented paradigm **R6** (Chang, 2018), with other distribution related packages using S3 or S4. The choice of R6 over S3 and S4 is discussed in detail in section 2.5.1. This choice allows **distr6** to fully leverage the conceptual model and make use of core R6 functionality. As well as introducing fundamental object-oriented programming (OOP) principles such as abstract classes and tried and tested design patterns (Gamma et al., 1996) including decorators, wrappers, and compositors (see section 2.5.3).

Besides an overview of **distr6**'s novel approach to probability distributions in R, this paper also presents a formal comparison of the different OOP paradigms while detailing the use of design patterns relevant to the package.

Motivating example: Higher-order distribution constructs

The strength of the object-oriented approach, both on the algorithmic and mathematical side, lies in its ability to efficiently express higher-order constructs and operations: actions between distributions, resulting in new distributions. One such example is mixture distributions (also known as spliced distributions). In the **distr6** software interface, a MixtureDistribution is a higher-order distribution depending on two or more other distributions. For example, take a uniform mixture of two distributions distr1 and distr2:

```
> my_mixt <- MixtureDistribution$new(list(distr1, distr2))
```

Internally, the dependency of the constructs on the components is remembered so that `my_mixt` is not only evaluable for `cdf` (and other methods), but also carries a symbolic representation of its construction and definition history in terms of `distr1` and `distr2`.

On the mathematical side, the object-oriented formalism allows clean definitions of otherwise more obscure concepts. For example, the mixture distribution is now defined as follows:

For distributions $d_1, \ldots, d_m$ over $\mathbb{R}^n$ and weights $w_1, \ldots, w_m$, we define the mixture of $d_1, \ldots, d_m$ with weights $w_1, \ldots, w_m$ to be the unique distribution $\tilde{d}$ such that $\tilde{d}.F(x) = \sum_{i=1}^{m} w_i \cdot d_i.F(x)$ for any $x \in \mathbb{R}^n$. Note the added clarity by defining the mixture on the distribution d_i, i.e., a first-order concept in terms of distributions.

Related software

This section provides a review of other related software that implement probability distributions. This is focused on, but not limited to, software in R.

R stats, actuar, and extraDistr The core R programming language consists of packages for basic coding and maths as well as the **stats** package for statistical functions. **stats** contains 17 common probability distributions and four lesser-known distributions. Each distribution consists of (at most) four functions: `dX,pX,qX,rX` where `X` represents the distribution name. These correspond to the probability density/mass, cumulative distribution, quantile (inverse cumulative distribution), and simulation functions, respectively. Each is implemented as a separate function, written in C, with both inputs and outputs as numerics. The strength of these functions lies in their speed and efficiency. There is no quicker way to find, say, the pdf of a Normal distribution than to run the `dnorm` function from **stats**. However, this is the limit of the package in terms of probability distributions. As there is no designated distribution object, there is no way to query results from the distributions outside of the 'dpqr' functions.

Several R packages implement `dpqr` functions for extra probability distributions. Of particular note are the **extraDistr** (Wolodzko, 2019) and **actuar** (Dutang et al., 2008) packages that add over 60 distributions between them. Both of these packages are limited to `dpqr` functions and therefore have the same limits as R **stats**.

distr The **distr** package was the first package in R to implement an object-oriented interface for distributions, using the S4 object-oriented paradigm. **distr** tackles the two fundamental problems of **stats** by introducing distributions as objects that can be stored and queried. These objects include important statistical results, for example, the expectation, variance, and moment generating functions of a distribution. The **distr** family of packages includes a total of five packages for object-oriented distributions in R. **distr** has two weaknesses that were caused by using the S4 paradigm. Firstly, the package relies on inheritance, which means that large class trees exist for every object, and extensibility is therefore non-trivial. The second weakness is that S4 objects are not referred to by 'pointers' but instead copies. This means that a simple mixture of two distributions is just under 0.5Mb in size (relatively quite large).

distributions3 and distributional The **distributions3** package (Hayes and Moller-Trane, 2019) defines distributions as objects using the S3 paradigm. However, whilst **distributions3** treats probability distributions as S3 objects, it does not add any properties, traits, or methods and instead uses the objects solely for `dpqr` dispatch. In comparison to **distr**, the **distributions3** package provides fewer features for inspection or composition. More recently, **distributional** (O'Hara-Wild and Hayes, 2020) builds on the **distributions3** framework (common authors exist between the two) to focus on the vectorization of probability distributions coded as S3 objects. Similarly to **distr6**, the primary use-case of this package is for predictive modeling of distributions as objects.

mistr The **mistr** package (Sablica and Hornik, 2020) is another recent distributions package, which is also influenced by **distr**. The sole focus of **mistr** is to add a comprehensive and flexible framework for composite models and mixed distributions. Similarly, to the previous two packages, **mistr** implements distributions as S3 objects.

Distributions.jl Despite not being a package written in R, the Julia **Distributions.jl** (Lin et al., 2019) package provided inspiration for **distr6**. **Distributions.jl** implements distributions as objects with statistical properties including expectation, variance, moment generating and characteristic functions, and many more. This package uses multiple inheritance for 'valueSupport' (discrete/continuous) and 'variateForm' (univariate/multivariate/matrixvariate). Every distribution inherits from both of these, e.g., a distribution can be 'discrete-univariate', 'continuous-multivariate', 'continuous-matrixvariate', etc. The package provides a unified and user-friendly interface, which was a helpful starting point for **distr6**.

Design principles

distr6 was designed and built around the following principles.

D1) **Unified interface** The package is designed such that all distributions, no matter how complex, have an identical user-facing interface. This helps make the package easy to navigate and the documentation simple to read. Moreover, it minimizes any confusion resulting from using multiple distributions. A clear inheritance structure also allows wrappers and decorators to have the same methods as distributions, which means even complex composite distributions should be intuitive to use. Whether a user constructs a simple Uniform distribution or a mixture of 100 Normal distributions, the same methods and fields are seen in both objects.

D2) **Separation of core/exotic and numerical/analytic** Via abstraction and encapsulation, core statistical results (common methods such as mean and variance) are separated from 'exotic' ones (less common methods such as anti-derivatives and p-norms). Similarly, implemented distributions only contain analytic results; users can impute numerical results using decorators. This separation has several benefits, including: 1) for predictive modeling with/of distributions, numerical results can take longer to compute than analytical results, and the difference between precision of analytical and numerical results can be substantial in the context of automated modeling, separation allows these differences to be highlighted and controlled; 2) separating numerical results allows an expanded interface for users to fine-tune and set their own parameters for how numerical results are computed; 3) a less-technical user can guarantee the precision of results as they are unlikely to use numerical decorators; 4) a user has access to the most important distribution methods immediately after construction but is not overwhelmed by many 'exotic' methods that they may never use. Use of decorators and wrappers allows the user to manually expand the interface at any time. For example, a user can choose between an undecorated Binomial distribution, with common methods such as mean and variance, or they can decorate the distribution to additionally gain access to survival and hazard functions.

D3) **Inheritance without over-inheritance** The class structure stems from a series of a few abstract classes with concrete child classes, which allows for a sensible, but not over-complicated, inheritance structure. For example, all implemented distributions inherit from a single parent class, so common methods can be unified and only coded once; note there is no separation of distributions into 'continuous' and 'discrete' classes. By allowing the extension of classes by decorators and wrappers, and not solely inheritance, the interface is highly scalable and extensible. By 'scalability', we refer to the interface's ability to grow to a large scale without additional overheads. The decorator and wrapper patterns on top of the R6 paradigm allow an (theoretically) unlimited number of distributions, wrappers, and methods without computational difficulty. By 'extensibility', we refer to the ability to extend the interface. Again this is made possible by clean abstraction of distributions, wrappers, core methods, and extra methods in decorators. All decorators and wrappers in **distr6** stem from abstract classes, which in turn inherit from the `Distribution` super-class. In doing so, any method of expanding an object's interface in **distr6** (i.e., via decorators, wrappers, or inheritance) will automatically lead to an interface that inherits from the top-level class, maintaining the principle of a unified interface (D1).

D4) **Inspection and manipulation of multiple parameterizations** The design process identified that use of distributions in R **stats** is inflexible in that in the majority of cases, only one parameterization of each distribution is allowed. This can lead to isolating users who may be very familiar with one parameterization but completely unaware of another. For example, the use of the *precision* parameter in the Normal distribution is typically more common in Bayesian statistics, whereas using the *variance* or *standard deviation* parameters is more common in frequentist statistics. **distr6** allows the user to choose from multiple parameterizations for all distributions (where more than one parameterization is possible/known). Furthermore, querying and updating of any parameter in the distribution is allowed, even if it was not specified in construction (section 2.4). This allows for a flexible parameter interface that can be fully queried and modified at any time.

D5) **Flexible interfacing for technical and non-technical users** Throughout the design process, it was required that **distr6** be accessible to all R users. This was a challenge as R6 is a very different paradigm from S3 and S4. To reduce the learning curve, the interface is designed to be as user-friendly and flexible as possible. This includes: 1) a 'sensible default principle' such that all distributions have justified default values; 2) an 'inspection principle' with functions to list all distributions, wrappers, and decorators. As discussed in (D2), abstraction and encapsulation allow technical users to expand any distribution's interface to be as arbitrarily complex as they like, whilst maintaining a minimal representation by default. Where possible defaults are 'standard' distributions, i.e. with location 0 and scale 1, otherwise sensible defaults are identified as realistic scenarios, for example `Binomial(n = 10,p = 0.5)`.

D6) **Flexible OO paradigms** Following from (D5), we identified that R6 is still relatively new in R with only 314 out of 16,050 packages depending on it (as of July 2020). Therefore this was acknowledged and taken into account when building the package. R6 is also the first paradigm in R with the dollar-sign notation (though S4 uses '@' notation) and with a proper construction method. Whilst new users are advised to learn the basics of R6, S3 compatibility is available for all common methods via **R62S3** (Sonabend, 2019). Users can therefore decide on calling a method via dollar-sign notation or dispatch. The example below demonstrates 'piping' and S3. As the core package is built on R6, the thin-wrappers provided by **R62S3** do not compromise the above design principles.

```
> library(magrittr)
> N <- Normal$new(mean = 2)
> N %>%
+   setParameterValue(mean = 1) %>%
+   getParameterValue("mean")
[1] 1
> pdf(N, 1:4)
[1] 0.398942280 0.241970725 0.053990967 0.004431848
```

Overview to functionality and API

distr6 1.4.3 implements 56 probability distributions, including 11 probability kernels. Individual distributions are modeled via classes that inherit from a common interface, implemented in the abstract `Distribution` parent class. The `Distribution` class specifies the abstract distribution interface for parameter access, properties, traits, and methods, such as a distribution's pdf or cdf. The most important interface points are described in Section 2.4.1

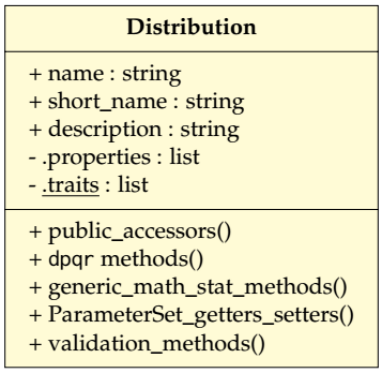


Figure 2: The Distribution class.

Concrete distributions, kernels, and wrappers are the grandchildren of `Distribution`, and children of one of the mid-layer abstract classes:

- `SDistribution`, which models abstract, generic distributions. Concrete distributions, such as `Normal`, which models the normal distribution, inherit from `SDistribution`.
- `Kernel`, which models probability kernels, such as `Triangular` and `Epanechnikov`. Probability kernels are absolutely continuous distributions over the Reals, with assumed mean 0 and variance 1.
- `DistributionWrapper`, which is an abstract parent for higher-order operations on distributions, including compositions, that is, operations that create distributions from other distributions, such as truncation or mixture.

Method	Description/Definition
`pdf/cdf/quantile/rand`	dpqr functions.
`mean`	$d.\mu = \mathbb{E}[X]$
`variance`	$d.\sigma^2 = \mathbb{E}[(X - d.\mu)^2]$
`traits`	List including value support (discrete/continuous/mixed); variate form (uni-/multi-/matrixvariate); type (mathematical domain).
`properties`	List including skewness ($\mathbb{E}[((X - d.\mu)/d.\sigma)^3]$) and symmetry (boolean).
`get/setParameterValue`	Getters and setters for parameter values.
`parameters`	Returns the internal parameterization set.
`print/summary`	Representation functions, summary includes distribution properties and traits.

Table 1: Common methods available to all classes inheriting from `Distribution`. Columns are method names and either, mathematical definition or a description of the method. For mean and variance, $d.\mu$, $d.\sigma$, and $d.\sigma^2$ represent the methods mean, standard deviation, and variance associated with distribution d. The symbol X denotes a random variable with distribution d. Horizontal lines separate mathematical, property, parameter, and representation methods.

- `DistributionDecorator`, whose purpose is supplementing methods to distributions in the form of a decorator design pattern. This includes methods such as integrated cdf or squared integrals of distribution defining functions.

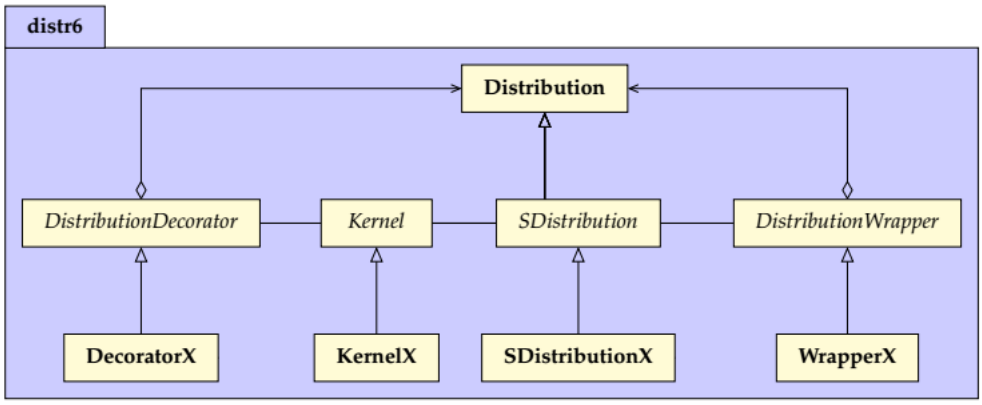


Figure 3: Simplified **distr6** class structure.

Figure 3 visualizes the key class structure of **distr6**, including the concrete `Distribution` parent class, from which all other classes in the package inherit from (with the exception of the `ParameterSet`). These abstract classes allow simple extensibility for concrete sub-classes.

The Distribution interface

The base, or top-level, class in **distr6** is the `Distribution` class. Its primary function is to act as a parent class for the implemented probability distributions and higher-order compositions. It is also utilized for the creation of custom distributions. By design, any distribution already implemented in **distr6** will have the same interface as a user-specified custom distribution, ensuring (D1) is upheld. The most important methods for a distribution are shown in Table 1 alongside their meaning and definitions (mathematical if possible). The two use-cases for the `Distribution` class are discussed separately.

Distribution for inheritance It is anticipated that the majority of **distr6** users will be using the package for the implemented distributions and kernels. With this in mind, the `Distribution` class defines all variables and methods common to all child classes. The most important of these are the common analytical expressions and the `dpqr` public methods. Every concrete implemented distribution/kernel has identical public `dpqr` methods that internally call private `dpqr` methods. This

accounts for inconsistencies occurring from packages returning functions in different formats and handling errors differently, a problem most prominent in multivariate distributions. Another example is the handling of non-integer values for discrete distributions. In some packages, this returns 0, or the value is rounded down, or an error is returned. The `dpqr` functions for all distributions have unified validation checks and return types (`numeric` or `data.table`). In line with base R and other distribution packages, **distr6** implements a single `pdf` function to cover both probability mass and probability density functions.

```
> Normal$new()$pdf(1:2)
[1] 0.24197072 0.05399097
> Binomial$new()$cdf(1:2, lower.tail = FALSE, log.p = TRUE, simplify = FALSE)
         Binom
1: -0.01080030
2: -0.05623972
```

A key design principle in the package is the separation of analytical and numerical results (D2), which is ensured by only including analytical results in implemented distributions. Missing methods in a distribution, therefore, signify that no closed-form expression for the method is available. However, all can be numerically estimated with the `CoreStatistics` decorator (see section 2.4.2). Ideally, all distributions will include analytical methods for the following: probability density/mass function (`pdf`), cumulative distribution function (`cdf`), inverse cumulative distribution function/quantile function (`quantile`), simulation function (`rand`), mean, variance, skewness, (excess) kurtosis, and entropy of the distribution (`mean`,`variance`,`skewness`,`kurtosis`,`entropy`), as well as the moment generating function (`mgf`), characteristic function (`cf`), and probability generating function (`pgf`). Speed is currently a limitation in **distr6**, but the use of **Rcpp** (Eddelbuettel and Francois, 2011) in all `dpqr` functions helps mitigate against this.

The fourth design principle of **distr6** ensures that multiple parameterizations of a given distribution can be both provided and inspected at all times. For example, the Normal distribution can be parameterized in terms of variance, standard deviation, or precision. Any of which can be called in construction with other parameters updated accordingly. If conflicting parameterizations are provided, then an error is returned. By example,

```
# set precision, others updated automatically
> Normal$new(prec = 4)
Norm(mean = 0, var = 0.25, sd = 0.5, prec = 4)
# try and set both precision and variance, results in error
> Normal$new(var = 1, prec = 2)
Error in FUN(X[[i]], ...) :
  Conflicting parametrisations detected. Only one of {var, sd, prec} should be given.
```

The same principle is used for parameter setting. The methods `getParameterValue` and `setParameterValue` are utilized for getting and setting parameter values, respectively. The former takes a single argument, the parameter name, and the second a named list of arguments corresponding to the parameter name and the value to set. The example below demonstrates this for a Gamma distribution. Here, the distribution is constructed, the shape parameter is queried, both shape and rate parameters are updated, and the latter queried, finally, the scale parameter is set, which auto-updates the rate parameter.

```
> G <- Gamma$new(shape = 1, rate = 1)
> G$getParameterValue("shape")
[1] 1
> G$setParameterValue(shape = 1, rate = 2)
> G$getParameterValue("rate")
[1] 2
> G$setParameterValue(scale = 2)
> G$getParameterValue("rate")
[1] 0.5
```

Distribution and parameter domains and types are represented by mathematical sets implemented in **set6** (Sonabend and Kiraly, 2020). This allows for a clear representation of infinite sets and, most importantly, for internal containedness checks. For example, all public `dpqr` methods first call the `$contains` method in their respective `type` and return an error if any points are outside the distribution's domain. As **set6** uses **Rcpp** for this method, these come at minimal cost to speed.

```
> B <- Binomial$new()
> B$pdf(-1)
Error in B$pdf(-1) :
  Not all points in {-1} lie in the distribution domain (N0).
```

These domains and types are returned along with other important properties and traits in a call to `properties` and `traits`, respectively. This is demonstrated below for the Arcsine distribution.

```
> A <- Arcsine$new()
> A$properties
$support
[0,1]


$symmetry
[1] "symmetric"

> A$traits
$valueSupport
[1] "continuous"

$variateForm
[1] "univariate"

$type
R
```

Extending distr6 with custom distributions Users of **distr6** can create temporary custom distributions using the constructor of the `Distribution` class directly. Permanent extensions, e.g., as part of an R package, should create a new concrete distribution as a child of the `SDistribution` class.

The `Distribution` constructor is given by

```
Distribution$new(name = NULL, short_name = NULL, type = NULL, support = NULL,
+ symmetric = FALSE, pdf = NULL, cdf = NULL, quantile = NULL, rand = NULL,
+ parameters = NULL, decorators = NULL, valueSupport = NULL, variateForm = NULL,
+ description = NULL)
```

The `name` and `short_name` arguments are identification for the custom distribution used for printing. `type` is a trait corresponding to scientific type (e.g., Reals, Integers,...), and `support` is the property of the distribution support. Distribution parameters are passed as a `ParameterSet` object. This defines each parameter in the distribution, including the parameter default value and support. The `pdf/cdf/quantile/rand` arguments define the corresponding methods and are passed to the private `.pdf/.cdf/.quantile/.rand` methods. As above, the public methods are already defined and ensure consistency in each function. At a minimum, users have to supply the distribution `name`, `type`, and either `pdf` or `cdf`. All other information can be numerically estimated with decorators (see section 2.4.2).

```
> d <- Distribution$new(name = "Custom Distribution", type = Integers$new(),
+                       support = Set$new(1:10),
+                       pdf = function(x) rep(1/10, length(x)))
> d$pdf(1:3)
[1] 0.1 0.1 0.1
```

DistributionDecorator

Decorators add functionality to classes in object-oriented programming. These are not natively implemented in R6, and this novel implementation is therefore discussed further in section 2.5.3. Decorators in **distr6** are only 'allowed' if they have at least three methods and cover a clear use case. This prevents too many decorators from bloating the interface. However, by their nature, they are lightweight classes that will only increase the methods in a distribution if explicitly requested by a user. Decorators can be applied to a distribution in one of three ways.

In construction:

```
> N <- Normal$new(decorators = c("CoreStatistics", "ExoticStatistics"))
```

Using the `decorate()` function:

```
> N <- Normal$new()
> decorate(N, c("CoreStatistics", "ExoticStatistics"))
```

Using the `$decorate` method inherited from the `DistributionDecorator` super-class:

```
> N <- Normal$new()
> ExoticStatistics$new()$decorate(N)
```

The first option is the quickest if decorators are required immediately. The second is the most efficient once a distribution is already constructed. The third is the closest method to true OOP but does not allow adding multiple decorators simultaneously.

Three decorators are currently implemented in **distr6**. These are briefly described.

CoreStatistics This decorator imputes numerical functions for common statistical results that could be considered core to a distribution, e.g., the mean or variance. The decorator additionally adds generalized expectation (`genExp`) and moments (`kthmoment`) functions, which allow numerical results for functions of the form $\mathbb{E}[f(X)]$ and for crude/raw/central K moments. The example below demonstrates how the `decorate` function exposes methods from the `CoreStatistics` decorator to the Normal distribution object.

```
> n <- Normal$new(mean = 2, var = 4)
> n$kthmoment(3, type = "raw")
Error: attempt to apply non-function
> decorate(n, CoreStatistics)
> n$kthmoment(3, type = "raw")
[1] 32
```

ExoticStatistics This decorator adds more 'exotic' methods to distributions, i.e., those that are unlikely to be called by the majority of users. For example, this includes methods for the p-norm of survival and cdf functions, as well as anti-derivatives for these functions. Where possible, analytic results are exploited. For example, this decorator can implement the survival function in one of two ways: either as i) 1 minus the distribution cdf if an analytic expression for the cdf is available, or ii) via numerical integration of the distribution.

FunctionImputation This decorator imputes numerical expressions for the `dpqr` methods. This is the most useful for custom distributions in which only the `pdf` or `cdf` is provided. Numerical imputation is implemented via **Rcpp**.

Composite distributions

Composite distributions - that is, distributions created from other distributions - are common in advanced usage. Examples for composites are truncation, mixture, or transformation of domain. In **distr6**, a number of such composites are supported. Implementation-wise, this uses the wrapper OOP pattern, which is not native to R6 but part of our extensions to R6, discussed in section 2.5.3.

As discussed above, wrapped distributions inherit from `Distribution` thus have an identical interface to any child of `SDistribution`, with the following minor differences:

- The `wrappedModels` method provides a unified interface to access any component distribution.
- Parameters are still accessed via the same method but stored in a `ParameterSetCollection` object instead of a `ParameterSet`, thus allowing efficient representation of composite and nested parameter sets.

The composition can be iterated and nested any number of times, consider the following example where a mixture distribution is created from two distributions that are in turn composites - a truncated Student T, and a Huberized exponential - note the parameter inspection and automatic prefixing of distribution 'short names' to the parameters for identification.

```
> M <- MixtureDistribution$new(list(
+   truncate(StudentT$new(), lower = -1, upper = 1),
+   huberize(Exponential$new(), upper = 4)
+ ))
> M$parameters()
```

```
                id   value           support                    description
1:        mix_T_df       1                R+           Degrees of Freedom
2: mix_trunc_lower      -1      R U {-Inf, +Inf}    Lower limit of truncation
3: mix_trunc_upper       1       R U {-Inf +Inf}    Upper limit of truncation
4:    mix_Exp_rate       1                R+                Arrival Rate
5:   mix_Exp_scale       1                R+                       Scale
6:   mix_hub_lower       0      R U {-Inf, +Inf}  Lower limit of huberization
7:   mix_hub_upper       4      R U {-Inf, +Inf}  Upper limit of huberization
8:     mix_weights uniform     {uniform} U [0,1]             Mixture weights
```

Implemented compositors We summarize some important implemented compositors (tables 2 and 3) to illustrate the way composition is handled and implemented.

Class	Parameters	Type of d	Components of d
`TruncatedDistribution`	$a, b \in \mathbb{R}$	$\mathbb{R}$	d', type $\mathbb{R}$
`HuberizedDistribution`	$a, b \in \mathbb{R}$	$\mathbb{R}$, mixed	d', type $\mathbb{R}$
`MixtureDistribution`	$w_i \in \mathbb{R}, \sum_{i=1}^n w_i = 1$	$\mathbb{R}^n$	d_i', type $\mathbb{R}^n$
`ProductDistribution`	-	$\mathbb{R}^N, N = \sum_{i=1}^n n_i$	d_i' type $\mathbb{R}^{n_i}$

Table 2: Examples of common compositors implemented in **distr6** - parameters and type. Column 2 (parameters) lists the parameters that the composite has. Column 3 (type) states the type of the resultant distribution d that is created when the class is constructed; this states the formal domain. Column 4 (components) states the number, names, and assumptions on the components, if any.

Class	$d.F(x)$	$d.f(x)$
`TruncatedDistribution`	$\frac{d'.F(x) - d'.F(a)}{d'.F(b) - d'.F(a)}$	$\frac{d'.f(x)}{d'.P([a,b])}$
`HuberizedDistribution`	$d'.F(x) + \mathbb{1}[x = b] \cdot d'.P(b)$	(no pdf, since mixed)
`MixtureDistribution`	$\sum_{i=1}^N w_i \cdot d_i'.F(x)$	$\sum_{i=1}^N w_i \cdot d_i'.f(x)$ (if exists)
`ProductDistribution`	$\prod_{i=1}^N d_i'.F(x)$	$\prod_{i=1}^N d_i'.f(x)$ (if exists)

Table 3: Common compositors implemented in **distr6** with mathematical definitions. Column 2 defines the resultant cdf, $d.F$, in terms of the component cdf, as implemented in the `cdf` method of the compositor in the same row. Column 3 defines the resultant pdf, $d.f$, in terms of the component pdf, as implemented in the `pdf` method of the compositor in the same row. Truncation (row 1) is currently implemented for the left-open interval $(a, b]$ only. For Huberization (row 2), the resultant distribution is in general, not absolutely, continuous and hence the pdf does not exist. d' resp. d_i' are the component distributions, as defined in table 2.

Example code to obtain a truncated or Huberized distribution is below. Here, we construct a truncated normal with truncation parameters -1 and 1, and a Huberized Binomial with bounding parameters 2 and 5.

```
> TN <- truncate(Normal$new(), lower = -1, upper = 1)
> TN$cdf(-2:2)
[1] 0.0 0.0 0.5 1.0 1.0
> class(TN)
[1] "TruncatedDistribution" "DistributionWrapper" "Distribution" "R6"

> HB <- huberize(Binomial$new(), lower = 2, upper = 5)
> HB$cdf(1:6)
[1] 0.0000000 0.0546875 0.1718750 0.3769531 1.0000000 1.0000000
> HB$median()
[1] 5
```

Vectorization of distributions A special feature of **distr6** is that it allows vectorization of distributions, i.e., vectorized representation of multiple distributions in an array-like structure. This is primarily done for computational efficiency with the general best R practice of vectorization. Vectorization of **distr6** distributions is implemented via the `VectorDistribution`, which is logically treated as a compositor.

Mathematically, a `VectorDistribution` is simply a vector of component distributions $d_1, \ldots, d_N$ that allows vectorized evaluation. Two kinds of vectorized evaluation are supported paired and product-wise vectorization, which we illustrate in the case of cdfs.

- Paired vectorized evaluation of the cdfs $d_1.F, \ldots, d_N.F$ at numbers $x_1, \ldots, x_N$, yields a real vector $(d_1.F(x_1), \ldots, d_N.F(x_N)$ via the `cdf` method.
- Product vectorized evaluation of the cdfs $d_1.F, \ldots, d_N.F$ at numbers $x_1, \ldots, x_M$, yields a real $(N \times M)$ matrix, with (i, j)-th entry $d_i.F(x_j)$.

In practical terms, paired evaluation is the evaluation of N distributions at N points (which may be unique or different). So, by example, for three distributions d_1, d_2, d_3, paired evaluation of their cdfs at $(x_1, x_2, x_3) = (4, 5, 6)$ respectively results in $(d_1.F(x_1), d_2.F(x_2), d_3.F(x_3)) = (d_1.F(4), d_2.F(5), d_3.F(6))$. In **distr6**:

```
> V <- VectorDistribution$new(distribution = "Normal", params = data.frame(mean = 1:3))
> V$cdf(4, 5, 6)
       Norm1     Norm2     Norm3
1: 0.9986501 0.9986501 0.9986501
```

In contrast, product-wise evaluation evaluates N distributions at the same M points. Product-wise evaluation of the cdfs of d_1, d_2, d_3 at $(x_1, x_2, x_3) = (4, 5, 6)$ results in

$$\begin{pmatrix} d_1.F(x_1) & d_1.F(x_2) & d_1.F(x_3) \\ d_2.F(x_1) & d_2.F(x_2) & d_2.F(x_3) \\ d_3.F(x_1) & d_3.F(x_2) & d_3.F(x_3) \end{pmatrix} = \begin{pmatrix} d_1.F(4) & d_1.F(5) & d_1.F(6) \\ d_2.F(4) & d_2.F(5) & d_2.F(6) \\ d_3.F(4) & d_3.F(5) & d_3.F(6) \end{pmatrix}$$

In **distr6**:

```
> V <- VectorDistribution$new(distribution = "Normal", params = data.frame(mean = 1:3))
> V$cdf(4:6)
       Norm1     Norm2     Norm3
1: 0.9986501 0.9772499 0.8413447
2: 0.9999683 0.9986501 0.9772499
3: 0.9999997 0.9999683 0.9986501
```

The `VectorDistribution` wrapper allows for efficient vectorization across *both* the distributions *and* points to evaluate, which we believe is a feature unique to **distr6** among distribution frameworks in R. By combing product and paired modes, users can evaluate any distribution in the vector at any point. In the following example, Normal(1, 1) is evaluated at (1,2), and Normal(2, 1) is evaluated at (3,4):

```
> V <- VectorDistribution$new(distribution = "Normal", params = data.frame(mean = 1:2))
> V$pdf(1:2, 3:4)
       Norm1      Norm2
1: 0.3989423 0.24197072
2: 0.2419707 0.05399097
```

Further, common composites such as `ProductDistribution` and `MixtureDistribution` inherit from `VectorDistribution`, allowing for efficient vector dispatch of pdf and cdf methods. Inheriting from `VectorDistribution` results in identical constructors and methods. Thus, a minor caveat is that users could evaluate a product or mixture at different points for each distribution, which is not a usual use case in practice.

Two different choices of constructors are provided. The first '`distlist`' constructor passes distribution *objects* into the constructor, whereas the second passes a reference to the distribution *class* along with the parameterizations. Therefore, the first allows different types of distributions but is vastly slow as the various methods have to be calculated individually, whereas the second only allows a single class of distribution at a time but is much quicker in evaluation. In the example below, the mixture uses the second constructor, and the product uses the first.

```
> M <- MixtureDistribution$new(distribution = "Degenerate",
+                              params = data.frame(mean = 1:10))
> M$cdf(1:5)
[1] 0.1 0.2 0.3 0.4 0.5
> class(M)
[1] "MixtureDistribution" "VectorDistribution"  "DistributionWrapper" "Distribution"
[5] "R6"

> P <- ProductDistribution$new(list(Normal$new(), Exponential$new(), Gamma$new()))
> P$cdf(1:5)
[1] 0.3361815 0.7306360 0.9016858 0.9636737 0.9865692
```

Design patterns and object-oriented programming

This paper has so far discussed the API and functionality in **distr6**. This section discusses object-oriented programming (OOP). Firstly, a brief introduction to OOP and OOP in R and then the package's contributions to the field.

S3, S4, and R6

R has four major paradigms for object-oriented programming: S3, S4, reference classes (R5), and most recently, R6. S3 and S4 are known as functional object-oriented programming (FOOP) paradigms, whereas R5 and R6 move towards class object-oriented programming (COOP) paradigms (R6) (Chambers, 2014). One of the main differences (from a user perspective) is that methods in COOP are associated with a class, whereas in FOOP, methods are associated with generic functions. In the first case, methods are called by first specifying the object, and in the second, a dispatch registry is utilized to find the correct method to associate with a given object.

S3 introduces objects as named structures, which in other languages are often referred to as 'typed lists'. These can hold objects of any type and can include meta-information about the object itself. S3 is the dominant paradigm in R for its flexibility, speed, and efficiency. As such, it is embedded deep in the infrastructure of R, and single dispatch is behind a vast majority of the base functionality, which is a key part of making R easily readable. S3 is a FOOP paradigm in which functions are part of a dispatch system and consist of a generic function that is external to any object and a specific method registered to a 'class'. However, the term 'class' is slightly misleading as no formal class structure exists (and by consequence, no formal construction or inheritance) and as such, S3 is not a formal OOP language[1].

S4 formalizes S3 by introducing: class-object separation, a clear notion of construction, and multiple inheritances (Chambers, 2014). S4 has more syntax for the user to learn and a few more steps in class and method definitions. As a result, S4 syntax is not overly user-friendly, and S3 is used vastly more than S4 (Chambers, 2014).

There is a big jump from S3 and S4 to R6 as they transition from functional- to class-object-oriented programming. This means new notation, semantics, syntax, and conventions. The key changes are: 1) introducing methods and fields that are associated with classes not, functions; 2) mutable objects with copy-on-modify semantics; and 3) new dollar-sign notation. In the first case, this means that when a class is defined, all the methods are defined as existing within the class, and these can be accessed at any time after construction. Methods are further split into *public* and *private*, as well as *active bindings*, which incorporate the abstraction part of OOP. The mutability of objects and change to copy-on-modify means that to create an independent copy of an object, the new method `clone(deep = TRUE)` has to be used, which would be familiar to users who know more classical OOP but very different from most R users. Finally, methods are accessed via the dollar-sign, and not by calling a function on an object.

Below, the three paradigms are contrasted semantically with a toy example to create a 'duck' class with a method 'quack'.

S3

```
> quack <- function(x) UseMethod("quack", x)
> duck <- function(name) return(structure(list(name = name), class = "duck"))
> quack.duck <- function(x) cat(x$name, "QUACK!")
```

[1] http://adv-r.had.co.nz/OO-essentials.html

```
> quack(duck("Arthur"))
Arthur QUACK!
```

S4

```
> setClass("duck", slots = c(name = "character"))
> setGeneric("quack", function(x) {
+  standardGeneric("quack")
+  })
> setGeneric("duck", function(name) {
+   standardGeneric("duck")
+  })
> setMethod("duck", signature(name = "character"),
+             definition = function(name){
+              new("duck", name = name)
+              })
> setMethod("quack",
+             definition = function(x) {
+             cat(x@name, "QUACK!")
+             })
> quack(duck("Ford"))
Ford QUACK!
```

R6

```
> duck <- R6::R6Class("duck", public = list(
+   initialize = function(name) private$.name = name,
+   quack = function() cat(private$.name, "QUACK!")),
+   private = list(.name = character(0)))
> duck$new("Zaphod")$quack()
Zaphod QUACK!
```

The example clearly highlights the extra code introduced by S4 and the difference between the S3 dispatch and R6 method system.

Comparing the paradigms There is no doubt that R6 is the furthest paradigm from conventional R usage, and as such, there is a steep learning curve for the majority of R users. However, R6 will be most natural for users coming to R from more traditional OOP languages. In contrast, S3 is a natural FOOP paradigm that will be familiar to all R users (even if they are not aware that S3 is being used). S4 is an unfortunate midpoint between the two, which whilst being very useful, is not particularly user-friendly in terms of programming classes and objects.

distr was developed soon after S4 was released and is arguably one of the best case studies for how well S4 performs. Whilst S4 formalizes S3 to allow for a fully OO interface to be developed, its dependence on inheritance forces design decisions that quickly become problematic. This is seen in the large inheritance trees in **distr** in which one implemented distribution can be nested five child classes deep. This is compounded by the fact that S4 does not use pointer objects but instead nests objects internally. Therefore, **distr** has problems with composite distributions in that they quickly become very large in size. For example, a mixture of two distributions can easily be around 0.5Mb, which is relatively large. In contrast, R6 introduces pointers, which means that a wrapped object simply points to its wrapped component and does not copy it needlessly.

Whilst a fully object-oriented interface can be developed in S3 and S4, they do not have the flexibility of R6, which means that in the long run, extensibility and scalability can be problematic. R6 forces R users to learn a paradigm that they may not be familiar with, but packages like **R62S3** allow users to become acquainted with R6 on a slightly shallower learning curve. Speed differences for the three paradigms are formally compared on the example above using **microbenchmark** (Mersmann, 2019); the results are in table 4. The R6 example is compared both including the construction of the class, `duck$new("Zaphod")$quack()`, and without construction, `d$quack()`, where `d` is the object constructed before comparison. A significant 'bottleneck' is noted when construction is included in the comparison. However, despite this, S4 is still significantly the slowest.

Design patterns

In the simplest definition, 'design patterns' are abstract solutions to common coding problems. They are probably most widely known due to the book 'Design Patterns Elements of Reusable Object-

Paradigm	mean (μs)	cld
S3	73.44	a
S4	276.17	c
R6	187.70	b
R6*	38.32	a

Table 4: Comparing S3, S4, and R6 in calling a method. R6 is tested both including object construction (R6) and without (R6*). 'cld' is the significance testing from **microbenchmark** where 'a' is the fastest and 'c' the slowest. Experiment conducted with **R6** version 2.4.1, **microbenchmark** version 1.4.7, and R version 4.0.2 (2020-06-22) on platform: x86_64-apple-darwin17.0 (64-bit) running under: macOS Catalina 10.15.3.

Oriented Software' (*Design Patterns*) (Gamma et al., 1996). **distr6** primarily makes use of the following design patterns

- Abstract Factory
- Decorator
- Composite
- Strategy

Strategy The strategy pattern is common in modeling toolboxes in which multiple algorithms can be used to solve a problem. This pattern defines an abstract class for a given problem and concrete classes that each implement different strategies, or algorithms, to solve the problem. For example, in the context of mathematical integration (a common problem in R), one could use Simpson's rule, Kronrod's, or many others. These can be specified by an `integrate` abstract class with concrete sub-classes `simpson` and `kronrod` (figure 4).

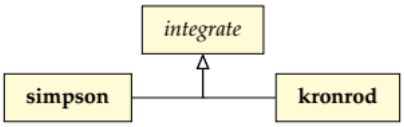


Figure 4: Strategy pattern example.

Composite The composite pattern defines a collection of classes with an identical interface when treated independently or when composed into a single class with constituent parts. To the user, this means that only one interface needs to be learned in order to interact with composite or individual classes. A well-built composite pattern allows users to construct complex classes with several layers of composition and yet still be able to make use of a single interface. By inheriting from a parent class, each class and composite share a common interface. Composition is a powerful design principle that allows both modification of existing classes and reduction of multiple classes (Király et al., 2021).

Decorator Decorators add additional responsibilities to an object without making any other changes to the interface. An object that has been decorated will be identical to its un-decorated counter-part except with additional methods. This provides a useful alternative to inheritance. Whereas inheritance can lead to large tree structures in which each sub-class inherits from the previous and contains all previous methods, decorators allow the user to pick and choose with responsibilities to add. Figure 5 demonstrates how this is useful in a shopping cart example. The top of the figure demonstrates using inheritance in which each sub-class adds methods to the `Cart` parent class. By the `Tax` child class, there are a total of five methods in the interface. At the bottom of the figure, the decorator pattern demonstrates how the functionality for adding items and tax is separated and can be added separately.

Contributions to R6

In order to implement **distr6**, several contributions were made to the R6 paradigm to extend its abilities and to implement the design patterns discussed above.

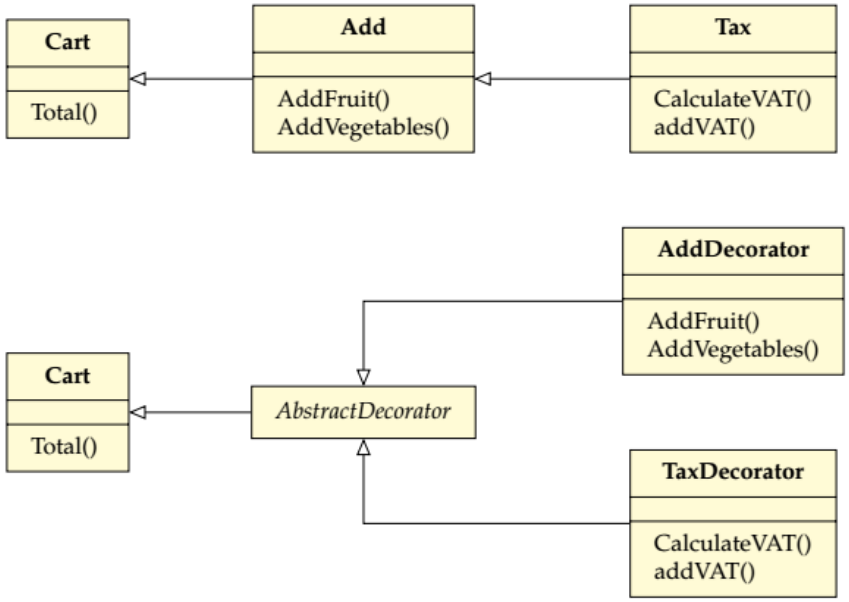


Figure 5: Decorator pattern example.

Abstract classes R6 did not have a concept of abstract classes, which meant that patterns such as adapters, composites, and decorators, could not be directly implemented without problems. This is produced in **distr6** with the `abstract` function, which is placed in the first line of all abstract classes. In the example below, `obj` expects the `self` argument from R6 classes, and `class` is the name of the class, `getR6Class` is a custom function for returning the name of the class of the given object.

```
abstract <- function(obj, class) {
  if (getR6Class(obj) == class) {
      stop(sprintf("%s is an abstract class that can't be initialized.", class))
    }
}
```

For example, in decorators, the following line is placed at the top of the `initialize` function:

```
abstract(self, "DistributionDecorator")
```

Decorators The typical implementation of decorators is to have an abstract decorator class with concrete decorators inheriting from this, each with their own added responsibilities. In **distr6**, this is made possible by defining the `DistributionDecorator` abstract class (see above) with a public `decorate` method. Concrete decorators are simply R6 classes where the public methods are the ones to 'copy' to the decorated object.

```
> DistributionDecorator
<DistributionDecorator> object generator
  Public:
    packages: NULL
    initialize: function ()
    decorate: function (distribution, ...)
    clone: function (deep = FALSE)

> CoreStatistics
<CoreStatistics> object generator
  Inherits from: <DistributionDecorator>
  Public:
    mgf: function (t)
    cf: function (t)
    pgf: function (z)
```

When the `$decorate` method from a constructed decorator object is called, the methods are simply copied from the decorator environment to the object environment. The `decorator()` function simplifies this for the user.

Composite and wrappers The composite pattern is made use of in what **distr6** calls 'wrappers'. Again, this is implemented via an abstract class (`DistributionWrapper`) with concrete sub-classes.

```
> DistributionWrapper
<DistributionWrapper> object generator
  Inherits from: <Distribution>
  Public:
    initialize: function (distlist = NULL, name, short_name, description, support,
    wrappedModels: function (model = NULL)
    setParameterValue: function (..., lst = NULL, error = "warn")
  Private:
    .wrappedModels: list

> TruncatedDistribution
<TruncatedDistribution> object generator
  Inherits from: <DistributionWrapper>
  Public:
    initialize: function (distribution, lower = NULL, upper = NULL)
    setParameterValue: function (..., lst = NULL, error = "warn")
  Private:
    .pdf: function (x, log = FALSE)
    .cdf: function (x, lower.tail = TRUE, log.p = FALSE)
    .quantile: function (p, lower.tail = TRUE, log.p = FALSE)
    .rand: function (n)
```

Wrappers in **distr6** alter objects by modifying either their public or private methods. Therefore, an 'unwrapped' distribution looks identical to a 'wrapped' one, despite inheriting from different classes. This is possible via two key implementation strategies: 1) on the construction of a wrapper, parameters are prefixed with a unique ID, meaning that all parameters can be accessed at any time; 2) the `wrappedModels` public field allows access to the original wrapped distributions. These two factors allow any new method to be called either by reference to `wrappedModels` or by using `$getParameterValue` with the newly prefixed parameter ID. This is demonstrated in the `.pdf` private method of the `TruncatedDistribution` wrapper (slightly abridged).

```
.pdf = function(x, log = FALSE) {
      dist <- self$wrappedModels()[[1]]
      lower <- self$getParameterValue("trunc_lower")
      upper <- self$getParameterValue("trunc_upper")

        pdf <- numeric(length(x))
        pdf[x > lower & x <= upper] <- dist$pdf(x[x > lower & x <= upper]) /
          (dist$cdf(upper) - dist$cdf(lower))

      return(pdf)
    }
```

As the public `pdf` is the same for all distributions, and this is inherited by wrappers, only the private `.pdf` method needs to be altered.

Examples

This final section looks at concrete short examples for four key use cases.

Constructing and querying distributions

The primary use case for the majority of users will be in constructing distributions in order to query their results and visualize their shape.

Below, a distribution (Binomial) is constructed and queried for its distribution-specific traits and parameterization-specific properties.

```
> b <- Binomial$new(prob = 0.1, size = 5)
> b$setParameterValue(size = 6)
> b$getParameterValue("size")
> b$parameters()
> b$properties
> b$traits
```

Specific methods from the distribution are queried as well.

```
> b$mean()
> b$entropy()
> b$skewness()
> b$kurtosis()
> b$cdf(1:5)
```

The distribution is visualized by plotting it's density, distribution, inverse distribution, hazard, cumulative hazard, and survival function; the output is in figure 6.

```
> plot(b, fun = "all")
```

Analysis of empirical data

distr6 can also serve as a toolbox for analysis of empirical data by making use of the three 'empirical' distributions: `Empirical`, `EmpricalMV`, and `WeightedDiscrete`.

First, an empirical distribution is constructed with samples from a standard exponential distribution.

```
> E <- Empirical$new(samples = rexp(10000))
```

The `summary` function is used to quickly obtain key information about the empirical distribution.

```
> summary(E)

Empirical Probability Distribution.

  Quick Statistics
    Mean: 0.105954
    Variance: 1.140673
    Skewness: 0.05808027
    Ex. Kurtosis: -0.473978

 Support: (-2.50, -2.19,...,2.27, 2.66)  Scientific Type: R

 Traits: discrete; univariate
 Properties: asymmetric; platykurtic; positive skew
```

The distribution is compared to a (standard) Normal distribution and then (standard) Exponential distribution; output in figure 7.

```
> qqplot(E, Normal$new(), xlab = "Empirical", ylab = "Normal")
> qqplot(E, Exponential$new(), xlab = "Empirical", ylab = "Exponential")
```

The CDF of a bivariate empirical distribution is visualized; output in figure 8.

```
> plot(EmpiricalMV$new(data.frame(rnorm(100, mean = 3), rnorm(100))), fun = "cdf")
```

Learning from custom distributions

Whilst empirical distributions are useful when data samples have been generated, custom distributions can be used to build an entirely new probability distribution. Though, here, we use a simple discrete uniform distribution. This example highlights the power of decorators to estimate distribution results without manual computation of every possible method. The output demonstrates the precision and accuracy of these results.

Below, a custom distribution is created, and by including the `decorators` argument, all further methods are imputed numerically. The distribution is summarized for properties, traits, and common results (this is possible with the 'CoreStatistics' decorator). The summary is identical to the analytic `DiscreteUniform` distribution.

```
> U <- Distribution$new(
+         name = "Discrete Uniform",
+         type = set6::Integers$new(), support = set6::Set$new(1:10),
+         pdf = function(x) ifelse(x < 1 | x > 10, 0, rep(1/10,length(x))),
+         decorators = c("CoreStatistics", "ExoticStatistics", "FunctionImputation"))
> summary(U)

Discrete Uniform

  Quick Statistics
    Mean: 5.5
    Variance: 8.25
    Skewness: 0
    Ex. Kurtosis: -1.224242

 Support: {1, 2,...,9, 10}  Scientific Type: Z

 Traits: discrete; univariate
 Properties: asymmetric; platykurtic; no skew

 Decorated with:  CoreStatistics, ExoticStatistics, FunctionImputation
```

The CDF and simulation function are called (numerically imputed with the `FunctionImputation` decorator), the hazard function from the `ExoticStatistics` decorator, and the `kthmoment` function from the `CoreStatistics` decorator.

```
> U$cdf(1:10)
[1] 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
> U$rand(10)
[1]  8 10  5  8  5 10  6  7  1  4
> U$hazard(2)
[1] 0.125
> U$kthmoment(2)
[1] 8.25
```

Composite distribution modeling

Composite distributions are an essential part of any distribution software. The following example demonstrates two types of composites: composition via distribution transformation (truncation) and composition via mixtures and vectors.

First, a Binomial distribution is constructed and truncated between 1 and 5, the CDF of the new distribution is queried.

```
> TB <- truncate(
   Binomial$new(size = 20, prob = 0.5),
   lower = 1,
   upper = 5
   )
> round(TB$cdf(0:6), 4)
[1] 0.0000 0.0000 0.0088 0.0613 0.2848 1.0000 1.0000
```

Next, a vector distribution is constructed of two Normal distributions, with respective means 1 and 2 and unit standard deviation. The parameters are queried (some columns suppressed).

```
> V <- VectorDistribution$new(distribution = "Normal",
+                             params = data.frame(mean = 1:2))
> V$parameters()
           id value support
1: Norm1_mean     1       R
2:  Norm1_var     1      R+
3:   Norm1_sd     1      R+
4: Norm1_prec     1      R+
5: Norm2_mean     2       R
6:  Norm2_var     1      R+
7:   Norm2_sd     1      R+
8: Norm2_prec     1      R+
```

Vectorization is possible across distributions, samples, and both. In the example below, the first call to `$pdf` evaluates both distributions at (1, 2), the second call evaluates the first at (1) and the second at (2), and the third call evaluates the first at (1, 2) and the second at (3, 4).

```
> V$pdf(1:2)
       Norm1     Norm2
1: 0.3989423 0.2419707
2: 0.2419707 0.3989423
> V$pdf(1, 2)
       Norm1     Norm2
1: 0.3989423 0.3989423
> V$pdf(1:2, 3:4)
       Norm1      Norm2
1: 0.3989423 0.24197072
2: 0.2419707 0.05399097
```

Finally, a mixture distribution with uniform weights is constructed from a $Normal(2, 1)$ distribution and an $Exponential(1)$.

```
> MD <- MixtureDistribution$new(
+   list(Normal$new(mean = 2, sd = 1),
+        Exponential$new(rate = 1)
+   )
+ )
> MD$pdf(1:5)
[1] 0.304925083 0.267138782 0.145878896 0.036153303 0.005584898
> MD$cdf(1:5)
[1] 0.3953879 0.6823324 0.8957788 0.9794671 0.9959561
> MD$rand(5)
[1] 3.6664473 0.1055126 0.6092939 0.8880799 3.4517465
```

Future work

Whilst **distr6** fulfils its primary purpose as an R6 interface for probability distributions with basic features, it is not consider 'feature-complete', as it currently lacks many of the important features included in **distr** and other related software. **distr6** is in constant development and has an active GitHub with open issues and projects. Some concrete short-term goals include:

- A more generalized convolution interface
- Expanding truncation to other interval types (currently only left-open is supported)
- Extending the `FunctionImputation` decorator to work on higher-order distributions as well as to improve speed and accuracy.
- Exposing further internal functionality for more user-control over numerical results.

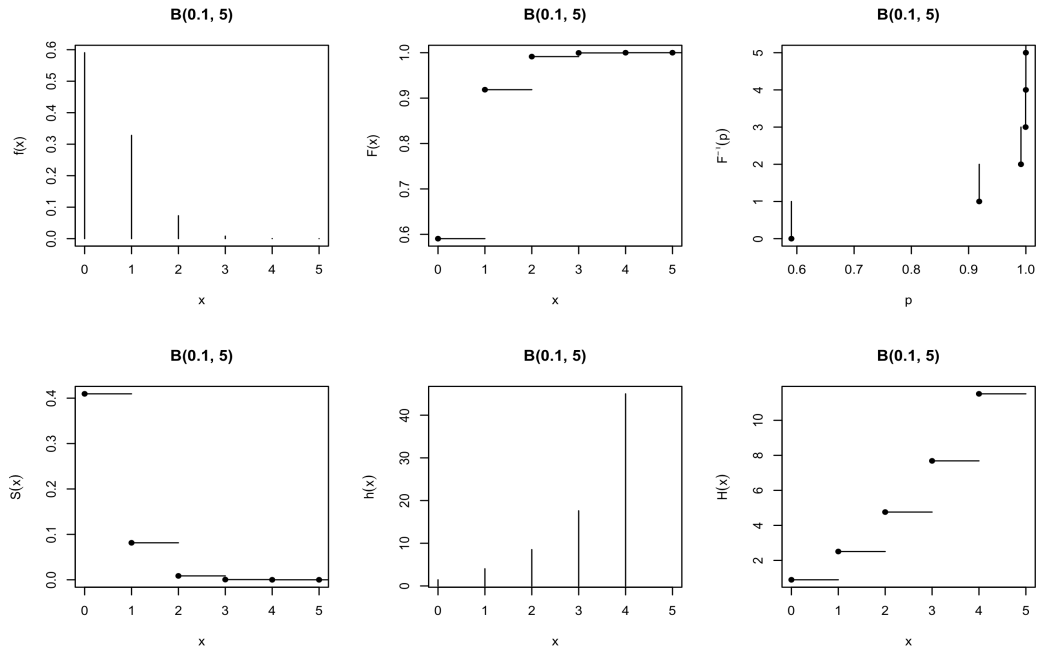


Figure 6: Visualizing a Binomial(0.1, 5) distribution, `b`, with `plot(b, fun = "all")`. Functions clockwise from top-left: probability mass, cumulative distribution, quantile (inverse cdf), cumulative hazard, hazard, and survival.

Summary

distr6 introduces a robust and scalable object-oriented interface for probability distributions to R and aims to be the first-stop for class object-oriented probability distributions in R. By making use of R6, every implemented distribution is clearly defined with properties, traits, and analytic results. Whilst R **stats** is limited to very basic `dpqr` functions for representing evaluated distributions, **distr6** ensures that probability distributions are treated as complex mathematical objects.

Future updates of the package will include adding further numerical approximation strategies in the decorators to allow users to choose different methods (instead of being forced to use one). Additionally, the extensions to R6 could be abstracted into an independent package in order to better benefit the R community.

distr6 is released under the MIT license on GitHub and CRAN. Extended documentation, tutorials, and examples are available at `https://alan-turing-institute.github.io/distr6/`. Code quality is monitored and maintained by an extensive suite of unit tests on multiple continuous integration systems.

Acknowledgments

We would like to thank and acknowledge Prof. Dr. Peter Ruckdeschel and Prof. Dr. Matthias Kohl for their work on the **distr** package and for extensive discussions, planning, and design decisions that were utilised in the development of **distr6**. RS receives a PhD stipend from EPSRC (EP/R513143/1).

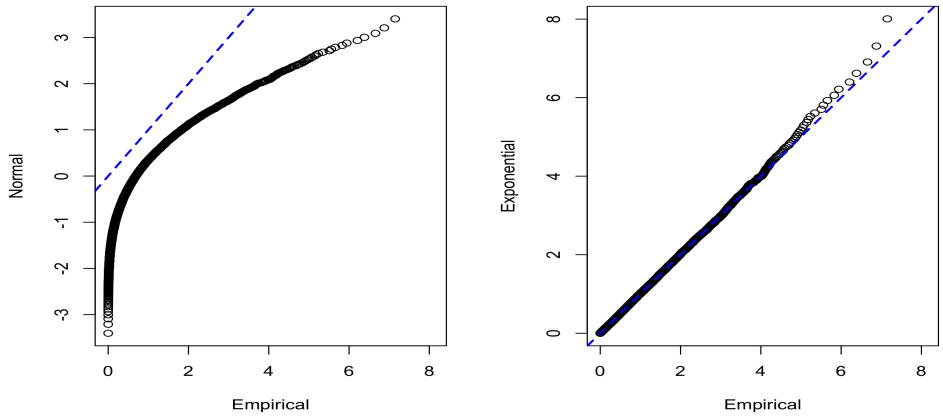


Figure 7: Q-Q plots comparing an unknown Empirical distribution, `E`, to theoretical Normal (left) and Exponential (right) distributions with `qqplot(E, Normal$new()` and `qqplot(E, Exponential$new()` respectively.

Bibliography

J. M. Chambers. Object-Oriented Programming, Functional Programming and R. *Statist. Sci.*, 29 (2):167–180, 2014. ISSN 0883-4237. URL https://projecteuclid.org:443/euclid.ss/1408368569. [p456]

W. Chang. R6: Classes with Reference Semantics, 2018. URL https://cran.r-project.org/package=R6. [p446]

C. Dutang, V. Goulet, and M. Pigeon. actuar: An R Package for Actuarial Science. *Journal of Statistical Software*, 25(7):38, 2008. [p447]

D. Eddelbuettel and R. Francois. Rcpp: Seamless R and C++ Integration. *Journal of Statistical Software*, 40(8):1–18, 2011. URL http://www.jstatsoft.org/v40/i08/. [p451]

E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design Patterns: Elements of Reusable Software, 1996. ISSN ISBN: 0-201-63361-2. [p446, 458]

A. Hayes and R. Moller-Trane. distributions3: Probability Distributions as S3 Objects, 2019. URL https://cran.r-project.org/package=distributions3. [p447]

F. J. Király, M. Löning, A. Blaom, A. Gueciouer, and R. Sonabend. Designing Machine Learning Toolboxes: Concepts, Principles and Patterns. *arXiv*, jan 2021. URL http://arxiv.org/abs/2101.04938. [p458]

D. Lin, J. M. White, S. Byrne, D. Bates, A. Noack, J. Pearson, A. Arslan, K. Squire, D. Anthoff, T. Papamarkou, M. Besançon, J. Drugowitsch, and M. Schauer. JuliaStats/Distributions.jl: a Julia package for probability distributions and associated functions. 2019. doi: 10.5281/zenodo.2647458. [p448]

O. Mersmann. microbenchmark: Accurate Timing Functions, 2019. URL https://cran.r-project.org/package=microbenchmark. [p457]

M. O'Hara-Wild and A. Hayes. distributional: Vectorised Probability Distributions, 2020. URL https://cran.r-project.org/package=distributional. [p447]

P. Ruckdeschel, M. Kohl, T. Stabla, and F. Camphausen. S4 Classes for Distributions, 2006. URL https://cran.r-project.org/package=distr. [p446]

L. Sablica and K. Hornik. mistr: A Computational Framework for Mixture and Composite Distributions. *The R Journal*, 12(1):283, 2020. ISSN 2073-4859. doi: 10.32614/RJ-2020-003. URL https://journal.r-project.org/archive/2020/RJ-2020-003/index.html. [p447]

R. Sonabend. R62S3: Automatic Method Generation from R6, may 2019. URL https://cran.r-project.org/package=R62S3. [p449]

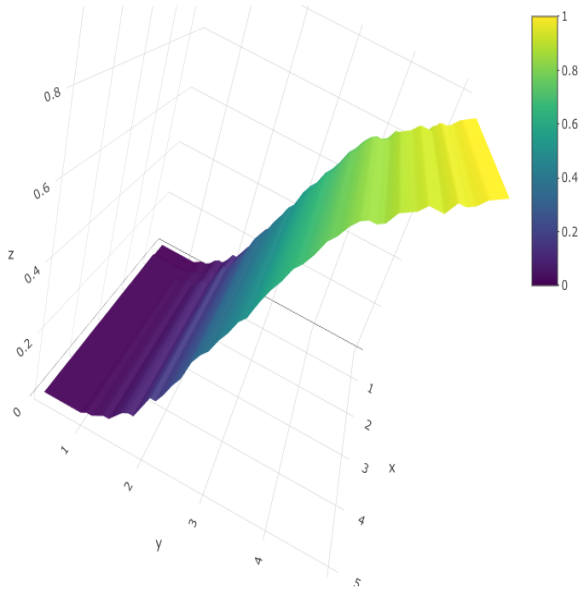


Figure 8: Visualizing a bivariate empirical distribution, `E`, with `plot("E", fun = "cdf")`.

R. Sonabend and F. J. Kiraly. set6: R6 Mathematical Sets Interface. *Journal of Open Source Software*, 5(55): 2598, nov 2020. ISSN 2475-9066. doi: 10.21105/joss.02598. URL https://joss.theoj.org/papers/10.21105/joss.02598. [p451]

T. Wolodzko. extraDistr: Additional Univariate and Multivariate Distributions, 2019. URL https://cran.r-project.org/package=extraDistr. [p447]

Raphael Sonabend
Department of Statistical Science
University College London
Gower Street
London, WC1E 6BT, United Kingdom
0000-0001-9225-4654
raphael.sonabend.15@ucl.ac.uk

Franz J. Király
Department of Statistical Science
University College London
Gower Street
London, WC1E 6BT, United Kingdom
f.kiraly@ucl.ac.uk

gofCopula: Goodness-of-Fit Tests for Copulae

by Ostap Okhrin, Simon Trimborn and Martin Waltz

Abstract The last decades show an increased interest in modeling various types of data through copulae. Different copula models have been developed, which lead to the challenge of finding the best fitting model for a particular dataset. From the other side, a strand of literature developed a list of different Goodness-of-Fit (GoF) tests with different powers under different conditions. The usual practice is the selection of the best copula via the p-value of the GoF test. Although this method is not purely correct due to the fact that non-rejection does not imply acception, this strategy is favored by practitioners. Unfortunately, different GoF tests often provide contradicting outputs. The proposed R-package brings under one umbrella 13 most used copulae - plus their rotated variants - together with 16 GoF tests and a hybrid one. The package offers flexible margin modeling, automatized parallelization, parameter estimation, as well as a user-friendly interface, and pleasant visualizations of the results. To illustrate the functionality of the package, two exemplary applications are provided.

Introduction

Being firstly introduced in 1959 by Abe Sklar (see Sklar (1959)), copulae gained enormous popularity in applications in the last two decades. Researchers from different fields recognize the power of copulae while working with multivariate datasets from insurance (Fang and Madsen, 2013; Shi et al., 2016), finance (Salvatierra and Patton, 2015; Oh and Patton, 2018), biology (Konigorski et al., 2014; Dokuzoğlu and Purutçuoğlu, 2017), hydrology (Liu et al., 2018; Valle and Kaplan, 2019), medicine (Kuss et al., 2014; Gomes et al., 2019), traffic engineering, (Huang et al., 2017; Ma et al., 2017), etc. For a recent review, we refer to Größer and Okhrin (2021). Unfortunately, the correct specification of the multivariate distribution is not easy to find, and often interest in the understanding of the functional form of the copula is dominated by the expected performance of the whole model. This is natural, taking into account the huge list of different copula models proposed in the literature for different needs; see, e.g., Durante and Sempi (2010), Joe and Kurowicka (2010), or Genest and Nešlehová (2014). Although an expanding list of R-packages devoted to copulae is existent, the issue of GoF testing is less frequently addressed. Primarily, GoF tests for copulae are implemented in copula comparison packages as **copula** (Hofert et al., 2020), **TwoCop** (Remillard and Plante, 2012), and **VineCopula** (Nagler et al., 2019), but since Remillard and Scaillet (2009) and Genest et al. (2009), many other powerful tests were developed that are not integrated into these packages. Most of the tests focus on the bivariate case, leaving a further gap in the existing R-package landscape.

Given a variety of tests, the selection of the most appropriate copula seems simple at first glance. However, the power of these tests varies significantly depending on the use case and the copula tested for. The absence of the overall best GoF test leads researchers and practitioners often to the selection of the test (and copula), which supports some subjective expectation, but not the copula that fits the data at its best. Although GoF tests are not intended for model selection but rather to decide whether the selected copula is *not* suitable for the data, the model selection strategy based on the rank of the p-values is still commonly used. Following proper scoring rules (Gneiting and Raftery, 2007), some tests still allow for selection, and even if not purely statistically sound, it is heavily advocated among practitioners; see De Valpine (2014). An eloquent illustrative example of different powers and contradictory decisions was provided in Zhang et al. (2016), where three different tests (R_n by Zhang et al. (2016), S_n by Genest et al. (2009), and J_n by Scaillet (2007)) were applied for testing the dependency between the standardized returns of the Bank of America and Citigroup. The model selection was done from three copula models: normal, Gumbel, and t-copula, based on their respective p-values. Interestingly,

1) for the year 2004, the R_n gave a favor for the Gumbel copula, while the S_n and J_n for the normal one;
2) for the year 2006, the S_n gave the favor for the normal copula, while R_n and J_n for the t-copula;
3) for the year 2009, the J_n indicated that the dependency is close to the normal one, while R_n and S_n were in favor of the Gumbel copula.

This implies that for each year, a *different* pair of tests returns consistent results. In an empirical study, it is difficult to decide which *copula* is suitable and which *test* provides the most plausible results. An extensive comparative study of different GoF tests was performed a decade ago by Genest et al. (2009), intensively discussing all, up to that moment existing, tests for copulae. The main findings are that

there is no superior blanket test, but several tests have very good power under different, often disjunct conditions. A test proposed by Zhang et al. (2016) fills some gaps in the set of models under which this test performs better than others under certain conditions. However, a common phenomenon in empirical studies is the interpretation of the non-rejection of a copula as the correct model. Especially, in situations where the used GoF test has low power, this is not necessarily the case. Tackling this issue, Zhang et al. (2016) also developed the hybrid test, which is simple in construction and implementation. It combines the power of different tests and is very helpful for practitioners; see Section 2.6. However, even in this case, the interpretation of finding the *correct* copula should be treated with care.

We propose the R-package **gofCopula** to automatize the whole empirical procedure of selecting the most suitable copula. Table 1 displays the broad range of available tests, copula models, and the maximum dimension. The latest version of this table is also accessible via the function `CopulaTestTable()` in the package. Further details on the functionality of each test are provided in Section 2.3, while Table A.1 of Appendix A contains some characteristics of the copulae implemented in the package.

Test	normal	t	clayton	gumbel	frank	joe	amh	galambos	huslerReiss	tawn	tev	fgm	plackett
gofCvM	≥ 2	≥ 2	≥ 2	≥ 2	≥ 2	≥ 2	2	2	2	2	2	2	2
gofKS	≥ 2	≥ 2	≥ 2	≥ 2	≥ 2	≥ 2	2	2	2	2	2	2	2
gofKendallCvM	≥ 2	≥ 2	≥ 2	≥ 2	≥ 2	≥ 2	2	2	2	2	2	2	2
gofKendallKS	≥ 2	≥ 2	≥ 2	≥ 2	≥ 2	≥ 2	2	2	2	2	2	2	2
gofRosenblattSnB	≥ 2	≥ 2	≥ 2	≥ 2	≥ 2	≥ 2	2	2	-	-	-	2	2
gofRosenblattSnC	≥ 2	≥ 2	≥ 2	≥ 2	≥ 2	≥ 2	2	2	-	-	-	2	2
gofRosenblattGamma	≥ 2	≥ 2	≥ 2	≥ 2	≥ 2	≥ 2	2	2	-	-	-	2	2
gofRosenblattChisq	≥ 2	≥ 2	≥ 2	≥ 2	≥ 2	≥ 2	2	2	-	-	-	2	2
gofArchmSnB	-	-	≥ 2	≥ 2	≥ 2	≥ 2	2	-	-	-	-	-	-
gofArchmSnC	-	-	≥ 2	≥ 2	≥ 2	≥ 2	2	-	-	-	-	-	-
gofArchmGamma	-	-	≥ 2	≥ 2	≥ 2	≥ 2	2	-	-	-	-	-	-
gofArchmChisq	-	-	≥ 2	≥ 2	≥ 2	≥ 2	2	-	-	-	-	-	-
gofKernel	2	2	2	2	2	2	2	2	2	2	2	2	2
gofWhite	2	2	2	2	2	2	-	-	-	-	-	-	-
gofPIOSTn	3	2	3	3	3	3	2	2	-	-	-	2	2
gofPIOSRn	3	2	3	3	3	3	2	2	-	-	-	2	2

Table 1: Implemented tests, copula models (columns), and the maximum available dimension of each test-copula combination. "-" means this combination is not available, "2" is available in dimension two, "3" in dimensions two and three, and "≥ 2" in any dimension. `amh` corresponds to the Ali-Mikhail-Haq copula, `tev` to the t-extreme value copula, and `fgm` to the Farlie-Gumbel-Morgenstern copula.

In summary, the package **gofCopula** offers the following attractive features which distinguish it from other R-packages:

- Each of the 13 copulae in Table 1 is available in a rotated form for the bivariate case. Furthermore, the flexible hybrid test is implemented to aggregate the results of the 16 tests.
- We provide an interface to integrate new GoF tests. The users can provide their own test statistics and perform the tests with the integrated parametric bootstrap and also make use of the automatized parallelization of **gofCopula**. The new tests can be further combined with other tests via the hybrid test.
- The whole copula community relies justifiably on the R-package **copula** for conducting different studies on copulae. Thus we provide an interface to use objects from the R-package **copula** and perform the GoF tests with **gofCopula**.
- For the estimation of the margins, ten different parametric distributions are available, in addition to the nonparametric estimation per default.
- GoF tests rely on bootstrapping methods which can result in substantially high computational costs. In contrast to other R-packages, the package **gofCopula** comes with an integrated option for automatized parallelization of the bootstrapping samples. For the convenience of the user, the parallelization can be activated by specifying the number of parallel jobs as an argument of the functions.

- As we believe in reproducible science, the user has the opportunity to specify the seeds for the bootstrapping procedure in order to guarantee full reproducibility of all results gained from the package **gofCopula**.
- An estimation function for the computation time is implemented, which fits a regression model to give an estimate for the time adapted to the users' machine.
- An informative console output is implemented, which keeps the user informed about the current test and copula under estimation, as well as the remaining time until the derivation of the test is performed. The latter functionality is supported by the R-package **progress** (Csárdi and FitzJohn, 2019).
- The results of the tests are provided with the package's own class `"gofCOP"`, which allows for a comprehensive overview of the test results. For a better comparison of the results, we extend the generic `plot` function for objects of class `"gofCOP"`, which illustrates the results in a convenient manner. The `plot` function is supported by the R-package **yarrr** (Phillips, 2017), and was customized to provide the user an insightful figure for the interpretation of the results.

The test statistics of six GoF tests were already implemented in R-packages. Thus, for the computation of the test statistics of some tests, we use the functions `gofTstat` and `BiCopGofTest` from the packages **copula** and **VineCopula**, respectively. For obvious reasons, we did not implement all existing tests on copulae, but we will embed new tests in the proposed package as soon as they become more relevant and actively used among academics and practitioners.

The paper, introducing the R-package **gofCopula**, is structured as follows: The tests and methodology implemented in the package are introduced in Sections 2.2 and 2.3 before presenting the functionalities of the package in Section 2.4. We explain major functions, how to apply them, and elaborate the main arguments of each function. The explanations are supported by R-code and output. To provide an impression of the runtime of various tests, we discuss the speed of the tests depending on the copula to test for, the number of observations, and the number of bootstrap samples. A simulated example (Section 2.5) contains a typical step-by-step procedure of how the package can be used in practice, which is also applied to two real-world examples (Section 2.6), in which all corresponding codes are given and explained. The cases are illustrated with interpretations of the console output and plots, both generated directly from **gofCopula**, without any additional code. The results of the two applications can be fully reproduced by the **gofCopula** package, which also contains the used datasets. All illustrations, simulations, and applications in this paper are fully reproducible and designed to guide the user into conducting their own research with the **gofCopula** package.

Estimation methods

Consider a d-dimensional random vector $X = \{X_1, \dots, X_d\}$ with corresponding marginal distributions F_j, $j = 1, \dots, d$. The multivariate distribution $F(x_1, \dots, x_d)$ can be decomposed via the copula function $C_\theta(u_1, \dots, u_d)$ as

$$(X_1, \dots, X_d) \sim F(x_1, \dots, x_d) = C_\theta\{F_1(x_1), \dots, F_d(x_d)\}.$$

Having a sample $\mathcal{X} = \{x_{ij}\}$, $i = 1, \dots, n$, $j = 1, \dots, d$ of size n with the columns defined as $\mathbf{x}_j$, the main task for empirical studies is to estimate the copula parameter θ and the margins F_j, $j = 1, \dots, d$ for a given copula specification. Since the properties and goodness of the estimator of θ heavily depend on the estimators of the latter, their choice is also of importance.

In the bivariate case, one of the standard methods of estimation of the univariate parameter θ is based on Kendall's τ by Genest and Rivest (1993). Following Joe (1997) for (X_1, X_2), and (X_1', X_2') being independent random pairs with continuous distribution F, Kendall's τ is defined via:

$$\tau = \mathrm{P}\{(X_1 - X_1')(X_2 - X_2') > 0\} - \mathrm{P}\{(X_1 - X_1')(X_2 - X_2') < 0\}.$$

This can be written in terms of the underlying copula C in form of $\tau = 4 \int C dC - 1$, linking Kendall's τ and the copula parameter of interest under a correct copula specification, e.g., for the normal copula, the equality $\tau = \frac{2}{\pi} \arcsin \theta$ holds, with θ being the copula correlation, c.f. Demarta and McNeil (2005). Thus, the parameter θ can be estimated via inversion of this relation and replacement of τ by its empirical counterpart. However, as shown in Genest et al. (1995), the ML method leads to substantially more efficient estimators. Therefore, we employ it as the first option in our study. The second reason for ML estimation is the fact that several implemented tests are based on the likelihood ratios. Thus, results based on Kendall's τ will not be supported by the theory behind these tests. The ML procedure

can be performed for the parameters of the margins and of the copula function simultaneously:

$$(\hat{\theta}, \hat{\alpha}_1, \ldots, \hat{\alpha}_d)^\top = \underset{\theta, \alpha_1, \ldots, \alpha_d}{\operatorname{argmax}} \mathcal{L}(\mathcal{X}, \theta, \alpha_1, \ldots, \alpha_d), \tag{1}$$

$$\text{with} \quad \mathcal{L}(\mathcal{X}, \theta, \alpha_1, \ldots, \alpha_d) = \sum_{i=1}^{n} \log \left[c\{F_1(x_{i1}, \alpha_1), \ldots, F_d(x_{id}, \alpha_d), \theta\} \prod_{j=1}^{d} f_j(x_{ij}, \alpha_j) \right], \tag{2}$$

where $c(\cdot)$ is the copula density, $\alpha_1, \ldots, \alpha_d$ are parameters of the margins, $f_j(\cdot)$ are marginal densities, and $\mathcal{L}(\cdot)$ is the full log-likelihood function. Nevertheless, simultaneous maximization of the function in (1) is very computationally intensive. Therefore, we consider only two-stage procedures, where at the first stage, we estimate margins parametrically (c.f. Joe (1997) and Joe (2005)) as

$$F_j(x, \hat{\alpha}_j) = F_j \left\{ x, \underset{\alpha}{\operatorname{argmax}} \sum_{i=1}^{n} \log f_j(x_{ij}, \alpha) \right\}, \quad \text{for} \quad j = 1, \ldots, d, \tag{3}$$

or nonparametrically (c.f. Chen and Fan (2006) and Chen et al. (2006)) as

$$\widehat{F}_j(x) = (n+1)^{-1} \sum_{i=1}^{n} \mathbf{1}\{x_{ij} \leq x\}, \quad j = 1, \ldots, d, \tag{4}$$

with $\mathbf{1}$ being the indicator function. Afterward, the copula parameter is estimated in the second step as

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} \sum_{i=1}^{n} \log c\{\widetilde{F}_1(x_{i1}), \ldots, \widetilde{F}_d(x_{id}), \theta\}, \tag{5}$$

where $\widetilde{F}(x) \in \{\widehat{F}(x), F(x, \hat{\alpha})\}$ are parametrically or nonparametrically estimated margins. In the case of parametric margins, one shall be aware that the two-step approach does not lead to efficient estimators, though the loss in the efficiency is moderate and mainly depends on the strength of dependencies (Joe, 1997). The method of nonparametric estimation of the marginal distributions for copula estimation was first used in Oakes (1994) and further investigated in Genest et al. (1995) and Shih and Louis (1995).

Furthermore, Fermanian and Scaillet (2003) and Chen and Huang (2007) consider a fully nonparametric estimation of the copula, which is heavily used in the GoF testing. It is called an empirical copula and is shown to be a consistent estimator of the true underlying copula, c.f. Gaensler and Stute (1987) and Radulovic and Wegkamp (2004). This estimator is defined as

$$C_n(u_1, \ldots, u_d) = n^{-1} \sum_{i=1}^{n} \prod_{j=1}^{d} \mathbf{1}\{\widehat{F}_j(x_{ij}) \leq u_j\}.$$

Goodness-of-fit tests for copulae

Having a list of different copulae, the most suitable one for real applications still needs to be found and motivated. For this purpose, a series of different GoF tests has been developed in the last decades. Several authors, e.g., Genest et al. (2009), tested the power of those tests against each other and showed that no superior test for all possible situations exist. We cover 16 tests and implement them into the **gofCopula** package. Most of these tests work with the parametric family of copulae denoted by $C_0 = \{C_\theta; \theta \in \mathbb{A} \subset \mathbb{R}^p\}$ for some integer $p \geq 1$ and the copula C, under the general H_0-hypothesis:

$$H_0 : C \in C_0.$$

We differentiate seven groups of GoF tests for copulae based on: (1) empirical copula process; (2) Kendall's process; (3) Rosenblatt integral transform; (4) transformation for Archimedean copulae; (5) Kernel density; (6) White's information matrix equality; and (7) pseudo in-and-out-of-sample (PIOS) estimator.

Empirical copula process

The first group is based on the most natural approach: the deviation of the empirical copula C_n from the parametric copula $C(u_1, \ldots, u_d; \theta)$, captured by the empirical copula process $\sqrt{n}\{C_n(u_1, \ldots, u_d) - C(u_1, \ldots, u_d; \theta)\}$. Based on an estimation of the parametric copula $C(u_1, \ldots, u_d; \hat{\theta})$, the following

process can be defined:

$$\mathbb{C}_n(u_1,\ldots,u_d) = \sqrt{n}\{C_n(u_1,\ldots,u_d) - C(u_1,\ldots,u_d;\hat{\theta})\}.$$

Different measures of divergence can be constructed to evaluate $\mathbb{C}_n$; see Fermanian (2005) and Genest and Rèmillard (2008). We implemented two commonly applied approaches using the Cramér-von Mises and Kolmogorov-Smirnov statistics:

$$S_n^E = \int_{[0,1]^d} \mathbb{C}_n(u_1,\ldots,u_d)^2 dC_n(u_1,\ldots,u_d), \qquad T_n^E = \sup_{u_1,\ldots,u_d\in[0,1]^d} |\mathbb{C}_n(u_1,\ldots,u_d)|.$$

Notice that the Cramér-von Mises statistic yields better performances in most cases (Genest et al., 2009). The evaluation of the d-dimensional integral in practice uses numerical approximations, and the test statistic S_n^E has been already implemented in the **copula** package as function `gofTstat`, so we included it into our package. The tests are later denoted by `gofCvM` and `gofKS`, respectively.

Kendall's process

The tests from the second group were developed and investigated by Genest and Rivest (1993), Wang and Wells (2000), Genest et al. (2006). The main idea behind them is to use the copula-based random variable:

$$C\{F_1(X_1),\ldots,F_d(X_d);\theta\} \sim K(\cdot,\theta), \tag{6}$$

where $K(\cdot,\theta)$ is the univariate Kendall's distribution (not uniform in general); see Barbe et al. (1996), Jouini and Clemen (1996). The empirical version of $K(\cdot)$ is given through:

$$K_n(v) = n^{-1}\sum_{i=1}^{n} \mathbf{1}\left[C_n\{\widehat{F}_1(x_{i1}),\ldots,\widehat{F}_d(x_{id})\} \le v\right], \quad v\in[0,1].$$

Based on the definition of Kendall's process $\sqrt{n}\{K_n(v) - K(\cdot,\theta)\}$ and a parametric $K(\cdot,\hat{\theta})$ estimated with the parameter $\hat{\theta}$, we can define an empirical process as

$$\mathbb{K}_n(v) = \sqrt{n}\{K_n(v) - K(v,\hat{\theta})\}. \tag{7}$$

On this basis, two applicable test statistics are Cramér-von Mises and Kolmogorov-Smirnov; see Genest et al. (2006).

$$S_n^{(K)} = \int_0^1 \mathbb{K}_n(v)^2 dK(v,\hat{\theta}), \qquad T_n^{(K)} = \sup_{v\in[0,1]} |\mathbb{K}_n(v)|.$$

Worth mentioning are the different null hypotheses $H_0'' : K \in \mathcal{K}_0 = \{K(\cdot,\theta) : \theta \in \Theta\}$ of these tests. Since $H_0 \subset H_0''$, the non-rejection of H_0'' does not imply non-rejection of H_0. However, for bivariate Archimedean copulae, H_0'' and H_0 are equivalent (Genest et al., 2009). Both tests are later denoted as `gofKendallCvM` and `gofKendallKS`, respectively.

Rosenblatt transform

Under the assumption of copula dependency, the conditional distribution of U_i given $U_1,\ldots,U_{i-1}$ is specified through:

$$\begin{aligned} C_d(u_i|u_1,\ldots,u_{i-1}) &= \mathrm{P}\{U_i \le u_i | U_1 = u_1 \ldots U_{i-1} = u_{i-1}\} \\ &= \frac{\partial^{i-1} C(u_1,\ldots,u_i,1,\ldots,1)/\partial u_1 \ldots \partial u_{i-1}}{\partial^{i-1} C(u_1,\ldots,u_{i-1},1,\ldots,1)/\partial u_1 \ldots \partial u_{i-1}}. \end{aligned}$$

The Rosenblatt transform (c.f. Rosenblatt, 1952) is then defined as follows.

Definition 1 *Rosenblatt's probability integral transform of a copula C is the mapping $\mathfrak{R} : (0,1)^d \to (0,1)^d$, $\mathfrak{R}(u_1,\ldots,u_d) = (e_1,\ldots,e_d)$ with $e_1 = u_1$ and $e_i = C_d(u_i|u_1,\ldots,u_{i-1})$, $\forall i = 2,\ldots,d$.*

If the copula is correctly specified, the variables $(e_1,\ldots,e_d)^\top$ resulting from the Rosenblatt transform should be independent from each other and uniformly distributed. Therefore, the null hypothesis $H_0 : C \in C_0$ is equivalent to

$$H_{0R} : (e_1,\ldots,e_d)^\top \sim \Pi, \tag{8}$$

where $\Pi(u_1,\ldots,u_d) = u_1 \cdot \ldots \cdot u_d$ is the product (independence) copula.

Two different types of tests may be constructed using this property. In the first type, similar to the previous two groups, we measure the deviation of the product copula of $(e_1, \ldots, e_d)^\top$ from the corresponding empirical copula:

$$D_n(u_1, \ldots, u_d) = n^{-1} \sum_{i=1}^{n} \prod_{j=1}^{d} \mathbf{1}\{e_{ij} \leq u_j\}.$$

Thus, following Genest et al. (2009), two Cramér-von Mises statistics result:

$$S_n^{(B)} = n \int_{[0,1]^d} \{D_n(u_1, \ldots, u_d) - \Pi(u_1, \ldots, u_d)\}^2 du_1 \cdots du_d,$$
$$S_n^{(C)} = n \int_{[0,1]^d} \{D_n(u_1, \ldots, u_d) - \Pi(u_1, \ldots, u_d)\}^2 dD_n(u_1, \ldots, u_d).$$

Since the H_0 changed to H_{0R}, the tests evaluate the difference of $D_n(u)$ to the product copula. In the package, these tests are defined as `gofRosenblattSnB` and `gofRosenblattSnC`, respectively.

The second type of test uses the fact that a specific combination of independent uniformly distributed random variables follows some known distribution. Based on this, two further Anderson-Darling type tests were introduced by Breymann et al. (2003). By defining

$$G_{i,\Gamma} = \Gamma_d \left\{ \sum_{j=1}^{d} (-\log e_{ij}) \right\},$$

where $\Gamma_d(\cdot)$ is the Gamma distribution with shape d and scale 1 and

$$G_{i,\chi^2} = \chi^2_d \left[\sum_{j=1}^{d} \{\Phi^{-1}(e_{ij})\}^2 \right],$$

where $\chi^2_d(\cdot)$ is the Chi-squared distribution with d degrees of freedom and Φ being the standard normal distribution. It results:

$$T_n = -n - \sum_{i=1}^{n} \frac{2i-1}{n} [\log G_{(i)} + \log\{1 - G_{(n+1-i)}\}],$$

where $G_{(i)}$ is the i-th ordered observation of the $G_{i,\Gamma}$ or G_{i,χ^2}. One should note that Anderson-Darling type tests have almost no power and even do not capture the type 1 error (Dobrić and Schmid, 2007), while the Cramér-von Mises tests behave much more satisfactory (Genest et al., 2009). Furthermore, the basic assumption of uniformly distributed and independent observations after applying the Rosenblatt transform is violated since those variables are not mutually independent and only approximately uniform. The latter two tests are denoted in the package as `gofRosenblattGamma` and `gofRosenblattChisq`, respectively, and are obtained via the function `gofTstat` from the package **copula**.

Transformation for Archimedean copulae

Recently, Hering and Hofert (2015) proposed a procedure of GoF testing based on a transformation similar to the one of Rosenblatt (1952) specifically designed for Archimedean copulae.

Definition 2 *Hering and Hofert's transformation of an Archimedean copula C of dimension $d \geq 2$ with d-monotone generator ψ and continuous Kendall distribution K is the mapping $\mathfrak{T} : (0,1)^d \to (0,1)^d$, $\mathfrak{T}(u_1, \ldots, u_d) = (v_1, \ldots, v_d)$ with $v_i = \left\{ \frac{\sum_{k=1}^{i} \psi^{-1}(u_k)}{\sum_{k=1}^{i+1} \psi^{-1}(u_k)} \right\}^i$, $\forall i = 1, \ldots, d-1$ and $v_d = K\{C(u_1, \ldots, u_d)\}$.*

Distribution function K is estimated empirically, and the variables $(v_1, \ldots, v_d)^\top$ are independent and uniformly distributed if the copula is correctly specified. Note that this transformation was originally considered in Wu et al. (2007) as a method for generating random numbers from Archimedean copulae, such as the inverse of the Rosenblatt transform can be used for sampling copulae. Following Hering and Hofert (2015), the main advantage of this approach in comparison to tests based on the Rosenblatt transform is the more convenient computation in higher dimensions, in which the Rosenblatt procedure is numerically challenging and unstable.

The null hypothesis equals (8) from the tests based on Rosenblatt's probability integral transform: $H_{0T} : (v_1, \ldots, v_d)^\top \sim \Pi$ with Π being the product copula. Consequently, the approaches to test it are

identical. In analogy to the naming introduced in Section 2.3.3, we denoted the tests as `gofArchmSnB`, `gofArchmSnC`, `gofArchmGamma`, and `gofArchmChisq` in the package.

Kernel density estimator

A test from this group has been introduced by Scaillet (2007). Following his approach, a d-variate quadratic kernel $\mathcal{K}$ with bandwidth $H = 2.6073n^{-1/6}\widehat{\Sigma}^{1/2}$ is used, with $\widehat{\Sigma}$ being a sample covariance matrix with $\widehat{\Sigma}^{1/2}$ its Cholesky decomposition. Using $\mathcal{K}_H(y_1,\dots,y_d) = \mathcal{K}(H^{-1}\{y_1,\dots,y_d\}^\top)/\det(H)$, the copula density is nonparametrically estimated by

$$\hat{c}(u_1,\dots,u_d) = n^{-1}\sum_{i=1}^{n}\mathcal{K}_H[(u_1,\dots,u_d)^\top - \{\widetilde{F}_1(x_{i1}),\dots,\widetilde{F}_d(x_{id})\}^\top],$$

where under $\widetilde{F}_i(\cdot)$, we consider nonparametric as well as parametric estimators of the margins. The test statistic is then:

$$J_n = \int_{[0,1]^d}\{\hat{c}(u_1,\dots,u_d) - \mathcal{K}_H * c(u_1\dots,u_d;\hat{\theta})\}^2 w(u_1,\dots,u_d)du_1\cdots du_d, \tag{9}$$

with "$*$" being a convolution operator, $w(u_1,\dots,u_d)$ a weight function, and $c(u_1,\dots,u_d;\hat{\theta})$ the copula density under the H_0, with estimated copula parameter $\hat{\theta}$. Note that the integral is computed numerically using the Gauss-Legendre quadrature method; see Scaillet (2007). The number of knots can be specified via the argument `nodes.Integration`. A scaling parameter for H is implemented via `delta.J`, and the internal size of the bootstrapping samples can be controlled via `MJ`. This test is denoted by `gofKernel` in the package.

White test

This test was introduced by Huang and Prokhorov (2014) and had its foundation in the information matrix equality stated by White (1982). Given the presence of certain regularity conditions, the White equality establishes a connection between the negative sensitivity matrix $\mathbb{S}(\theta)$, and the variability matrix $\mathbb{V}(\theta)$ defined as

$$\mathbb{S}(\theta) = -\,\mathrm{E}_0\left[\frac{\partial^2}{\partial\theta\partial\theta^\top}\log c\{F_1(x_1),...,F_d(x_d);\theta\}\right],$$
$$\mathbb{V}(\theta) = \mathrm{E}_0\left(\left[\frac{\partial}{\partial\theta}\log c\{F_1(x_1),...,F_d(x_d);\theta\}\right]\left[\frac{\partial}{\partial\theta}\log c\{F_1(x_1),...,F_d(x_d);\theta\}\right]^\top\right),$$

where E_0 is the expectation under correct model specification, which is represented by the null hypothesis to be specified. The equality states:

$$\mathbb{S}(\theta) = \mathbb{V}(\theta).$$

Using this approach, the testing problem can be formulated as follows:

$$H_{0W} : \mathbb{S}(\theta) = \mathbb{V}(\theta) \qquad \text{vs.} \qquad H_{1W} : \mathbb{S}(\theta) \neq \mathbb{V}(\theta).$$

Following Schepsmeier (2015), a test statistic is based on empirical versions of the two information matrices, denoted by $\widehat{\mathbb{S}}(\hat{\theta})$ and $\widehat{\mathbb{V}}(\hat{\theta})$. These are aggregated via $d(\hat{\theta}) = \mathrm{vech}\{\widehat{\mathbb{S}}(\hat{\theta}) + \widehat{\mathbb{V}}(\hat{\theta})\}$ with vech denoting vectorization of the lower triangular of a matrix. As a result, $d(\hat{\theta})$ is a vector of dimension $\frac{p(p+1)}{2}$, given the copula parameter vector is of dimension p. It can be shown that the constructed test statistics:

$$T_W = n\{d(\hat{\theta})\}^\top \hat{A}_{\hat{\theta}}^{-1} d(\hat{\theta}),$$

with $\hat{A}_{\hat{\theta}}^{-1}$ being the sample estimator of the asymptotic covariance matrix of $\sqrt{n}d(\hat{\theta})$, follows asymptotically a χ^2 distribution with $\frac{p(p+1)}{2}$ degrees of freedom. For the derivation of the test statistic, this test relies on the function `BiCopGofTest` from the **VineCopula** package, again, in order to avoid code redundancy. Note that the implementation of the test can be unstable for the t-copula; see Nagler et al. (2019). This is the reason why it could not be computed in some cases of the second empirical example in Section 2.6.2. This test is called `gofWhite` in the package.

Cross-validated tests

A recent test using a leave-one-block strategy, and its approximation were introduced by Zhang et al. (2016). Authors derive $\hat{\theta}$ as in (5) and compare it with $\hat{\theta}_{-b}$, $1 \leq b \leq B$, which are delete-one-block pseudo ML estimates:

$$\hat{\theta}_{-b} = \underset{\theta \in \Theta}{\operatorname{argmax}} \sum_{b' \neq b}^{B} \sum_{i=1}^{m} \log c\{\widetilde{F}_1(x_{i1}), \ldots, \widetilde{F}_d(x_{id}); \theta\}, \quad b = 1, \ldots, B,$$

where B is the number of non-overlapping blocks and m the length of each block. Note that in the general setting, these blocks need not be of the same size. However, we follow here the approach of Zhang et al. (2016), who restrict themselves to the same length case of each block. This assumption also simplifies the usage in terms of many parameters. The resulting test statistics,

$$T_n(m) = \sum_{b=1}^{B} \sum_{i=1}^{m} \left[\log \frac{c\{\widetilde{F}_1(x_{i1}), \ldots, \widetilde{F}_d(x_{id}); \hat{\theta}\}}{c\{\widetilde{F}_1(x_{i1}), \ldots, \widetilde{F}_d(x_{id}); \hat{\theta}_{-b}\}} \right], \tag{10}$$

compares the full likelihood, "in-sample", against the resulting likelihoods from the leave-one-block out estimation, "out-of-sample". If the data in each block significantly influence the estimation of the copula parameter under the null hypothesis, then the chosen copula model is inadequate to represent the data.

Depending on the number of blocks, B, a possibly huge amount of dependence parameter estimations have to be performed to get (10). In the case of equal length of each block, $\lceil \frac{n}{m} \rceil$ parameters should be computed. To overcome this drawback, under suitable regularity conditions, Zhang et al. (2016) proposed the test statistic asymptotically equivalent to (10):

$$R_n = \operatorname{tr}\{\widehat{\mathbb{S}}(\hat{\theta})^{-1} \widehat{\mathbb{V}}(\hat{\theta})\}. \tag{11}$$

As we see, this result is very similar to the White (1982) test, but the power of the test is much higher. Both exact and asymptotic test statistics are denoted in the package as `gofPIOSTn` and `gofPIOSRn`, respectively.

Hybrid test

Many power studies including Genest et al. (2009) showed that no overall single optimal test exists for testing for copula models. Zhang et al. (2016) introduced a Hybrid test to combine the testing power of several tests. Having q different tests and the corresponding p-values, $p^{(1)}, \ldots, p^{(q)}$, the combined p-value is defined to be:

$$p^{hybrid} = \min\{q \cdot \min(p^{(1)}, \ldots, p^{(q)}), 1\}. \tag{12}$$

In Zhang et al. (2016), it is shown that the consistency of (12) is ensured as long as at least one of the q tests is consistent.

Bootstrapping test statistics

As the distribution of the test statistics is in most cases unknown, we perform a parametric bootstrap to receive the p-values. The necessary steps are described as follows:

Step 1. Generate bootstrap sample $\left\{\epsilon_i^{(m)}, i = 1, \ldots, n\right\}$ from copula $C(u_1, \ldots, u_d; \hat{\theta})$ under H_0 with $\hat{\theta}$ and estimated marginal distributions $\widetilde{F}$ obtained from original data;

Step 2. Based on $\left\{\epsilon_i^{(m)}, i = 1, \ldots, n\right\}$ from Step 1, estimate θ of the copula under H_0 and compute test statistics under consideration, say ;

Step 3. Repeat M-times Steps (1. – 2.) and obtain M statistics $T_n^m, m = 1, \ldots, M$;

Step 4. Compute an empirical p-value as $p_e = M^{-1} \sum_{m=1}^{M} \mathbf{1}\{|T_n^m| \geq |T_n|\}$ with T_n being the test statistics estimated from original dataset.

Depending on the different tests, variants of the described steps have to be performed. For example, the Kernel density estimation test of Scaillet (2007) described in Section 2.3.5 relies on a double bootstrapping procedure, in which for the computation of each test statistic, T_n and T_n^m in the steps above, an additional bootstrapping is utilized. Thus, the double bootstrapping approach consists of one bootstrap to calculate the p-value from a given test statistic and a second bootstrap to calculate the test statistic from an estimated copula. For further details, we refer to Scaillet (2007). Both bootstrapping procedures can be controlled via the arguments `M` and `MJ`, respectively.

Functionality of the R-package

The core of the **gofCopula** package is the function `gof`, which computes different tests for different copulae for a given dataset, based on the user's choice.

```
R> library("gofCopula")
R> data("IndexReturns2D", package = "gofCopula")
R> system.time(result <- gof(IndexReturns2D, M = 100, seed.active = 1))


The margins will be estimated as: ranks
normal copula
Test gofCvM is running
Test gofKendallCvM is running
Test gofKendallKS is running
Test gofKernel is running
Test gofKS is running

t copula
Test gofCvM is running
Test gofKendallCvM is running
Test gofKendallKS is running
Test gofKernel is running
Test gofKS is running

clayton copula
Test gofCvM is running
Test gofKendallCvM is running
Test gofKendallKS is running
Test gofKernel is running
Test gofKS is running

gumbel copula
Test gofCvM is running
Test gofKendallCvM is running
Test gofKendallKS is running
Test gofKernel is running
Test gofKS is running

frank copula
Test gofCvM is running
Test gofKendallCvM is running
Test gofKendallKS is running
Test gofKernel is running
Test gofKS is running

joe copula
Test gofCvM is running
Test gofKendallCvM is running
Test gofKendallKS is running
Test gofKernel is running
Test gofKS is running

amh copula
The copula amh is excluded from the analysis since the parameters do not fit its
parameter space. See warnings and manual for more details.

galambos copula
Test gofCvM is running
Progress: [===>-----------------------------------]  15% | time left:  3s
...

   user  system  elapsed
 629.26    1.08   628.94
```

```
Warnings:
...
```

gof considers all 13 available copula models if no copulae or tests are specified. If a copula is unsuitable in the sense that the estimated parameter is at the boundary of the parameter space, the copula is automatically excluded, and the user is informed via a console statement (see above) and additional warnings. In the given example, this is the case for the AMH, tawn, and FGM copulae because the used `IndexReturns2D` dataset exhibits an estimated Kendall's $\hat{\tau} = 0.611$, which none of these three copulae can model adequately; see Table A.1. The object `result` is of class `"gofCOP"` and has length 10, which is the number of copulae used in testing (here: 13) minus the ones excluded during calculation (here: 3). Following Table 1, five tests are available for all of these copula models in $d = 2$, and these are used in the given function call.

If the user specifies copulae, tests, or both, the intersection of possible tests and copulae following Table 1 is considered. For example, if `copula = c("normal","tawn")` is specified, the function calculates the five tests which are implemented for both copulae (assuming $d = 2$). If, on the other hand, `tests = c("gofKernel","gofArchmSnB")` is selected, the five Archimedean copulae implemented for both tests are computed. In the case when both copulae and tests are defined, the function provides results for the possible combinations. During the calculation, the user is informed about the computation progress by statements about the running test and copula. Furthermore, a progress bar indicates the percentage of progress for this specific test as well as a dynamically updated estimated remaining time.

```
R> result$normal

$method
[1] "Parametric bootstrap goodness-of-fit test with hybrid test and normal copula"

$copula
[1] "normal"

$margins
[1] "ranks"

$param.margins
list()

$theta
          [,1]
[1,] 0.8347428

$df
NULL

$res.tests
                    p.value   test statistic
CvM                    1.00       0.01520542
KendallCvM             0.41       0.06286712
KendallKS              0.11       0.80800000
Kernel                 0.39       0.56012429
KS                     1.00       0.31392428
hybrid(1, 2)           0.82               NA
hybrid(1, 3)           0.22               NA
...
```

The first element of `result` provides results for the normal copula. Note that in the field `res.tests` the hybrid tests, starting after the individual ones, contain numbers in brackets indicating which tests are considered for this hybrid. Thus, `hybrid(1,2)` means that this is the hybrid of `CvM` and `KendallCvM` tests. The p-value 0.82 in testing for normality is obtained following formula (12) and therefore is $\min\{2 \cdot \min(1.00, 0.41), 1\} = 0.82$. To access the rotated versions of the copulae, one can set, for example, `copula = c("clayton","gumbel")` together with `flip = c(0,180)`, which would test for the Clayton copula and the 180 degrees rotated Gumbel copula.

gofCOP class

Objects of class "gofCOP" are generated by the function gof or a single test function like, e.g., gofPIOSTn. They consist of different sub-elements - one for each copula - as, e.g., result$normal in the given example. These sub-elements are lists of length seven and contain the estimation and test results for the specific copula. They present in the field method a description of the test scenario. The field margins lists the defined marginal distribution that can also be a vector of distributions where each element is applied to the respective data column, whereas param.margins returns the estimates of the parameters of the marginal distributions if a parametric approach was specified. Field theta contains the ML estimate of the copula parameter. In case of the t- and t-EV-copulae, the section df is the estimated number of degrees of freedom for the copula. The values of these parameters are identical for all the tests. In res.tests, the p-values and test statistics (only for individual tests) are given for each of the executed tests. Each row corresponds to one test from the individual to the hybrid tests. p-values of all the individual tests are computed via the bootstrap method described in Section 2.3.9. The number of bootstrap samples M can be adjusted via the parameter M.

Plotting gofCOP class objects

"gofCOP" objects can be called by a generic plot function allowing the user to get the p-values of the single, and the hybrid tests visualized in a pirateplot of the R-package **yarrr**. It enables the user to select which copulae and hybrid testing sizes are desired for plotting. The remaining customization options are equal to those of the function pirateplot from the package **yarrr**, except for the arguments formula, data, sortx, xaxt, xlim, ylim, and ylab.

```
R> plot(result, copula = c("clayton", "joe", "plackett"), hybrid = c(1, 3, 5))
```

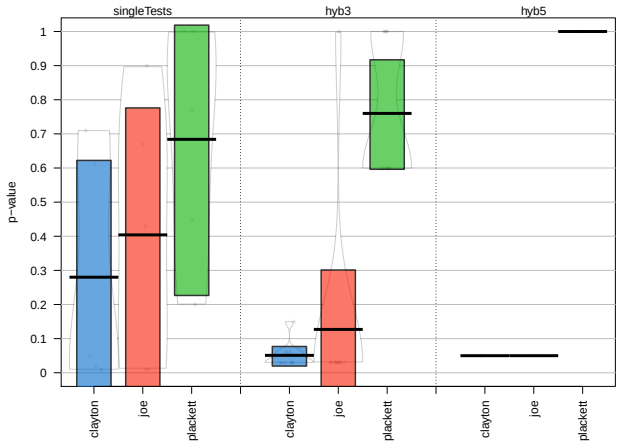


Figure 1: Resulting p-values of different hybrid tests for the Clayton, Joe, and Plackett copula visualized in a pirateplot.

Specifying hybrid = c(1,3,5) means that the p-values of the single tests (column singleTests in Figure 1), the p-values of hybrid tests of size three (column hyb3), and size five (column hyb5) should be plotted, separated by selected copulae. For example, we focus on the column hyb3 for the Plackett copula. It contains information of all hybrid tests, which include three single tests for the Plackett copula. In this case, we can see that the mean of these tests is approximately 0.76, as shown by the thick horizontal line. All test p-values are shown by light-grey points in the column, indicating the heterogeneity of the tests ranging from 0.6 to 1. Finally, the green bar around the mean line is a Bayesian highest density interval, which provides the user, together with the shown density estimate in the grey continuous lines, further information about the distribution of the p-values. For more details on the pirateplot and its customization options, we refer to Phillips (2017).

Fire-and-forget

The R-package **gofCopula** includes all the discussed tests in Section 2.3. For each of the tests, a separate function is implemented with a variety of arguments. We give shortly the most important arguments all the tests share before we go into details about the structure of the package.

- copula: The copula to test for. Possible options depend on the test and dimension.

- `x`: A matrix containing the data with rows being observations and columns being variables.
- `M` (default: 1000): Number of bootstrapping loops.
- `param` (default: 0.5): The copula parameter to use if it shall not be estimated. In case of the Gumbel copula, the default value is set to 1.5.
- `param.est` (default: `TRUE`): Boolean. `TRUE` means that `param` will be estimated.
- `margins` (default: `ranks`): Specifies which estimation method shall be used. The default `ranks` stands for formula (4), which is the standard approach to convert data. Alternatively, the following distributions can be specified: `beta`, `cauchy`, Chi-squared (`chisq`), `f`, `gamma`, Log normal (`lnorm`), Normal (`norm`), `t`, `weibull`, Exponential (`exp`).
- `flip` (default: 0): The parameter to rotate the copula by 90, 180, 270 degrees clockwise. Only applicable for bivariate copula.
- `seed.active` (default: `NULL`): Sets the seeds for the bootstrapping procedure. It has to be either an integer or a vector of `M+1` integers. If an integer is provided, the seeds for the bootstrap samples will be simulated based on it. If `M+1` seeds are given, these are used in the bootstrapping procedure. In the default case (`seed.active = NULL`), R generates the seeds from the computer runtime.
- `processes` (default: 1): The number of parallel processes which are performed to speed up the bootstrapping. Should not be larger than the number of logical processors.

The package is coded as a *fire-and-forget* package. Each of the single tests just requires the input of a dataset `x` and a `copula` to test for. All the other function parameters have reasonable default values such that quick first results can be achieved easily. The calculation steps of each GoF test function are the following:

- *Estimation of margins*: At first, the function transforms the data nonparametrically or parametrically to $\mathcal{U}[0,1]$; see Section 2.2. This transformation is performed automatically, and a console statement informs the user about the transformation.
- *Estimation of copula*: Afterward, the parameters of the copula model are estimated, so a two-stage estimation is applied; see Section 2.2. Since a full ML estimation is computationally demanding, canonical ML estimation or inference for margins is applied. In case the ML estimation fails, the package automatically changes to inversion of Kendall's tau (see Section 2.2), which guarantees a result. The user is informed about that switch by a warning message.
- *Bootstrapping*: Following the estimation of the copula parameters, the bootstrapping procedure will be performed, and the empirical p-value will be derived according to the test statistics in Section 2.3.9. Since the bootstrapping procedure can require a long computational time, it can pay out to parallelize the bootstrapping via the argument `processes`.

Hybrid testing and further functionality

Besides `gof` and the single tests, the package **gofCopula** offers additional functionality for the user. Next to descriptions, illustrative examples are provided, assuming the following was called beforehand:

```
R> library("gofCopula")
R> data("IndexReturns2D", package = "gofCopula")
R> (res <- gof(copula = "normal", x = IndexReturns2D, M = 10, seed.active = 1,
+              tests = c("gofPIOSRn", "gofCvM", "gofKernel")))

The margins will be estimated as: ranks
normal copula
Test gofPIOSRn is running
Test gofCvM is running
Test gofKernel is running
--------------------------------------------------------------------------------
Parametric bootstrap goodness-of-fit test with hybrid test and normal copula

Parameters:
theta.1 = 0.834742824340301

Tests results:
               p.value test statistic
PIOSRn             0.5    -0.11032857
```

```
Sn                  1.0     0.01520542
Kernel              0.3     0.56012429
hybrid(1, 2)        1.0             NA
hybrid(1, 3)        0.6             NA
hybrid(2, 3)        0.6             NA
hybrid(1, 2, 3)     0.9             NA

Please use the functions gofGetHybrid() and gofOutputHybrid() for
display of subsets of Hybrid tests. To access the results, please
obtain them from the structure of the gofCOP object.
```

The functions are:

- `gofCustomTest`: This function has - next to the standard arguments listed in Section 2.4.3 - the argument `customTest`, a character string referencing one customized test loaded in the workspace. The function containing this test should have two arguments: a matrix named `x` for the dataset and a character string named `copula` for the copula to test for. The whole calculation process, including the estimation of the margins and the copula, the calculation of the test statistics, and the bootstrapping of the p-value, is performed for this customized test. The procedure is shown using the test statistics of the `gofCvM` test.

```
R> Testfunc = function(x, copula) {
+     C.theo = pCopula(x, copula = copula)
+        C.n = F.n(x, X = x)
+        CnK = sum((C.n - C.theo)^2)
+     return(CnK)
+ }

R> gofCustomTest(copula = "normal", x = IndexReturns2D, M = 10,
+                customTest = "Testfunc", seed.active = 1)

The margins will be estimated as: ranks
-------------------------------------------------------------------------------
Parametric bootstrap goodness-of-fit test with Testfunc test and normal copula

Parameters:
theta.1 = 0.834742824340301

Tests results:
           p.value test statistic
Testfunc         1     0.01520542
```

- `gofGetHybrid`: Allows calculating hybrid test p-values for given p-values from customized tests with an object of class `"gofCOP"` generated in the package. Through the combination of `gofCustomTest` and `gofGetHybrid`, the users are not limited to the implemented tests in the package and have the opportunity to include their own tests in the analysis. Note that the function `gofOutputHybrid` has slightly different but comparable functionality, which is the reason it is not separately shown.

```
R> gofGetHybrid(result = res, nsets = 5, p_values = c("MyTest" = 0.7,
+               "AnotherTest" = 0.3))

-------------------------------------------------------------------------------
Hybrid test p-values for given single tests.

Parameters:
theta.1 = 0.834742824340301

Tests results:
                      p.value
PIOSRn                    0.5
Sn                        1.0
Kernel                    0.3
MyTest                    0.7
AnotherTest               0.3
hybrid(1, 2, 3, 4, 5)     1.0
```

- `gofTest4Copula`: Returns for a given copula and a given dimension the list of applicable implemented tests.

```
R> gofTest4Copula("gumbel", d = 5)

 [1] "gofArchmChisq"      "gofArchmGamma"      "gofArchmSnB"
 [4] "gofArchmSnC"        "gofCustomTest"      "gofCvM"
 [7] "gofKendallCvM"      "gofKendallKS"       "gofKS"
[10] "gofRosenblattChisq" "gofRosenblattGamma" "gofRosenblattSnB"
[13] "gofRosenblattSnC"
```

- `gofCopula4Test`: Returns for a given test the list of applicable implemented copulae.

```
R> gofCopula4Test("gofPIOSTn")

 [1] "normal"   "t"        "clayton"  "gumbel"   "frank"    "joe"
 [7] "amh"      "galambos" "fgm"      "plackett"
```

- `gofCheckTime`: Estimates the time necessary to compute a selected single or group of GoF tests for a given number of bootstrapping rounds. This function uses an underlying regression model, so the results may vary from reality and also from the progress bar predictions. See Section 2.4.5.

```
R> gofCheckTime("normal", x = IndexReturns2D, tests = "gofRosenblattSnC",
+               M = 10000, seed.active = 1)

The margins will be estimated as: ranks
An estimate of the computational time is under derivation.
Depending on the tests chosen, dimensionality and complexity of the data, this
might take a while.
The computation will take approximately 0 d, 0 h, 10 min and 7 sec.
```

- `gofco`: In the case a copula is already estimated with the package **copula**, one can provide an object of class `"copula"` to this function, and the parameter estimates are taken from the respective object.

```
R> copObject = normalCopula(param = 0.8)
R> gofco(copObject, x = IndexReturns2D, M = 10, seed.active = 1,
+        tests = c("gofPIOSRn", "gofKernel"))

The margins will be estimated as: ranks
Test gofPIOSRn is running
Test gofKernel is running
-------------------------------------------------------------------------------
Parametric bootstrap goodness-of-fit test with hybrid test and normal copula

Parameters:
theta.1 = 0.8

Tests results:
             p.value test statistic
PIOSRn           0.9    -0.03641543
Kernel           0.2     0.57115224
hybrid(1, 2)     0.4             NA

Please use the functions gofGetHybrid() and gofOutputHybrid() for
display of subsets of Hybrid tests. To access the results, please
obtain them from the structure of the gofCOP object.
```

Computational time and parallelization

One of the main drivers of long computation times is the high number of bootstrapping loops to achieve an asymptotically reliable result. As mentioned in Section 2.4.4, the build-in function `gofCheckTime` allows estimating the necessary computation time for a given test, copula, dataset, and number of bootstrapping rounds. Since different machines may have highly varying computation times for tests, the function relies on a regression using the number of bootstrapping loops as the independent variable. To ensure that the linear model is a valid assumption, we investigated the case using the functions `gofKendallKS` and `gofKernel`; see Section 2.5.2 for the results.

Enabling parallelization of the bootstrapping is necessary for computationally demanding tests as `gofPIOSTn` where, e.g., the computation for a dataset of 500 observations and 1000 bootstrapping loops for the t-copula can take, depending on the engine, up to several hours. However, even for tests with faster computation time, parallelization is useful given the sample size and the number of bootstrapping loops is sufficiently high. This is shown in Table 2 in the form of a comparison between the computation times of five tests for five copulas without and with parallelization on four cores. The dataset contained $n = 500$ observations randomly generated from a bivariate standard normal distribution, and the number of bootstrapping loops was set to $M = 1000$.

Test	normal	t	clayton	gumbel	frank
#processes = 1:					
gofKendallCvM	200.33	530.22	166.85	249.86	167.81
gofKendallKS	115.43	450.50	100.69	172.26	88.83
gofPIOSRn	84.90	433.32	67.20	148.09	50.98
gofRosenblattChisq	75.79	431.02	73.09	143.94	55.97
gofRosenblattGamma	75.28	420.31	75.77	142.16	52.17
#processes = 4:					
gofKendallCvM	106.19	288.86	104.29	140.25	94.69
gofKendallKS	68.73	238.55	56.13	89.02	53.00
gofPIOSRn	48.81	235.35	47.37	83.71	33.66
gofRosenblattChisq	49.43	230.33	45.83	85.70	35.47
gofRosenblattGamma	47.45	234.50	48.41	82.04	37.32

Table 2: Computation times in seconds without and with parallelization using an Intel Core i7-4712MQ CPU with 2.3 GHz on a 64-Bit Windows 10 system.

Simulations

Empirical process of using the package

We would like to illustrate the power of the GoF tests with the use of the **gofCopula** package. In practice, one is often confronted with realizations of random variables for which an adequate copula model has to be found, as, e.g., in the two examples from the financial domain provided in Section 2.6. To illustrate the procedure, we focus in this Section on an easy replicable example. For this purpose, we start by simulating $n = 1000$ observations from a Clayton copula with Kendall's $\tau = 0.5$.

```
R> library("gofCopula")
R> param = iTau(copula = claytonCopula(), tau = 0.5)
R> n = 1000; set.seed(1)
R> x = rCopula(n = n, copula = claytonCopula(param = param))
```

To gain a better understanding of the data, Figure 2 shows the simulated data with different margins, reflecting the typical shape of the Clayton copula.

```
R> par(mfrow = c(1,2))
R> u = cbind(ecdf(x[,1])(x[,1]), ecdf(x[,2])(x[,2])) * n / (n + 1)
R> plot(u, col = "blue3", pch = 19, cex.lab = 1.25,
+       xlab = expression(u[1]), ylab = expression(u[2]))
R> plot(qnorm(u), col = "blue3", pch = 19, cex.lab = 1.25,
```

```
+       xlab = expression(x[1]), ylab = expression(x[2]))
R> par(mfrow = c(1,1))
```

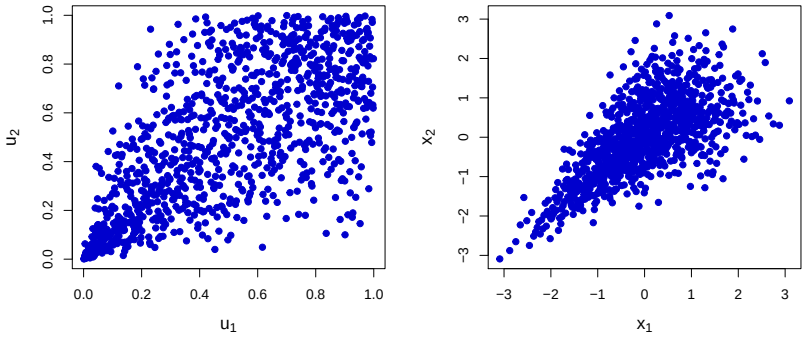


Figure 2: $n = 1000$ observations sampled from a Clayton copula with $\tau = 0.5$. Margins are transformed using ranks on the left plot and are standard normal on the right plot.

To make an adequate decision on which copula should be used in the respective modeling task, the GoF testing should involve more than looking at one or two test results and should consider a reasonable amount of potential copula models. We structure our procedure by testing for three groups of copulae separately: Elliptical, Archimedean, and extreme value (EV) copulae. In the function call, we select the FGM and Plackett copulae together with the EV category, although they do not belong to any of the three categories. Elliptical copulae include the normal and t-copula, while the Clayton, Gumbel, Frank, Joe, and AMH copulae are the Archimedean ones. Galambos, Husler-Reiss, Tawn, and t-EV belong to the EV category. Notice that this categorization could be modified, as, e.g., the Gumbel copula is also an EV copula. However, the given approach offers not only a logical structuring of the modeling task, but leads to using a close to maximal number of tests via only three function calls. The bootstrap parameters were set to $M = 100$ and $MJ = 1000$. As this task is computationally demanding, we set the argument `processes = 7` to speed up the calculation using parallelization on 7 cores. We use the default `margins = "ranks"` and set `seed.active = 10` for reproducibility.

```
R> cop_1 = gof(x = x, M = 100, MJ = 1000, processes = 7, seed.active = 10,
+              copula = c("normal", "t"))
R> cop_2 = gof(x = x, M = 100, MJ = 1000, processes = 7, seed.active = 10,
+              copula = c("clayton", "gumbel", "frank", "joe", "amh"))
R> cop_3 = gof(x = x, M = 100, MJ = 1000, processes = 7, seed.active = 10,
+              copula = c("galambos", "huslerReiss", "tawn", "tev", "fgm", "plackett"))
```

To evaluate the gained objects of class `"gofCOP"`, one can manually inspect the resulting p-values and look closer at the performances and differences between the single tests and the corresponding hybrids. However, the easiest and most informative way is to visualize the p-values, which is done using the `plot` function.

```
R> plot(cop_1)
```

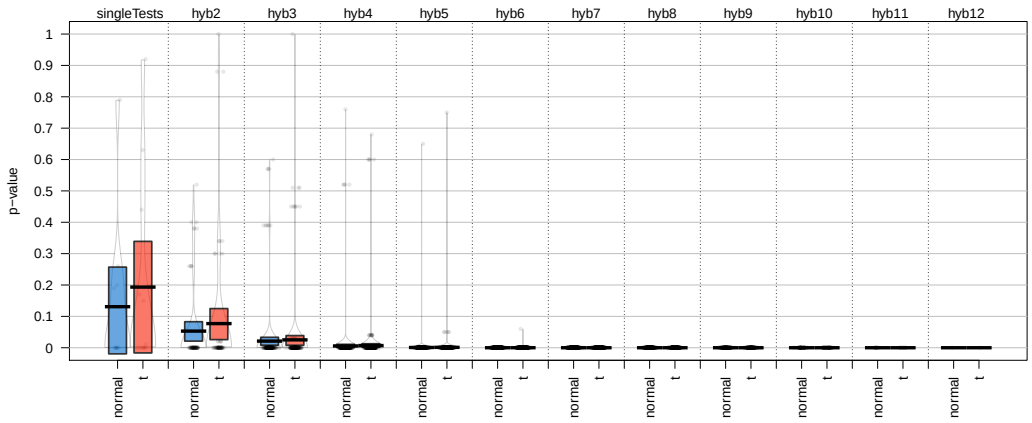


Figure 3: p-values of the hybrid tests for the data from Figure 2 for elliptical copulae.

```
R> plot(cop_2)
```

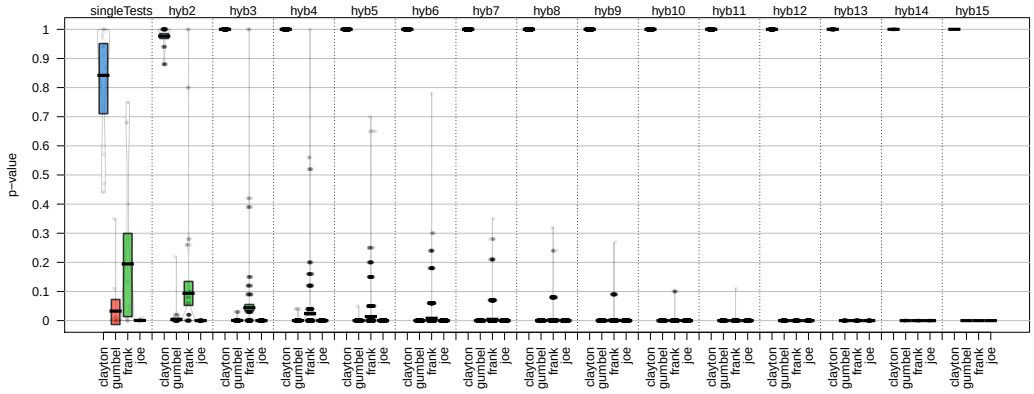


Figure 4: p-values of the hybrid tests for the data from Figure 2 for Archimedean copulae.

```
R> plot(cop_3)
```

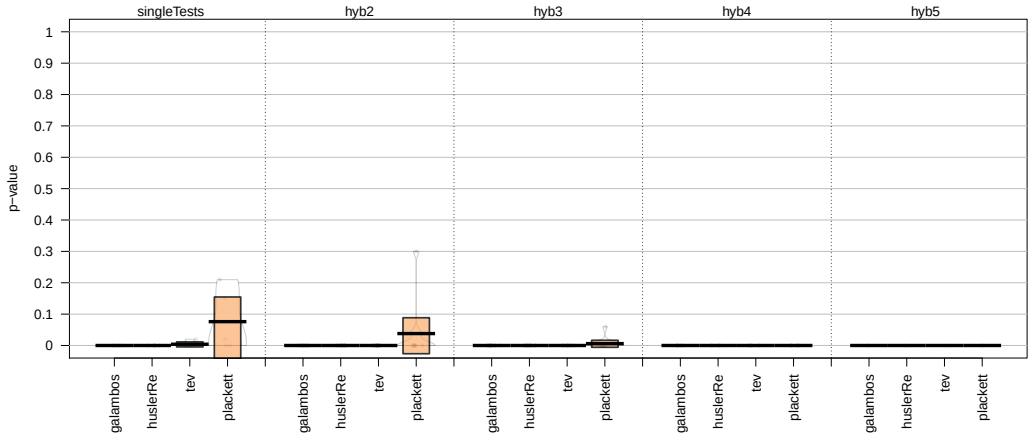


Figure 5: p-values of the hybrid tests for the data from Figure 2 for EV, FGM, and Plackett copulae.

Interpreting Figures 3, 4, and 5 clearly shows the ability of the tests to detect the true copula. The column `singleTests` in Figure 4 indicates that the Clayton copula is appropriate. The decision is supported by the higher-order hybrid tests, as all p-values except for the Clayton copula become 0, strongly rejecting the H_0-hypothesis in these cases. Notice that similar to the introductory example in Section 2.4, the AMG, Tawn, and FGM are automatically excluded, which is why they do not appear in the plots. Having such a result at hand, the user can proceed with the modeling task with the selected copula.

Validating the model for the time estimation

In the next step, we validate the assumption of using a linear model for estimating the computation time in `gofCheckTime`. We have chosen the `gofKendallKS` test as a representative for the group of single bootstrapping tests and `gofKernel`, as the test having a double bootstrapping procedure. Both tests are available for all copulae in the bivariate case. For `gofKendallKS`, we measured the computation times for 12 copulae, varying numbers of bootstrap loops (M) and sample sizes (n) of the underlying dataset, which is simulated from a normal copula with Kendall's $\tau = 0.1$. This value is selected because it falls within the attainable interval of Kendall's τ for all copulae, see Table A.1. For `gofKernel`, we fixed $n = 100$ and investigated the situation for 12 copulae, different M, and different sample sizes MJ of the internal bootstrap. The results are shown in Figures 6, 7, and 8. The t-EV copula is not included in these illustrations due to its tremendous computation time, which can exceed the one of the t-Copula even for small sample sizes by a factor of 10 and higher. However, similar properties in the behavior of the computation time depending on the number of bootstrapping loops can be found. All calculations were performed without parallelization using an Intel Core i7-4712MQ CPU with 2.3 GHz on a 64-Bit Windows 10 system.

For the `gofKendallKS` test, the computation time increases linearly with the number of bootstrapping loops M, while the t-Copula is generally the most time-demanding of the considered copulae. This holds for all the analyzed sample sizes. A similar observation can be made for the `gofKernel` test. Here, a rapid increase in computation time is expected if both M and MJ increase. However, following Figures 7 and 8, this is not the case, and a linear dependency is justifiable. Therefore, the package implements for `gofKernel` a linear model with M and MJ being independent variables.

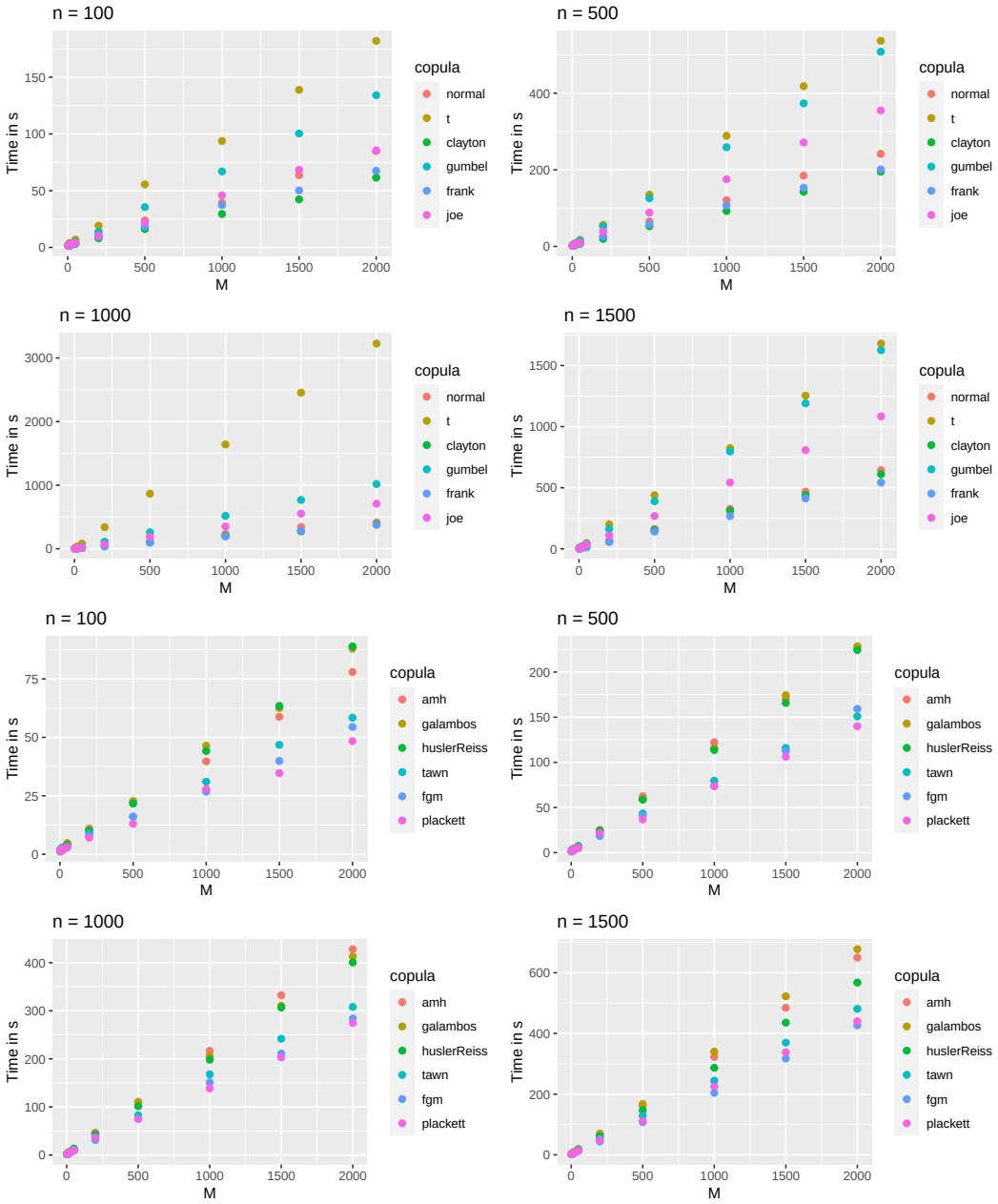


Figure 6: Computation times of `gofKendallKS` for different copulae, sample sizes n, and number of bootstrapping loops M.

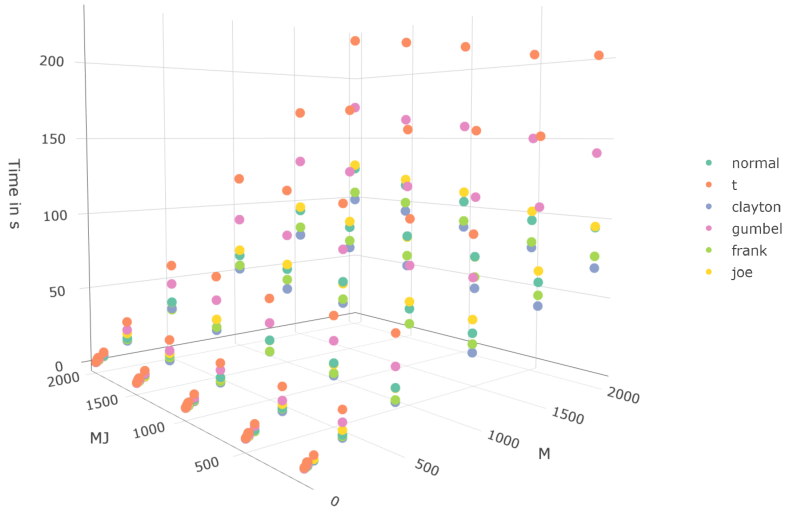


Figure 7: Computation times of `gofKernel` for different copulae, number of bootstrapping loops M, and internal bootstrap sample size MJ.

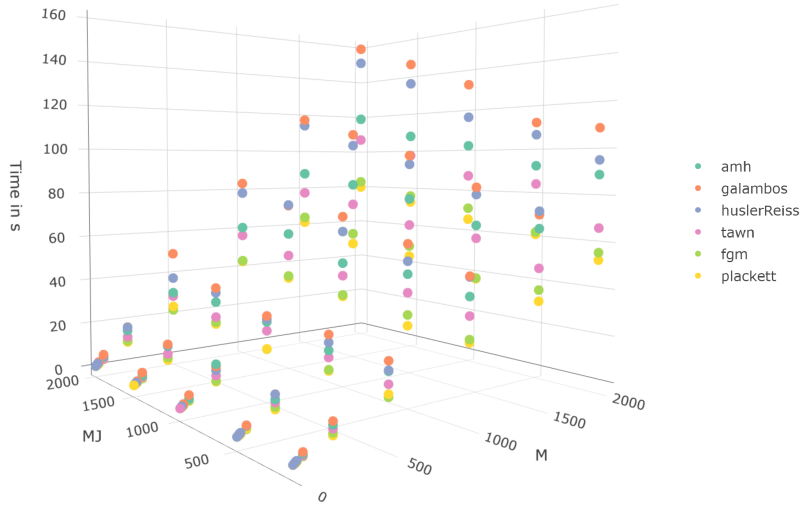


Figure 8: Computation times of `gofKernel` for different copulae, number of bootstrapping loops M, and internal bootstrap sample size MJ.

Application

Cryptocurrency market

We intend to demonstrate the functionality of the **gofCopula** package and show the empirical procedure as described in Section 2.5.1 on a real-world example from the market of cryptocurrencies. To account for the relevant steps in a realistic application study, we split the procedure into *Data Investigation* and *Goodness-of-Fit testing*.

Data investigation

We have chosen Bitcoin (BTC) and Litecoin (LTC) for our analysis. The objective is to detect which copula is appropriate to model the dependence structure between BTC-LTC and check whether the copula changes over the years. For that purpose, we use the volatility-adjusted log-returns of the currencies in the time span from 2015 to 2018. The volatility correction was performed by fitting a

GARCH(1,1) process to each time series for each year separately in order to extract their standardized residuals. These are included in the package as `CryptoCurrencies`, whereas each element of the list contains the data for a particular year. In order to gain a visual impression beforehand, we plotted the data with margins transformed to standard normal, leading to Figure 9. A strong dependency between both cryptocurrencies is visible, especially in the year 2018. Based on these residual diagrams, it is possible to take a guess which copula is the most adequate for the given situation. For 2015, one could possibly argue that the elliptical shape of a normal copula is present, while in 2016 and 2018, the shapes are more similar to the one of a t-Copula. Finally, for the year 2017, Figure 9 shows a comparable plot to Figure 2 from the simulated example in Section 2.5.1, indicating a Clayton copula might be present. However, these visual impressions are to a certain degree subjective and need to be backed up by the GoF tests. Ideally, the test results would match our plot-based guesses.

```
R> library("gofCopula")
R> data("CryptoCurrencies", package = "gofCopula")
R> par(mfrow = c(2,2))
R> years = as.character(2015:2018)

R> for(i in years){
+     x1 = CryptoCurrencies[[i]][,1]
+     x2 = CryptoCurrencies[[i]][,2]
+      n = length(x1)
+     plot(qnorm(cbind(ecdf(x1)(x1), ecdf(x2)(x2)) * n / (n + 1)), col = "blue3",
+          pch = 19, cex.lab = 1.25, main = i, xlab = "Bitcoin", ylab = "Litecoin")
+  }
R> par(mfrow = c(1,1))
```

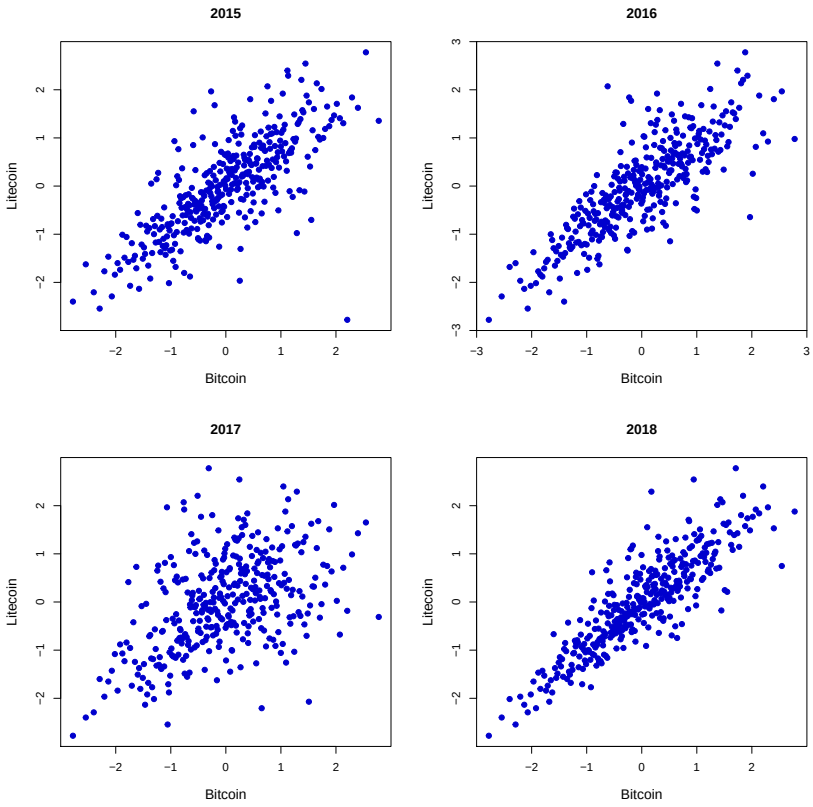


Figure 9: Residual plots for BTC-LTC with margins transformed to standard normal.

Goodness-of-fit testing

In this example, the focus in testing is on the most popular copula models in practice: normal, t, Clayton, Gumbel, and Frank copulae. To get the highest testing power, we include all tests, which are available for all five copulae. Thus, following Table 1, each test in the package except the ones based on the transformation for Archimedean copulae (see Section 2.3.4) is computed. Additionally, all

possible hybrid tests are considered. We use the function `gof` while setting the bootstrap parameters $M = 100$ and $MJ = 1000$. We specify the number of cores for the parallelization to `processes = 7`. For replicability, we set `seed.active = 1:101` and apply the non-parametric margin transformation by default.

```
R> copulae = c("normal", "t", "clayton", "gumbel", "frank")

R> BTC_LTC_15 = gof(x = CryptoCurrencies[["2015"]], copula = copulae, M = 100,
+                  MJ = 1000, processes = 7, seed.active = 1:101)
R> BTC_LTC_16 = gof(x = CryptoCurrencies[["2016"]], copula = copulae, M = 100,
+                  MJ = 1000, processes = 7, seed.active = 1:101)
R> BTC_LTC_17 = gof(x = CryptoCurrencies[["2017"]], copula = copulae, M = 100,
+                  MJ = 1000, processes = 7, seed.active = 1:101)
R> BTC_LTC_18 = gof(x = CryptoCurrencies[["2018"]], copula = copulae, M = 100,
+                  MJ = 1000, processes = 7, seed.active = 1:101)
```

After finishing the calculations, we proceed by plotting the received objects of class `"gofCOP"`. For a detailed explanation about the information contained in the **gofCopula** pirateplots, please see Section 2.4.2 and Phillips (2017).

```
R> plot(BTC_LTC_15)
```

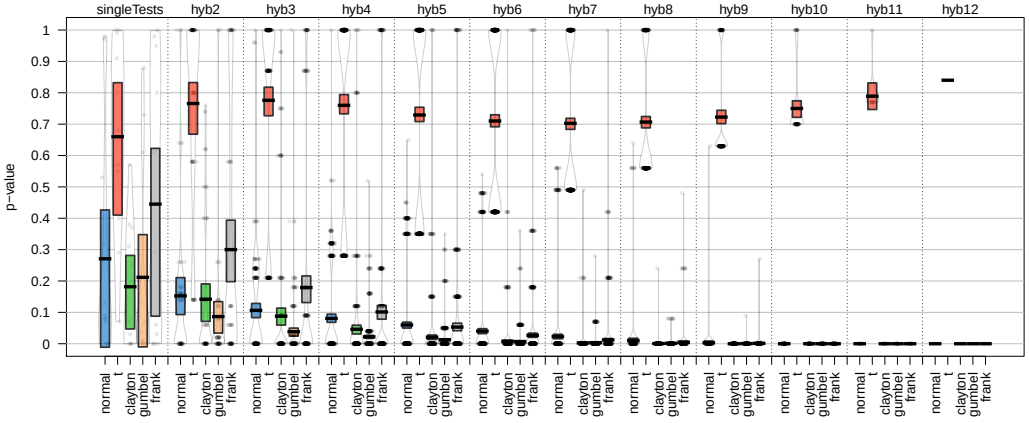


Figure 10: p-values of the single and hybrid tests for BTC-LTC in the year 2015.

```
R> plot(BTC_LTC_16)
```

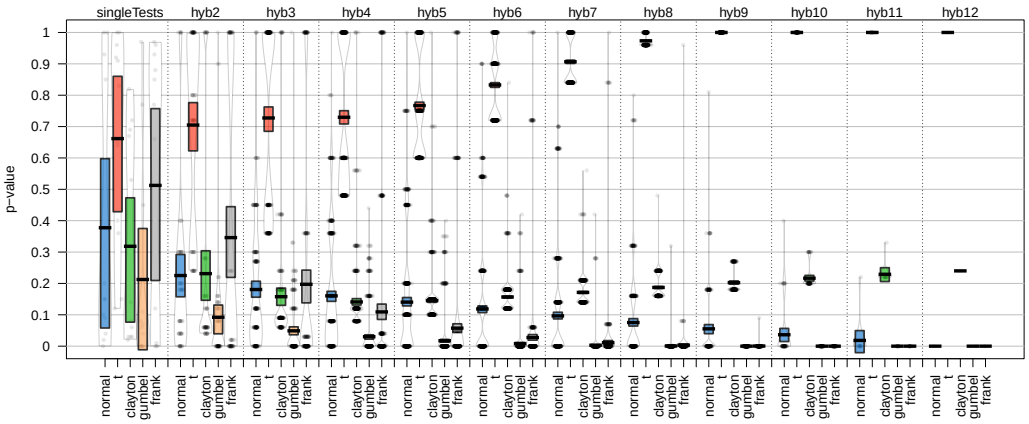


Figure 11: p-values of the single and hybrid tests for BTC-LTC in the year 2016.

```
R> plot(BTC_LTC_17)
```

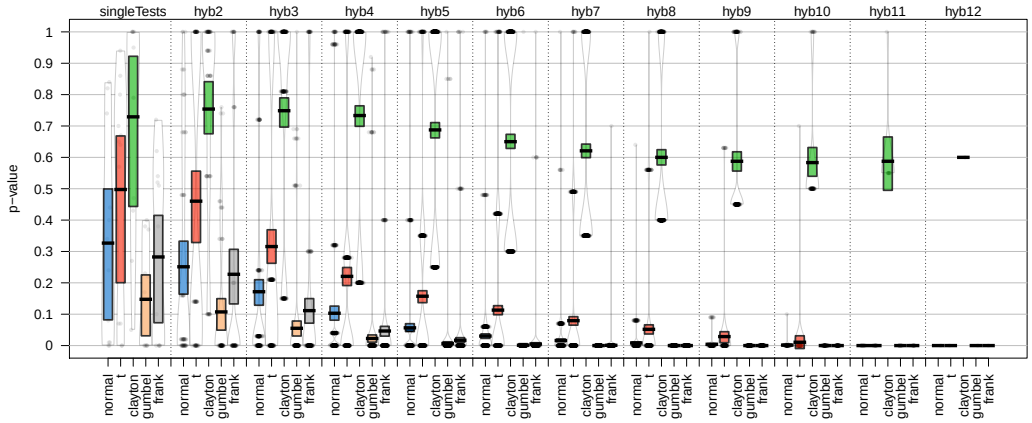


Figure 12: *p*-values of the single and hybrid tests for BTC-LTC in the year 2017.

```
R> plot(BTC_LTC_18)
```

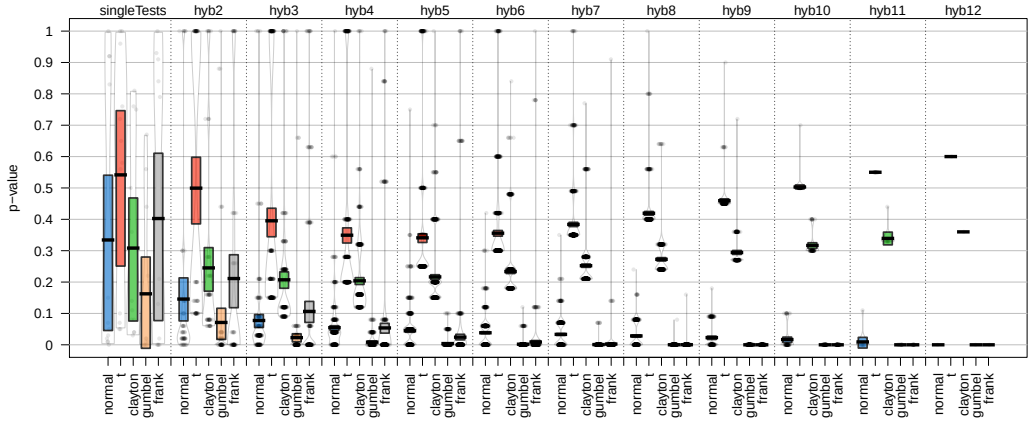


Figure 13: *p*-values of the single and hybrid tests for BTC-LTC in the year 2018.

Figures 10 to 13 show the resulting *p*-values in the form of `"gofCOP"`-plots for all the considered years. Following the usual approach in practice, we select the copula corresponding to the highest *p*-values. For the year 2015, we see that the *t*-copula is favored by the tests, as all remaining *p*-values become 0 with increasing hybrid testing size; see Figure 10. This rejects our initial visual guess that a normal copula might be appropriate. Continuing with 2016, we see our visual opinion solidified, as the plot suggests using a *t*-copula to capture the dependence structure. We can even see that the *p*-values converge to 1 for the *t*-copula when we consider the hybrid testing orders 9, 10, 11, and 12 . Figure 12 is in line with our visual impression as well. We see that the tests favor a Clayton copula, while the other copula models are rejected by the higher-order hybrid tests. Finally, for 2018 Figure 13 gives for the *t*-copula the highest *p*-values, although the difference to the *p*-values of the Clayton copula is not too large. Therefore, in three out of four years, the results from **gofCopula** matched our visual impressions from the residual plots.

Summarizing, the following conclusions can be drawn from this analysis:

- Generally, the dominant copulae in describing the dependency between the volatility-adjusted log-returns of BTC-LTC is the *t*-copula. Following the test results, the year 2017 is an exception, as the dependence structure shifted towards a Clayton copula. This observation reflects the change in the market during the year 2017, as many investors got attracted by cryptocurrencies in this phase. Due to the developed hype, both the prices of BTC and LTC drastically increased, resulting in a modified underlying dependence structure between the two currencies.
- The hybrid tests are able to stabilize the results of the single tests, as they clearly selected for 2015, 2016, and 2018 the *t*-copula and in 2017 the Clayton copula. Therefore it is recommendable

to take hybrid tests into account in order to use the package adequately and get the highest testing power.

Stock return data

As a second real-world example, we analyze the volatility-adjusted stock log-returns of Citigroup (C) and the Bank of America (BoA) in the time span from 2004 to 2012. The procedure is, again, splitted into *Data Investigation* and *Goodness-of-Fit testing*.

Data investigation

This data was analyzed by Zhang et al. (2016), and we are expanding their procedure and consider the same copulae and tests as in the example in Section 2.6.1. The volatility correction was performed similarly in terms of fitting a GARCH(1,1) process, and the resulting data is included in **gofCopula** in the list `Banks`. Note that in this section, we focus on the years 2004 and 2007, while the results of the other years are given in Appendix B. We start by visualizing the residuals with margins transformed to standard normal.

```
R> library("gofCopula")
R> data("Banks", package = "gofCopula")
R> par(mfrow = c(1,2))

R> for(i in c("2004", "2007")){
+     x1 = Banks[[i]][,1]
+     x2 = Banks[[i]][,2]
+      n = length(x1)
+     plot(qnorm(cbind(ecdf(x1)(x1), ecdf(x2)(x2)) * n / (n + 1)), col = "blue3",
+     pch = 19, cex.lab = 1.25, main = i, xlab = "Citigroup", ylab = "Bank of America")
+  }
R> par(mfrow = c(1,1))
```

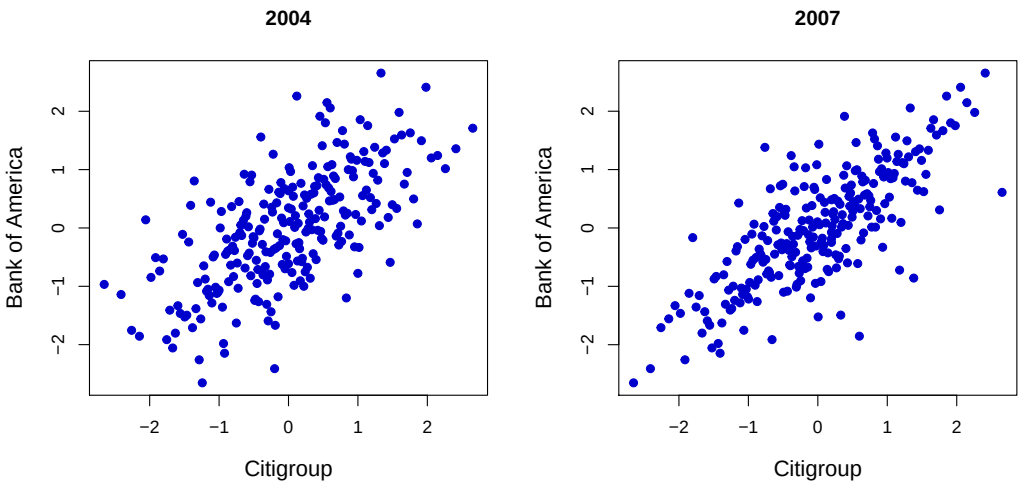


Figure 14: Residual plots for C/BoA in 2004 and 2007 with standard normal margins.

Analyzing the shape of the data in Figure 14 for 2004, one may argue that the elliptical normal copula is present, while in 2007, a t-copula is possibly more appropriate. To check these assumptions, we proceed with the GoF testing.

Goodness-of-fit testing

We set `M = 100` and `MJ = 1000` as bootstrap parameters, parallelize via `processes = 7`, and set `seed.active = 1:101` for reproducibility. Further, we implicitly keep the default `margins = "ranks"` to perform the margin transformation nonparametrically.

```
R> copulae = c("normal", "t", "clayton", "gumbel", "frank")

R> C_BoA_04 = gof(x = Banks[["2004"]], copula = copulae, M = 100, MJ = 1000,
+                 processes = 7, seed.active = 1:101)
R> C_BoA_07 = gof(x = Banks[["2007"]], copula = copulae, M = 100, MJ = 1000,
+                 processes = 7, seed.active = 1:101)
```

Following these calculations, we continue to plot the results, leading to Figures 15 and 16. For a detailed explanation about the information contained in the **gofCopula** pirateplots, please see Section 2.4.2 and Phillips (2017). Interpreting these "gofCOP"-plots of the p-values, the tests propose for 2004 indeed a normal copula (and a Frank one, which is radially symmetric), although a t-copula is a valid assumption as well. Compared to 2004, the p-values for the normal copula definitely decreased in 2007 and converged slowly to 0 with increasing hybrid testing size. The decision goes clearly in favor of the t-copula, which is in line with our original guess. Evaluating the results from Appendix B leads to similar conclusions as in Section 2.6.1. The hybrid tests are relatively stable and match in the majority of the cases the visual impressions from the residual plots. The proper copula seems to be the t-copula in most of the years, although in 2004 and 2009, the normal copula is a reasonable assumption. The hybrid tests are able to stabilize the selection of the copula.

```
R> plot(C_BoA_04)
```

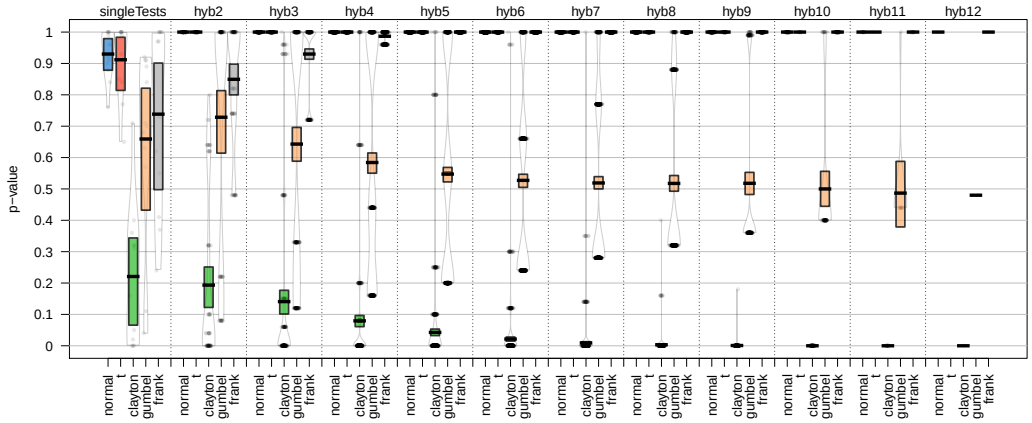


Figure 15: p-values of the C/BoA data for 2004. The column hyb12 of the t-copula is empty, as the p-value of the test gofWhite could not be computed due to instability in the test statistics. For a detailed description of this phenomenon, see Nagler et al. (2019).

```
R> plot(C_BoA_07)
```

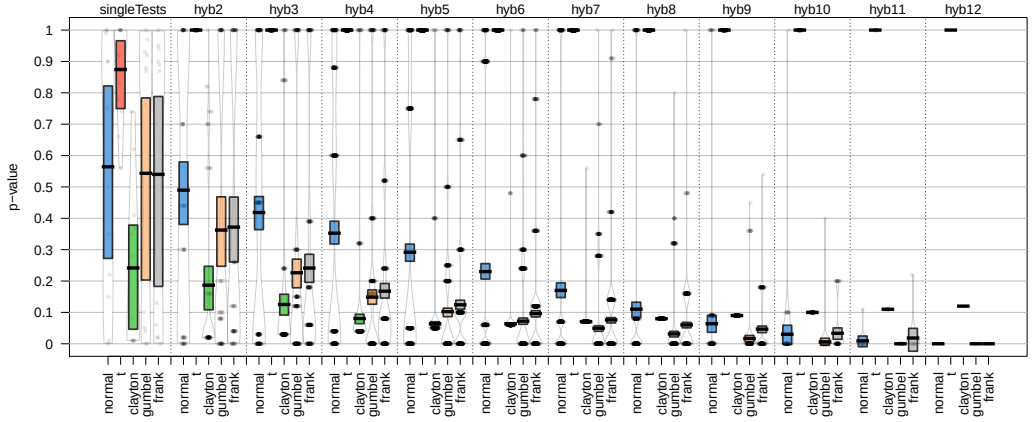


Figure 16: p-values of the C/BoA data for 2007.

Conclusion

This paper introduces a **gofCopula** package that provides maximum flexibility in performing statistical Goodness-of-Fit tests for copulae. The package provides an interface for 16 most popular GoF tests for 13 copulae with automatic estimation of margins via different techniques. The user is not limited to the implemented tests as self-defined test statistics functions can be easily embedded via a function provided in the package. As the computation of p-values relies on a parametric bootstrap, efficient and user-friendly parallelization is available. During the bootstrapping procedure, all tests inform the user about the progress of the calculations as well as the estimated time until the results are available. Additionally, **gofCopula** allows for the replication of said results. The package offers intelligible and interpretable visualization of the results of the hybrid tests that strengthen the overall test power. The flexibility and the usefulness of the tests are shown via a simulation and two empirical studies in economic sciences. In a nutshell, the broad range of tests, the comprehensive combination of methods, and an informative user-interface make **gofCopula** a fire-and-forget package providing flexibility in testing for the proper copula.

Acknowledgements

The authors are grateful to Shulin Zhang, Qian Zhou, Peter Song, and Sören Pannier for helpful discussions and to Oliver Scaillet for the code of the version of his test used in Zhang et al. (2016) that is being adapted in this package. Financial support from NUS FRC grant R-146-000-298-114 "Augmented machine learning and network analysis with applications to cryptocurrencies and blockchains" as well as CityU Start-Up Grant 7200680 "Textual Analysis in Digital Markets" is gratefully acknowledged by Simon Trimborn. Ostap Okhrin thankfully received financial support from RFBR, project number 20-04-60158.

Bibliography

P. Barbe, C. Genest, K. Ghoudi, and B. Rémillard. On Kendalls's process. *Journal of Multivariate Analysis*, 58:197–229, 1996. URL https://doi.org/10.1006/jmva.1996.0048. [p471]

W. Breymann, A. Dias, and P. Embrechts. Dependence structures for multivariate high-frequency data in finance. *Quantitative Finance*, 1:1–14, 2003. URL https://doi.org/10.1080/713666155. [p472]

S. X. Chen and T. Huang. Nonparametric estimation of copula functions for dependence modeling. *The Canadian Journal of Statistics*, 35(2):265–282, 2007. URL https://doi.org/10.1002/cjs.5550350205. [p470]

X. Chen and Y. Fan. Estimation and model selection of semiparametric copula-based multivariate dynamic models under copula misspecification. *Journal of Econometrics*, 135(1–2):125–154, 2006. URL https://doi.org/10.1016/j.jeconom.2005.07.027. [p470]

X. Chen, Y. Fan, and V. Tsyrennikov. Efficient estimation of semiparametric multivariate copula models. *Journal of the American Statistical Association*, 101(475):1228–1240, 2006. URL https://doi.org/10.1198/016214506000000311. [p470]

G. Csárdi and R. FitzJohn. *progress: Terminal Progress Bars*, 2019. URL https://CRAN.R-project.org/package=progress. R package version 1.2.2. [p469]

P. De Valpine. The common sense of p values. *Ecology*, 95(3):617–621, 2014. URL https://doi.org/10.1890/13-1271.1. [p467]

S. Demarta and A. J. McNeil. The t-copula and related copulas. *International Statistical Review*, 73(1): 111–129, 2005. URL https://doi.org/10.1111/j.1751-5823.2005.tb00254.x. [p469, 495]

J. Dobrić and F. Schmid. A goodness of fit test for copulas based on Rosenblatt's transformation. *Computational Statistics & Data Analysis*, 51(9):4633–4642, 2007. URL https://doi.org/10.1016/j.csda.2006.08.012. [p472]

D. Dokuzoğlu and V. Purutçuoğlu. Comprehensive analyses of gaussian graphical model under different biological networks. *Acta Physica Polonica, A.*, 132(3), 2017. URL https://doi.org/10.12693/APhysPolA.132.1106. [p467]

F. Durante and C. Sempi. Copula theory: An introduction. In P. Jaworski, F. Durante, W. K. Härdle, and T. Rychlik, editors, *Copula Theory and Its Applications*, pages 3–31. Springer, Berlin, Heidelberg, 2010. URL https://doi.org/10.1007/978-3-642-12465-5_1. [p467]

Y. Fang and L. Madsen. Modified gaussian pseudo-copula: Applications in insurance and finance. *Insurance: Mathematics and Economics*, 53(1):292–301, 2013. URL https://doi.org/10.1016/j.insmatheco.2013.05.009. [p467]

J.-D. Fermanian. Goodness-of-fit tests for copulas. *Journal of Multivariate Analysis*, 95(1):119–152, 2005. URL https://doi.org/10.1016/j.jmva.2004.07.004. [p471]

J.-D. Fermanian and O. Scaillet. Nonparametric estimation of copulas for time series. *Journal of Risk*, 5: 25–54, 2003. URL https://doi.org/10.21314/JOR.2003.082. [p470]

P. Gaensler and W. Stute. *Seminar on Empirical Processes*. Springer Basel AG, Boca Raton, 1987. URL https://doi.org/10.1007/978-3-0348-6269-1. [p470]

C. Genest and J. Nešlehová. Copulas and copula models. *Wiley StatsRef: Statistics Reference Online*, 2014. URL https://doi.org/10.1002/9781118445112.stat07523. [p467]

C. Genest and B. Rèmillard. Validity of the parametric bootstrap for goodness-of-fit testing in semi-parametric models. *Annales de l'Institut Henri Poincaré, Probabilités et Statistiques*, 6(44):1096–1127, 2008. URL https://doi.org/10.1214/07-AIHP148. [p471]

C. Genest and L.-P. Rivest. Statistical inference procedures for bivariate Archimedean copulas. *Journal of the American Statistical Association*, 88(3):1034–1043, 1993. URL https://doi.org/10.1080/01621459.1993.10476372. [p469, 471]

C. Genest, K. Ghoudi, and L.-P. Rivest. A semi-parametric estimation procedure of dependence parameters in multivariate families of distributions. *Biometrika*, 82(3):543–552, 1995. URL https://doi.org/10.1093/biomet/82.3.543. [p469, 470]

C. Genest, J.-F. Quessy, and B. Rémillard. Goodness-of-fit procedures for copula models based on the probability integral transformation. *Scandinavian Journal of Statistics*, 33:337–366, 2006. URL https://doi.org/10.1111/j.1467-9469.2006.00470.x. [p471]

C. Genest, B. Rémillard, and D. Beaudoin. Goodness-of-fit tests for copulas: A review and a power study. *Insurance: Mathematics and Economics*, 44:199–213, 2009. URL https://doi.org/10.1016/j.insmatheco.2007.10.005. [p467, 470, 471, 472, 474]

T. Gneiting and A. E. Raftery. Strictly proper scoring rules, prediction, and estimation. *Journal of the American Statistical Association*, 102(477):359–378, 2007. URL https://doi.org/10.1198/016214506000001437. [p467]

M. Gomes, R. Radice, J. Camarena Brenes, and G. Marra. Copula selection models for non-gaussian outcomes that are missing not at random. *Statistics in medicine*, 38(3):480–496, 2019. URL https://doi.org/10.1002/sim.7988. [p467]

J. Größer and O. Okhrin. Copulae: An overview and recent developments. *Wiley Interdisciplinary Reviews: Computational Statistics*, page e1557, 2021. URL https://doi.org/10.1002/wics.1557. [p467]

C. Hering and M. Hofert. Goodness-of-fit tests for archimedean copulas in high dimensions. In K. Glau, M. Scherer, and R. Zagst, editors, *Innovations in Quantitative Risk Management*, pages 357–373, Cham, 2015. Springer International Publishing. URL https://doi.org/10.1007/978-3-319-09114-3_21. [p472]

M. Hofert, I. Kojadinovic, M. Maechler, and J. Yan. *copula: Multivariate Dependence with Copulas*, 2020. URL https://CRAN.R-project.org/package=copula. R package version 1.0-0. [p467]

K. Huang, L. Dai, M. Yao, Y. Fan, and X. Kong. Modelling dependence between traffic noise and traffic flow through an entropy-copula method. *Journal of Environmental Informatics*, 29(2), 2017. URL https://doi.org/10.3808/jei.201500302. [p467]

W. Huang and A. Prokhorov. A goodness-of-fit test for copulas. *Econometric Reviews*, 33(7):751–771, 2014. URL https://doi.org/10.1080/07474938.2012.690692. [p473]

H. Joe. *Multivariate Models and Dependence Concepts*. Chapman & Hall, London, 1997. [p469, 470]

H. Joe. Asymptotic efficiency of the two-stage estimation method for copula-based models. *Journal of Multivariate Analysis*, 94(2):401–419, 2005. URL https://doi.org/10.1016/j.jmva.2004.06.003. [p470]

H. Joe and D. Kurowicka. *Dependence Modeling: Vine Copula Handbook*. World Scientific, 2010. URL https://doi.org/10.1142/7699. [p467]

M. Jouini and R. Clemen. Copula models for aggregating expert opinions. *Operations Research*, 3(44): 444–457, 1996. URL https://doi.org/10.1287/opre.44.3.444. [p471]

S. Konigorski, Y. E. Yilmaz, and S. B. Bull. Bivariate genetic association analysis of systolic and diastolic blood pressure by copula models. *BMC Proceedings*, 8(1):S72, 2014. URL https://doi.org/10.1186/1753-6561-8-S1-S72. [p467]

O. Kuss, A. Hoyer, and A. Solms. Meta-analysis for diagnostic accuracy studies: a new statistical model using beta-binomial distributions and bivariate copulas. *Statistics in medicine*, 33(1):17–30, 2014. URL https://doi.org/10.1002/sim.5909. [p467]

Z. Liu, S. Guo, L. Xiong, and C.-Y. Xu. Hydrological uncertainty processor based on a copula function. *Hydrological sciences journal*, 63(1):74–86, 2018. URL https://doi.org/10.1080/02626667.2017.1410278. [p467]

X. Ma, S. Luan, B. Du, and B. Yu. Spatial copula model for imputing traffic flow data from remote microwave sensors. *Sensors*, 17(10):2160, 2017. URL https://doi.org/10.3390/s17102160. [p467]

F. Michiels and A. De Schepper. A copula test space model how to avoid the wrong copula choice. *Kybernetika*, 44(6):864–878, 2008. URL http://hdl.handle.net/10338.dmlcz/135896. [p495]

T. Nagler, U. Schepsmeier, J. Stoeber, E. C. Brechmann, B. Graeler, and T. Erhardt. *VineCopula: Statistical Inference of Vine Copulas*, 2019. URL https://CRAN.R-project.org/package=VineCopula. R package version 2.3.0. [p467, 473, 490, 496, 497, 498]

D. Oakes. Multivariate survival distributions. *Journal of Nonparametric Statistics*, 3(3-4):343–354, 1994. URL https://doi.org/10.1080/10485259408832593. [p470]

D. H. Oh and A. J. Patton. Time-varying systemic risk: Evidence from a dynamic copula model of cds spreads. *Journal of Business & Economic Statistics*, 36(2):181–195, 2018. URL https://doi.org/10.1080/07350015.2016.1177535. [p467]

N. Phillips. *yarrr: A Companion to the e-Book "YaRrr!: The Pirate's Guide to R"*, 2017. URL https://CRAN.R-project.org/package=yarrr. R package version 0.1.5. [p469, 477, 487, 490]

J.-D. F. D. Radulovic and M. Wegkamp. Weak convergence of empirical copula processes. *Bernoulli*, 10 (5):847–860, 2004. URL https://doi.org/10.3150/bj/1099579158. [p470]

B. Remillard and J.-F. Plante. *TwoCop: Nonparametric test of equality between two copulas*, 2012. URL https://CRAN.R-project.org/package=TwoCop. R package version 1.0. [p467]

B. Remillard and O. Scaillet. Testing for equality between two copulas. *Journal of Multivariate Analysis*, 100:377–386, 2009. URL https://doi.org/10.1016/j.jmva.2008.05.004. [p467]

M. Rosenblatt. Remarks on a multivariate transformation. *Annals of Mathematical Statistics*, 23(3): 470–472, 1952. URL https://doi.org/10.1214/aoms/1177729394. [p471, 472]

I. D. L. Salvatierra and A. J. Patton. Dynamic copula models and high frequency data. *Journal of Empirical Finance*, 30:120–135, 2015. URL https://doi.org/10.1016/j.jempfin.2014.11.008. [p467]

O. Scaillet. Kernel-based goodness-of-fit tests for copulas with fixed smoothing parameters. *Journal of Multivariate Analysis*, 98(3):533–543, 2007. URL https://doi.org/10.1016/j.jmva.2006.05.006. [p467, 473, 474]

U. Schepsmeier. Efficient information based goodness-of-fit tests for vine copula models with fixed margins: A comprehensive review. *Journal of Multivariate Analysis*, 138:34–52, 2015. URL https://doi.org/10.1016/j.jmva.2015.01.001. [p473]

P. Shi, X. Feng, and J.-P. Boucher. Multilevel modeling of insurance claims using copulas. *The Annals of Applied Statistics*, 10(2):834–863, 2016. URL https://doi.org/10.1214/16-AOAS914. [p467]

J. H. Shih and T. A. Louis. Inferences on the association parameter in copula models for bivariate survival data. *Biometrics*, 51(4):1384–1399, 1995. URL https://doi.org/10.2307/2533269. [p470]

A. Sklar. Fonctions de répartition à n dimension et leurs marges. *Publications de l'Institut de Statistique de l'Université de Paris*, 8:299–231, 1959. [p467]

D. Valle and D. Kaplan. Quantifying the impacts of dams on riverine hydrology under non-stationary conditions using incomplete data and gaussian copula models. *Science of The Total Environment*, 677: 599–611, 2019. URL https://doi.org/10.1016/j.scitotenv.2019.04.377. [p467]

W. Wang and M. Wells. Model selection and semiparametric inference for bivariate failure-time data. *Journal of the American Statistical Association*, 95(449):62–76, 2000. URL https://doi.org/10.1080/01621459.2000.10473899. [p471]

H. White. Maximum likelihood estimation of misspecified models. *Econometrica: Journal of the Econometric Society*, 50:1–25, 1982. URL https://doi.org/10.2307/1912526. [p473, 474]

F. Wu, E. Valdez, and M. Sherris. Simulating from exchangeable archimedean copulas. *Communications in Statistics—Simulation and Computation*, 36(5):1019–1034, 2007. URL https://doi.org/10.1080/03610910701539781. [p472]

S. Zhang, O. Okhrin, Q. M. Zhou, and P. X.-K. Song. Goodness-of-fit test for specification of semiparametric copula dependence models. *Journal of Econometrics*, 193(1):215–233, 2016. URL https://doi.org/10.1016/j.jeconom.2016.02.017. [p467, 468, 474, 489, 491]

Appendix A

Table A.1 contains parameter ranges, the bivariate cumulative distribution function (CDF), and possible values of Kendall's τ for the copulae available in **gofCopula**.

Copula	$\theta \in$	$C(u_1, u_2)$	$\tau \in$
Normal	$[-1, 1]$	$\int_{-\infty}^{\Phi^{-1}(u_1)} \int_{-\infty}^{\Phi^{-1}(u_2)} \frac{1}{2\pi\sqrt{1-\theta^2}} \exp\left\{\frac{2\theta st - s^2 - t^2}{2(1-\theta^2)}\right\} dsdt$	$[-1, 1]$
t	$[-1, 1]$ $\nu > 0$	$\int_{-\infty}^{t_\nu^{-1}(u_1)} \int_{-\infty}^{t_\nu^{-1}(u_2)} \frac{1}{2\pi\sqrt{1-\theta^2}} \left\{1 + \frac{s^2+t^2-2\theta st}{\nu(1-\theta^2)}\right\}^{-\frac{\nu+2}{2}} dsdt$	$[-1, 1]$
Clayton	$[-1, \infty)\backslash\{0\}$	$\left\{\max(u_1^{-\theta} + u_2^{-\theta} - 1, 0)\right\}^{-\frac{1}{\theta}}$	$[-1, 1]$
Gumbel	$[1, \infty)$	$\exp\left[-\left\{(-\log u_1)^{\frac{1}{\theta}} + (-\log u_2)^{\frac{1}{\theta}}\right\}^{\theta}\right]$	$[0, 1]$
Frank	$(-\infty, \infty)\backslash\{0\}$	$-\frac{1}{\theta}\log\left[1 + \frac{\{\exp(-\theta u_1)-1\}\{\exp(-\theta u_2)-1\}}{\exp(-\theta)-1}\right]$	$[-1, 1]$
Joe	$[1, \infty)$	$1 - \left\{(1-u_1)^{\theta} + (1-u_2)^{\theta} - (1-u_1)^{\theta}(1-u_2)^{\theta}\right\}^{\frac{1}{\theta}}$	$[0, 1]$
AMH	$[-1, 1]$	$\frac{u_1 u_2}{1-\theta(1-u_1)(1-u_2)}$	$[\frac{5-8\log 2}{3}, \frac{1}{3}]$ $\approx [-0.1817, 0.3333]$
Galambos	$[0, \infty)$	$u_1 u_2 \exp\left[\left\{(-\log u_1)^{-\theta} + (-\log u_2)^{-\theta}\right\}^{-\frac{1}{\theta}}\right]$	$[0, 1]$
Husler-Reiss	$[0, \infty)$	$\exp\left\{\log(u_1)\Phi(\frac{1}{\theta} + \frac{1}{2}\theta\log\frac{\log u_1}{\log u_2}) + \log(u_2)\Phi(\frac{1}{\theta} + \frac{1}{2}\theta\log\frac{\log u_2}{\log u_1})\right\}$	$[0, 1]$
Tawn	$[0, 1]$	$u_1 u_2 \exp\left(-\theta\frac{\log u_1 \log u_2}{\log u_1 + \log u_2}\right)$	$[0, \frac{8\arctan(\sqrt{\frac{1}{3}})}{\sqrt{3}} - 2]$ $\approx [0, 0.4184]$
t-EV	$[-1, 1]$ $\nu > 0$	see Demarta and McNeil (2005)	$[0, 1]$
FGM	$[-1, 1]$	$u_1 u_2 + u_1 u_2 \theta(1-u_1)(1-u_2)$	$[-\frac{2}{9}, \frac{2}{9}]$
Plackett	$(0, \infty)$	$\frac{\{1+(\theta-1)(u_1+u_2)\} - \sqrt{\{1+(\theta-1)(u_1+u_2)\}^2 - 4u_1u_2\theta(\theta-1)}}{2(\theta-1)}$	$[-1, 1]$

Table A.1: This table is mainly based on Michiels and De Schepper (2008). AMH abbreviates Ali-Mikhail-Haq. FGM stands for Farlie-Gumbel-Morgenstern. Φ is the CDF of the univariate standard normal distribution and Φ^{-1} its inverse. t_ν^{-1} is the inverse CDF of the univariate t-distribution with ν degrees of freedom. The expression of the CDF for the t-EV is complex due to the construction via a Pickands dependence function, which is why we do not explicitly list it. The given parameterization of the tawn copula is based on the one implemented in the package **copula**.

Appendix B

This section is devoted to the results of Section 2.6.2.

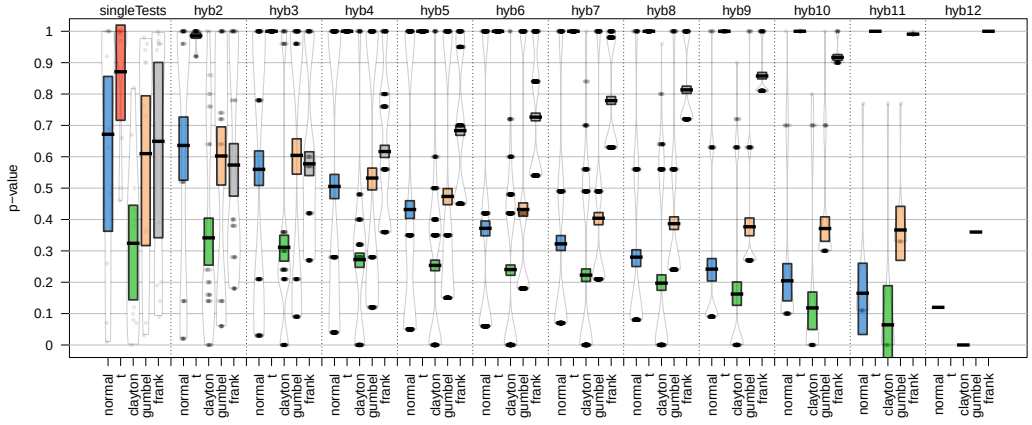


Figure B.1: p-values of the C/BoA data for 2005. The column `hyb12` of the t-copula is empty, as the p-value of the test `gofWhite` could not be computed due to instability in the test statistics. For a detailed description of this phenomenon, see Nagler et al. (2019).

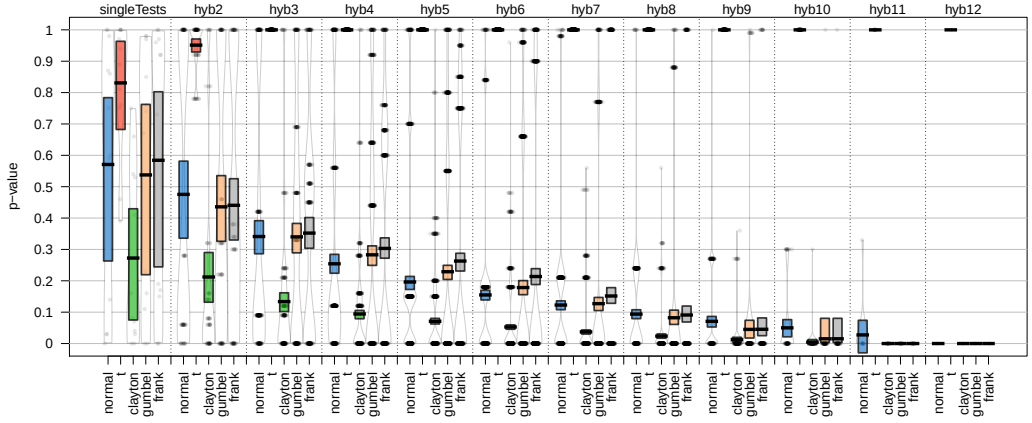


Figure B.2: p-values of the C/BoA data for 2006.

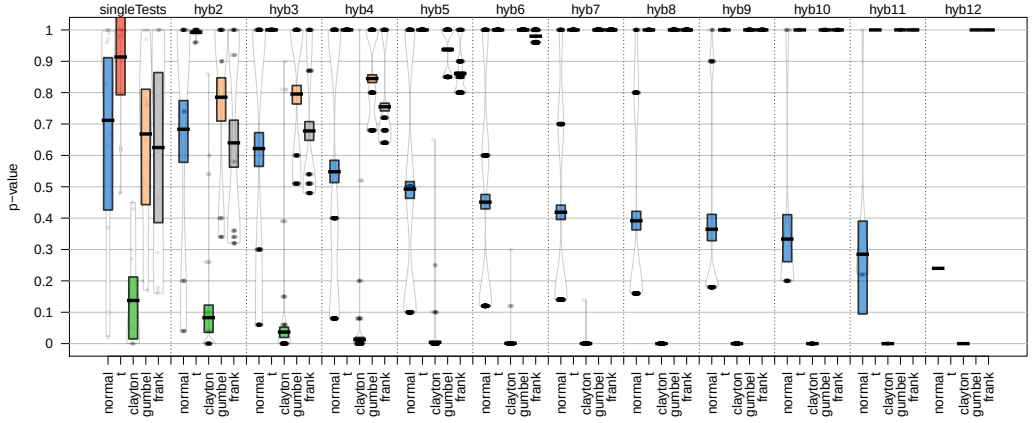


Figure B.3: p-values of the C/BoA data for 2008. The column `hyb12` of the t-copula is empty, as the p-value of the test `gofWhite` could not be computed due to instability in the test statistics. For a detailed description of this phenomenon, see Nagler et al. (2019).

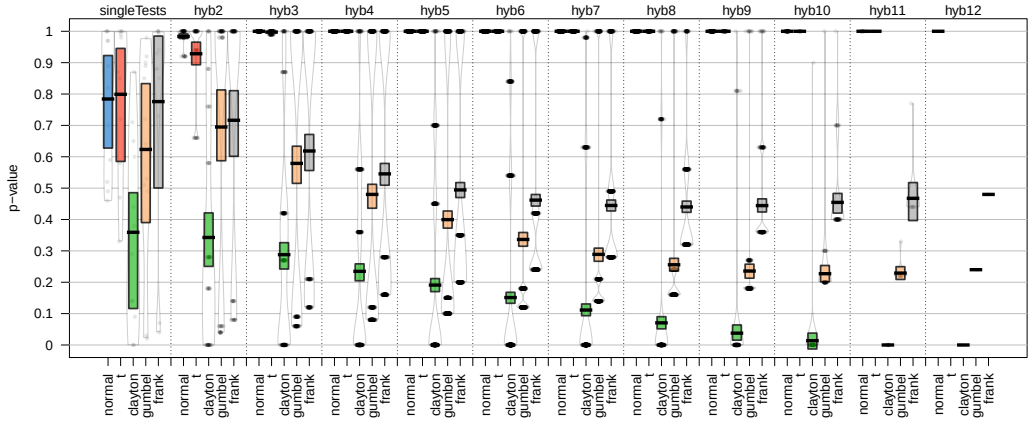


Figure B.4: p-values of the C/BoA data for 2009. The column `hyb12` of the t-copula is empty, as the p-value of the test `gofWhite` could not be computed due to instability in the test statistics. For a detailed description of this phenomenon, see Nagler et al. (2019).

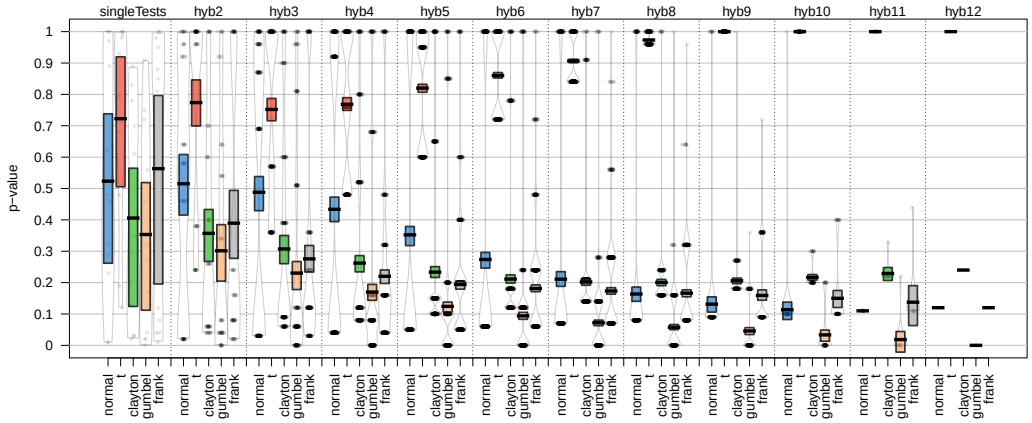


Figure B.5: p-values of the C/BoA data for 2010.

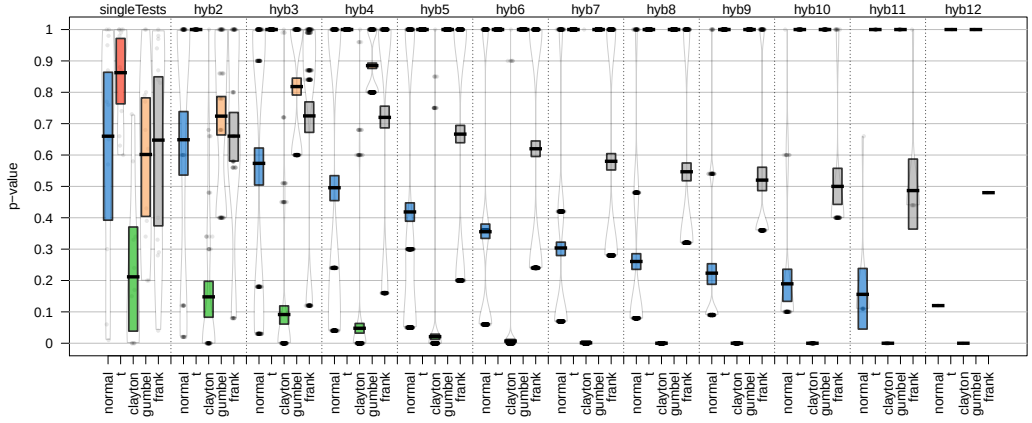


Figure B.6: p-values of the C/BoA data for 2011.

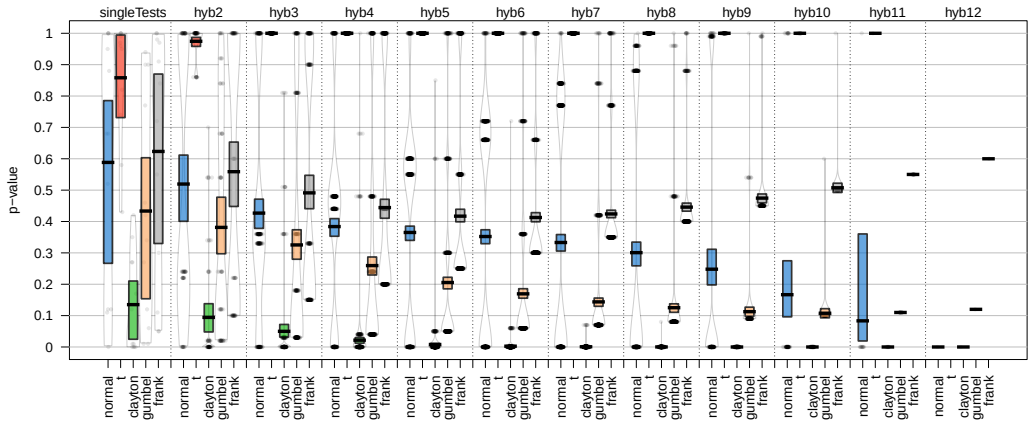


Figure B.7: *p*-values of the C/BoA data for 2012. The column `hyb12` of the *t*-copula is empty, as the *p*-value of the test `gofWhite` could not be computed due to instability in the test statistics. For a detailed description of this phenomenon, see Nagler et al. (2019).

Ostap Okhrin
Chair of Econometrics and Statistics, esp. in Traffic Sciences
Institute of Transportation Economics
Faculty of Transportation
Technische Universität Dresden
Würzburger Street 35, 01187 Dresden
Germany
ostap.okhrin@tu-dresden.de

Simon Trimborn
Department of Management Sciences
and School of Data Science
City University of Hong Kong
7-268, Lau Ming Wai Academic Building
Hong Kong
trimborn.econometrics@gmail.com

Martin Waltz
Chair of Econometrics and Statistics, esp. in Traffic Sciences
Institute of Transportation Economics
Faculty of Transportation
Technische Universität Dresden
Würzburger Street 35, 01187 Dresden
Germany
martin.waltz@tu-dresden.de

Analyzing Dependence between Point Processes in Time Using IndTestPP

by Ana C. Cebrián and Jesús Asín

Abstract The need to analyze the dependence between two or more point processes in time appears in many modeling problems related to the occurrence of events, such as the occurrence of climate events at different spatial locations or synchrony detection in spike train analysis. The package **IndTestPP** provides a general framework for all the steps in this type of analysis, and one of its main features is the implementation of three families of tests to study independence given the intensities of the processes, which are not only useful to assess independence but also to identify factors causing dependence. The package also includes functions for generating different types of dependent point processes, and implements computational statistical inference tools using them. An application to characterize the dependence between the occurrence of extreme heat events in three Spanish locations using the package is shown.

Introduction

A point process in time (PP in short) is a random collection of points in a space in $\mathbb{R}^+$ where each point usually represents the time of an event. Examples of events that can be modeled as point processes in time include the occurrence of earthquakes, heat waves, or the arrivals of insurance claims. Many real problems involve not one but two or more PPs, and they should be studied in a multivariate framework where a description of the independence or dependence structure between them is required. Examples of these situations are the timing of the trades and mid-quote changes in the stock exchange, the occurrence of temperature extremes or other climate events at different spatial locations, or the synchrony detection in spike train analysis.

In those situations, statistical tests are required to assess the independence between two or more PPs. If we can assume that the processes are independent, their modeling is much simpler, since it can be carried out separately for each process without any loss of information. The tests are also useful to identify the type of dependence and select the type of vector of point processes used to model them. The need of testing independence between PPs appears in climate and environmental sciences (Cronie and van Lieshout, 2016; Abaurrea et al., 2015), in neuroscience (Tuleau-Malot et al., 2014; Albert et al., 2015), in biology (Myllymäki et al., 2017), and many other fields.

Two types of independence between PPs may be of interest, general independence (Rubin-Delanchy and Heard, 2014a) and independence, given the intensities of the processes. The election of the type of independence as null hypothesis depends on the aim of the study, but the second type is more useful in modeling problems based on PPs. In effect, the most frequent approach to model systematic dependence structures caused by common factors is the use of nonhomogeneous processes with intensities, which are functions of the same or dependent covariates. To analyze if the dependence is well represented by those covariates, the null hypothesis of independence given the intensities has to be checked. When the existing dependence cannot be explained by the available covariates, models taking into account that dependence should be considered to model the vectors of PPs.

The R package **IndTestPP** (Cebrián, 2020) provides a general framework for all the steps to analyze the dependence in a vector of point processes in time: from data processing and tests of independence to inference tools for parameters of interest. That makes it a useful tool for applications based on the modeling of a vector of point processes. As far as we know, there is not other software for this type of analysis. One of the main features of the package is the implementation of the three families of independence tests by Cebrián et al. (2020), which cover a wide variety of homogeneous and nonhomogeneous processes appearing in real problems: Poisson processes, processes with a parametric marginal model, point processes with known marginal intensities, etc. The package also provides functions to generate four different models of dependent PPs, and two types of independent PPs, which are useful to develop inference tools based on computational statistical methods.

The outline of the paper is as follows. The two first sections *Vector of point processes in time* and *Point processes in R* introduce some properties for vectors of point processes and some R packages related to this topic. The three following sections describe the implementation in R of the tests of independence, the measures of dependence, and the tools for generating PPs. The final section shows an illustrative example of an analysis to characterize the dependence between the occurrence of extreme heat events in three Spanish locations using **IndTestPP**.

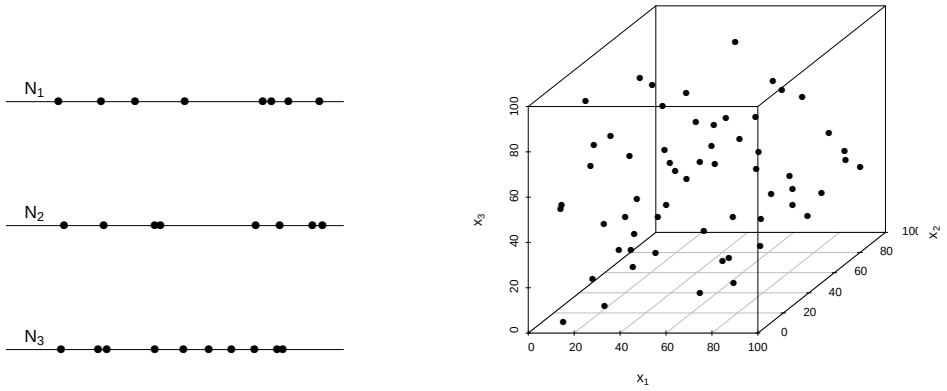


Figure 1: Vector of three point processes (left), and multivariate point process in a tridimensional space (right).

Vectors of point processes in time

A point process in time, N, is defined in $\mathbb{R}^+$ and can be described in different equivalent ways. Here, we will mainly use the sequence of occurrence times, $T_1, T_2, ..., T_n$, and also the set of random variables $N(A)$ representing the number of points in A, for each $A \in \mathbb{R}^+$. The notation for $A = (0, t]$ is $N(t) = N((0, t])$. The intensity measure of the process Λ gives the expected number of points in a set, so that $\Lambda((0, t]) = E(N((0, t]))$. Its derivative function, provided it exists, is the intensity function,

$$\lambda(t) = \frac{\partial \Lambda((0, t])}{\partial t}.$$

If the intensity is constant, the process is homogeneous and nonhomogeneous otherwise. The most known PP is the Poisson process, where $N(A)$ has a $Poisson(\mu_A)$ distribution, with $\mu_A = \int_A \lambda(t)dt$, and $N(A_1), \dots N(A_k)$ are independent variables provided that $A_i \cap A_j = \emptyset, \ \forall i \neq j$.

Herein, we will consider vectors of point processes $\mathbf{N} = (N_1, \dots, N_d)$ observed in the same space $\Omega = (0, T] \subset \mathbb{R}^+$. This definition must not be mixed up with a multivariate point processes defined as a process of random points $\mathbf{X_i} = (X_{i1}, \dots, X_{id})$ in a d-dimensional space $V \in \mathbb{R}^d$, whose simplest example with $d = 2$ is a spatial point processes. Figure 1 shows the differences between the two concepts with $d = 3$. A vector of PPs can be seen as a marked point process with discrete marks, and can be represented by a countable collection of pairs (T_i, D_i) where $T_i \in \mathbb{R}^+$ are the occurrence times and $D_i \in \{1, \dots, d\}$ are the component indexes.

Most of the results in this work are developed for vectors of $d = 2$ processes, denoted by (N_x, N_y) with intensities $\lambda_x(t)$ and $\lambda_y(t)$. If the results can be extended to higher values of d, it is specified. The n_x and n_y points in each observed process are denoted $t_1, \dots, t_{n_x}$ and $s_1, \dots, s_{n_y}$, respectively.

Many types of dependence structures can appear between the marginal processes of a vector. The most direct way of modeling it is to use models to represent the dependence between the occurrence times of the processes, such as the common Poisson shock processes, the queue processes, the Poisson processes with dependent marks, or the multivariate Neyman-Scott processes, described later.

Point processes in R

There exist many packages in R devoted to the analysis of spatial point processes: the extensive **spatstat** (Baddeley et al., 2015), whose main functionalities include exploratory data analysis, model-fitting, and simulation, **stpp** (Gabriel et al., 2020), **splancs** (Rowlingson and Diggle, 2017), and many others. **IDSpatialStats** (Giles et al., 2019) provides spatial dependence measures, and future directions include the extension to the spatio-temporal case. However, the number of packages dealing with the analysis of point processes in time is not so high, and most of them deal with univariate analysis of the processes. **NHPoisson** (Cebrián et al., 2015) provides a global framework for the modeling and diagnosis of Poisson processes in time, **PtProcess** (Harte, 2010) fits and analyses time-dependent

marked point processes with an emphasis on earthquake modeling, and **mmpp** (Hino et al., 2017) offers various similarity and distance metrics for marked point processes.

The aim of **IndTestPP** is the analysis of vectors of point processes in time, in particular of its dependence, and it provides a general framework for all the steps involved in this type of analysis: data processing, estimation of the marginal intensities of the processes, analysis of independence given the intensity, identification of factors causing dependence, and inference tools based on computational statistics. **mppa** (Rubin-Delanchy and Heard, 2014b) provides a test for dependence between point processes on the real line, but with a different aim since it tests general independence. The three families of tests implemented in **IndTestPP** are more general since they are not restricted to Poisson processes, and they test independence given the marginal intensities. This type of conditional independence is more useful in statistical modeling of vectors of point processes since it helps to identify the factors that cause the dependence. An example of how all the steps of the modeling of a vector of point processes can be carried out using **IndTestPP** is shown in the application section.

Testing independence between point processes in time

Most of the analysis of independence between point processes in the literature involve spatial processes, but few works deal with the study of independence between processes in time. **IndTestPP** includes the three families of tests to assess independence between PPs in time by Cebrián et al. (2020), i.e., the POISSON, the CLOSE, and the CROSS families, and a graphical tool, the Dutilleul plot. In all of them, the null hypothesis is the independence between the point processes, given their marginal intensities, and the alternative the existence of any type of random dependence between them. All the tests in these families are constructed by keeping fixed the first observed process, a common approach to test independence given the marginal structure. However, each test is based on different assumptions, and all together they cover a wide range of types of processes appearing in real problems.

POISSON family

The family of tests POISSON is implemented in the function `CondTest`, and it includes two tests to assess the independence between two homogeneous or nonhomogeneous processes, based on the conditional distribution of $N_y|N_x$. The assumptions of the tests are that N_y is a Poisson process with intensity function $\lambda_y(t)$, specified in the vector argument `lambday`.

This family is based on the following property. If N_x and N_y are independent, and a point t_i occurs in N_x, the distribution of N_y does not change. Then, Y_i, the number of points in N_y in intervals l_i of length $2r$ around t_i, follows a $Poisson(\mu_i)$ distribution with $\mu_i = \int_{l_i} \lambda_y(t)dt$.

Two options are available to perform a test. The test implemented with the argument `type='Poisson'` is based on the fact that under the independence between N_x and N_y and if not overlapping intervals are used, the statistic $Y = \sum_{i=1}^{n_x} Y_i$ has a $Poisson(\mu)$ distribution with $\mu = \sum_{i=1}^{n_x} \mu_i$. A test based on a Normal approximation is implemented with the argument `type='Normal'`. Again, under the null and with not overlapping intervals, the variables $(Y_i - \mu_i)/\sqrt{\mu_i}$ are zero mean independent variables with standard deviation equal to 1, and the asymptotic distribution of the statistic

$$O_{n_x} = \frac{1}{\sqrt{n_x}} \sum_{j=1}^{n_x} \frac{Y_j - \mu_j}{\sqrt{\mu_j}}$$

is $N(0,1)$. If the argument is `type='All'`, both tests are calculated.

The intervals where the number of points is counted, l_i , are centered intervals around points t_i of radius r specified by the argument `r`. If `changer=TRUE`, when two intervals overlap, their lengths are shortened by half of the intersection period; in this way the resulting intervals are disjoint and, consequently, the corresponding variables Y_i are independent.

The power study by Cebrián et al. (2020) shows that the Normal test performs better provided that conditions to guarantee the normal approximation are fulfilled. These conditions are quite weak, even with a complex intensity, mean values of μ_i around 0.6 points per interval lead to a valid approximation with $n_x = 50$, and around 0.3 with $n_x = 100$.

CLOSE family

The CLOSE family includes two tests, the parametric bootstrap (PaB) and the Lotwick-Silverman (LoS) tests, implemented in the functions `TestIndNH` and `TestIndLS`, respectively. The LoS test can only be applied to homogeneous processes, but the PaB test also to nonhomogeneous ones. On the other

hand, the LoS does not require any assumption to be applied, while PaB requires that N_y follows a parametric model with a generation algorithm, such as a Poisson or a Neyman-Scott cluster process. Although both tests allow checking the independence between d processes in general, the calculation of the statistic is only implemented for $d = 2$ and $d = 3$.

These tests are based on the close point distance, and they aim to compare the behavior of the sets of close points in a vector of observed processes and in vectors with the same marginal distributions but independent components (Abaurrea et al., 2015). A point s_j in N_y is a close point of t_i in N_x if the intervals to their previous points, s_{j-1} and t_{i-1}, overlap. The set behavior is summarized by $\bar{d}x_i$, the mean of the distances between t_i and its close points, $|s_j - t_i|$. The set of close points and the mean distance for each point t_i are calculated by the functions `uniogentri` and `DistObs`, respectively.

Given the complexity of the test statistic, its distribution has to be obtained by computational statistical methods. These methods require approaches to generate a sample of r processes $N_j^* = (N_x, N_{y,j}^*)$, where the observed process N_x is fixed, $N_{y,j}^*$ has the same distribution as N_y, and N_x and $N_{y,j}^*$ are independent. The PaB and LoS tests result from two different generation approaches.

Parametric bootstrap test. In this test, the $N_{y,j}^*$ processes are generated independently from N_x using a parametric model. Two types of marginal models are implemented in **TestIndNH**: Poisson processes (`type="Poisson"`) and Neyman-Scott cluster processes (`type="PoissonCluster"`).The generation of Poisson processes in a given period uses the function `simNHPc`, based on a two-step algorithm which generates homogeneous occurrence times, and transform them into the points of a NH process with intensity $\lambda(t)$ (Ross, 2006). Neyman-Scott cluster processes are obtained by the function `IndNHNeyScot`. Details about these processes are explained later.

Lotwick-Silverman test (LoS). `TestIndLS` generates processes using a Monte Carlo method conditional on the observed marginal structure (Lotwick and Silverman, 1982). The steps are the following:

1. The observed processes (N_x, N_y) are wrapped onto a circumference by identifying the opposite sides of the time interval where they are observed.
2. Fixing N_x, a new N_y^* is generated by translating N_y a random uniform amount on the circumference. This breaks any dependence between the processes and keeps the marginal distributions, provided they do not change over time.

The mean distances of the close point sets in the generated vectors of processes in the PaB and the LoS tests are calculated by the functions `DistSim` and `DistShift`, respectively. The calculation of the p-value requires the generation of processes in two steps of the algorithm, to calculate the expectation of the mean distances $\bar{d}x_{ij}$ and to estimate the distribution of the statistic in a correlated sample. This calculation is implemented so that the same generated processes are used in the two steps and the computing time is kept low; otherwise, it would be multiplied by the number of simulations. Moreover, parallel computation is implemented.

According to the power study by Cebrián et al. (2020), both LoS and PaB tests have high power, but LoS performs slightly better in the homogeneous processes with small samples and low dependence.

CROSS family

The CROSS family includes two tests based on the cross K and the cross J spatial functions adapted to the case of PPs in time, implemented in the functions `NHK` and `NHJ`, respectively. These functions also provide estimators of the cross functions. They do not require any assumption about the distribution of the marginal processes, only to know their intensities, and the p-values are calculated using a LoS approach. The tests can be applied to two homogeneous or nonhomogeneous processes, and more generally to two sets of processes, $\mathbf{C} = (N_{x1}, N_{x2}, \ldots, N_{xl_C})$ and $\mathbf{D} = (N_{y1}, N_{y2}, \ldots, N_{yl_D})$. The information about the processes is provided by arguments `posC`, a vector containing all the occurrence times in the processes in **C**, and `typeC`, a vector containing the code j of the process N_{xj} where each point in `posC` occurs; **D** is specified analogously by `posD` and `typeD`. For the sake of simplicity, the results are expressed for the case, $\mathbf{C} = (N_x)$ and $\mathbf{D} = (N_y)$ with intensities $\lambda_x(t)$ and $\lambda_y(t)$.

K-function. $K_{xy}(r)$ is the expected value of the number of points in N_y within a distance r of a randomly chosen point in N_x, adjusted for the possible time-varying intensity. `NHK` calculates two different estimators of $K_{xy}(r)$ at a given grid of r distances, and the corresponding test statistics based on them, $\mathcal{K} = \frac{1}{R}\sum_{r=r_1}^{r_R} \hat{K}_{xy}(r)/2r$. The estimator calculated by default (`typeEst = 2`) performs better in terms of size and power (Cebrián et al., 2020).

J-function. It compares the functions $D_{xy}(r)$ (distribution function of the distances from a point in N_x to the nearest point in N_y) and $F_y(r)$ (distribution function of the distances from a point in the

space to the nearest point in N_y), in terms of the ratio $J_{xy}(r) = (1 - D_{xy}(r))/(1 - F_y(r))$, if $F_y(r) < 1$.

The estimators of $D_{xy}(r)$ and $F_y(r)$, calculated by `NHD` and `NHF`, are used in `NHJ` to estimate $J_{xy}(r)$. To estimate $F_y(r)$, a grid of L values is required. It can be provided in argument `L` or an automatic selection is calculated otherwise. In the homogeneous PPs, the previous estimators are equal to the empirical distribution functions, and the calculation algorithms are changed to reduce the computational cost. The test statistic, which summarizes the deviations of the function from 1, is $\mathcal{J} = \frac{1}{R}\sum_{r=r_1}^{r_R} |\hat{J}_{xy}(r) - 1|$.

In both functions, `NHK` and `NHJ`, the grid of r distances where the estimators are evaluated are provided in argument `r`. If it is NULL, an automatic selection based on length T is carried out. The test statistics are also evaluated at that grid, and since dependence often appears between close observations, the addends with high r can make it more difficult to discriminate between dependent and independent processes. To avoid that effect, the statistic can be calculated using only the addends with $r < r0$ by using the argument `rTest=r0`. To identify an adequate value of $r0$, values $\hat{K}_{xy}(r)$ or $\hat{K}_{xy}(r)/2r$ can be optionally plotted.

Computation of the *p*-values. The calculation of the p-value in CROSS tests is based on a LoS approach for nonhomogenous processes. First, the observed processes (N_x, N_y) are wrapped onto a circumference. Then, fixing N_x, a new N_y^* is generated by translating N_y and its intensity $\lambda_y(t)$, a random uniform amount. This breaks any dependence, but in nonhomogeneous PPs, it changes the distribution of the marginal processes. However, since the cross functions are adjusted for the time-varying intensity, which is also translated, valid samples of $K_{xy}(r)$ (or $J_{xy}(r)$) under independence are obtained. Using the empirical distribution of those samples, the p-value and confidence envelopes for $K_{xy}(r)$ (or $J_{xy}(r)$) are obtained. Parallel computation is implemented for these calculations.

Dutilleul plot

The function `DutilleulPlot` carries out Diggle's randomization testing procedure extended by Dutilleul (2011), which graphically assesses the independence between two homogeneous or nonhomogeneous Poisson processes, given their marginal structure. The idea is to plot the cumulative relative frequency of the nearest neighbor distances between the points in the two observed processes and to analyze the independence using a confidence band calculated from simulated independent Poisson processes with the observed marginal intensities.

Dependence measures

Unfortunately, there does not exist a general definition to quantify the dependence between two PPs. However, we suggest some measures implemented in **IndtestPP** which can be useful to describe the level of dependence between many types of processes.

Correlation between the counting variables of two PPs. `CountingCor` calculates a sample estimator of $\rho_{xy}^L = Cor(X_i, Y_i)$, the correlation coefficient between X_i and Y_i the number of points in processes N_x and N_y, in an interval l_i of length L, using a partition of the observed period. Given the discrete character of X_i and Y_i, and since the usual aim is to quantify any type of dependence, not only linear correlation, Spearman or Kendall coefficients are often more adequate. Kendall should be preferred with short intervals since there will be a high number of 0 or 1 occurrences per interval, and the Kendall Tau-b coefficient implemented in the function makes an adjustment for the ties.

In nonhomogenous processes, variables X_i (and Y_i) in intervals measured at different times are not i.d. In the case of Poisson processes, `CountingCor` can calculate a standardized version of the measure, so that all the variables have the same mean and variance, and if $\Lambda_{x,i} = \sum_{t\in l_i} \lambda_x(t)$ and $\Lambda_{y,i} = \sum_{t\in l_i} \lambda_y(t)$ are high enough, they are also i.d.,

$$\rho_{xy}^L = Cor\left(\frac{X_i - \Lambda_{x,i}}{\sqrt{\Lambda_{x,i}}}, \frac{Y_i - \Lambda_{y,i}}{\sqrt{\Lambda_{y,i}}}\right).$$

This coefficient measures the correlation given the marginal intensities. That means that the coefficient measures the correlation once that dependence captured by the intensities (through common covariates, for example) has been removed.

Percentage of concordant intervals. A simpler descriptive measure is the percentage of concordant intervals, that is, the percentage of intervals with occurrences in both processes. It is calculated by `BinPer` as $n_{x,y}/(n_{x,y} + n_{x,0} + n_{0,y})$, where $n_{x,y}$ is the number of intervals with at least one point in both processes, and $n_{0,y}$ and $n_{x,0}$ are the number of intervals with at least one point in one process and

0 in the other. This percentage will tend to be zero in short intervals of independent processes, while positive values will suggest positive dependence. An adequate length of interval that depends on the marginal intensities has to be selected to obtain useful interpretations.

Extremal dependence coefficients. In the case of PPs resulting from a Peak over threshold (POT) approach, another interesting measure is the extremal dependence between the variables X and Y where the POT approach is applied. The extremal dependence is the tendency for one variable to be large, given that the other one is large. The extremal dependence coefficient χ of X given Y is defined as $\chi_{x|y} = \lim_{u\to 1} \chi_{x|y}(u)$, where

$$\chi_{x|y}(u) = P\left(U > u | V > u\right),$$

and (U, V) are the transformed uniform marginals of X and Y. Another extremal coefficient is $\bar{\chi}_{x|y} = \lim_{u\to 1} \bar{\chi}_{x|y}(u)$, where

$$\bar{\chi}_{x|y}(u) = 2\,\frac{\log P\left(U > u\right)}{\log P\left(U > u, V > u\right)} - 1.$$

$\chi_{x|y}$ is on the scale $[0, 1]$, with the set $(0, 1]$ corresponding to asymptotic dependence, and the measure $\bar{\chi}_{x|y}$ falls within the range $[-1, 1]$, with $[-1, 1)$ corresponding to asymptotic independence. Thus, ($\chi_{x|y} > 0$, $\bar{\chi}_{x|y} = 1$) signifies asymptotic dependence, and the value of χ determines the strength of dependence within that class; ($\chi_{x|y} = 0$, $\bar{\chi}_{x|y} < 1$) signifies asymptotic independence, and $\bar{\chi}_{x|y}$ determines the strength of dependence within that class. Full details can be found in Coles et al. (1999).

The function `depchi` estimates the functions $\chi_{x|y}(u)$ and $\bar{\chi}_{x|y}(u)$ in a given grid of values u. Both $\chi_{x|y}(u)$ and $\chi_{y|x}(u)$ (or $\bar{\chi}_{x|y}(u)$ and $\bar{\chi}_{y|x}(u)$) are calculated and optionally plotted. These graphs are useful to estimate the limit of the function and obtain $\hat{\chi}_{x|y}$ (and $\hat{\bar{\chi}}_{x|y}$). In the plot of $\chi_{x|y}(u)$, the expected behavior under independence is also plotted.

Generating point processes with different dependence structures

The generation of vectors of PPs with a given dependence structure is necessary to implement Monte Carlo, parametric bootstrap, or other inference methods based on simulation, such as those described in section *Inference based on computational statistical methods*. There are different approaches to model the dependence between the marginal processes in a vector of PPs, but the most direct way is to model the dependence between the occurrence times of the processes. **IndTestPP** includes functions for the generation of four types of vectors of homogeneous or nonhomogeneous PPs, which will be described later in this section: common Poisson shock processes, multivariate Neyman-Scott processes, queue processes, and marked Poisson processes. These types of vectors allow modeling three dependence structures frequently observed in real problems.

- *Dependence between two or more PPs provoked by the same shock triggering event*. This is the most common dependence structure and can be modeled by a common Poisson shock process (CPSP) or a multivariate Neyman-Scott process (MNSP). Both models show a short-term and positive dependence, generated by common shocks, but in each one, the shocks yield points in the processes in a different way. This dependence appears in the spike trains of two neurons, in climate and environmental processes, or in financial problems, for example, when a political crisis provokes the occurrence of large decreases in different economical indexes.
- *Dependence between shifted processes*. This is a point-to-point dependence, where the occurrence of an event in a process triggers an event in the other so that the points in N_x are shifted a positive random amount in N_y. It can be modeled by a queue or a network of queues (QUE). Examples of this type of dependence are the processes of the reporting and resolution times of insurance claims or the occurrence times of floods provoked by an event of intense rainfall.
- *Dependence between neighbour points in different processes*. It appears when the occurrence of an event in one process boots or blocks the occurrence of an event in the others. It can be modeled by a marked Poisson process with dependent marks generated, for example, by a Markov chain (MPP). This model yields medium or long-term dependence, since given that the process of all the points is a Poisson process, a model of rare events, the distance between consecutive points tends to be large. An example of this type of dependence is the process of the growth of a species of plant, which favors or avoids the growth of another plant during a period of time.

Common Poisson shock processes

The function `DepNHCPSP` generates d dependent Poisson processes, which are the marginal processes of a CPSP. A CPSP, see Abaurrea et al. (2015) for full details, is a multivariate PP with an underlying Poisson process of shocks, N_0, which may yield a point in one or more of d marginal processes N_j. These marginal processes are dependent Poisson processes, where the dependence between N_i and N_j only comes from the occurrence of simultaneous points in those processes.

Generation algorithm. The CPSPs show a property which straightforwardly leads to a generation algorithm: they can be decomposed into m independent Poisson processes, with $m = \sum_{k=1}^{d} \binom{d}{k}$. The m indicator processes are the processes of the points occurring only in N_1, ..., only in N_d, simultaneously only in N_1 and N_2, ..., simultaneously only in N_1, N_2, and N_3,..., and finally, simultaneously in N_1, N_2,... and N_d. For example, a CPSP with $d = 2$ is decomposed into three independent indicator processes, $N_{(1)}$, $N_{(2)}$, and $N_{(12)}$, with intensities $\lambda_{(1)}$, $\lambda_{(2)}$, and $\lambda_{(12)}$. Each marginal process N_j can be expressed as the sum of all the indicator processes including the index j, and its intensity is the sum of the indicator intensities. In the case $d = 2$, $N_1 = N_{(1)} + N_{(12)}$, $N_2 = N_{(2)} + N_{(12)}$, $\lambda_1 = \lambda_{(1)} + \lambda_{(12)}$, and $\lambda_2 = \lambda_{(2)} + \lambda_{(12)}$. Then, d dependent Poisson processes can be generated in two steps.

1. Generation of m independent Poisson processes $N_{(1)}$, $N_{(2)}$,..., $N_{(12)}$,... and $N_{(12...d)}$, with the adequate intensities, using the function `simNHPc`.
2. Each N_j is obtained as the union of the points in the indicator processes with index j, $N_{(.j.)}$.

The intensity of the processes to be generated with `DepNHCPSP` is specified in argument `lambdaiM`, a matrix whose columns are the intensity vectors of the indicator processes. Independent Poisson processes in the same period of time cannot be generated using `DepNHCPSP` but with `IndNHPP`.

Estimation. It is simple since it reduces to the identification of the indicator processes and the estimation of m independent Poisson processes. In the case $d = 2$, `CPSPpoints` identifies the three indicator processes, using as input the points in the two marginal processes. The related function `CPSPPOTevents` calculates the occurrence times, length, maximum, and mean intensity of the extreme events of the indicator processes of the CPSP resulting from a POT approach. The marginal and indicator processes of a CPSP are plotted by the functions `PlotMCPSP` and `PlotICPSP`, respectively. Poisson processes can be fitted to the indicator processes using the package **NHPoisson**.

Multivariate Neyman-Scott processes

The function `DepNHNeyScot` generates d dependent PPs, which are the marginal processes of an MNSP. A Neyman-Scott process (NSP) is a process of clusters of points such that the cluster centers C_i are a Poisson process, the number of points in each cluster, Z_i, are independent Poisson variables possibly with different means μ_i, and the distances of each point t_j in a cluster to its cluster center, D_{ij}, are i.i.d. variables. We call a multivariate NSP to a vector of NSP with the same cluster centers. The marginal processes are dependent processes, but they are not Poisson.

Generation algorithm. The previous definition leads to the following generation algorithm.

1. A Poisson process with a given intensity is generated to obtain the cluster centers C_i.
2. Given the number of generated cluster centers J, independent series of the number of points in each cluster, (Z_i^l) for $i = 1, \ldots, J$, for each marginal process l with $l = 1, \ldots, d$, are generated using Poisson distributions.
3. Given the series (Z_i^l), independent distances D_{ij}^l to the cluster center C_i for $j = 1, \ldots, Z_i^l$, $i = 1, \ldots, J$ are generated for each marginal process l. The points in each marginal process are obtained as $C_i + D_{ij}^l$.

`DepNHNeyScot` implements two common distributions to model the distances from the points to the cluster center, $N(0, \sigma)$ or $U(min, max)$. High values of σ or the range $(max - min)$ lead to a high variability around the center and to a lower dependence between the processes. Independent NSP in the same period of time cannot be generated using `DepNHNeyScot`, but with `IndNHNeyScot`.

This is the only model whose estimation is not easy, since the cluster centers are usually unobserved, and they are required to estimate both the underlying Poisson process and the distances of the points in each cluster to its cluster center.

Queue processes

`DepNHPPqueue` generates d dependent Poisson processes using $d - 1$ queues in a tandem. A queue models the input and output times of a customer in a waiting line. In a tandem of queues, the servers

are lined up one behind the other so that an arriving customer undergoes each server before leaving the system. `DepNHPPqueue` generates all the intermediate processes in a tandem where the first queue can be $M(t) \setminus G \setminus 1$ or $M(t) \setminus G \setminus \infty$, that is, a queue where the input is a nonhomogeneous Poisson process, service times have a general distribution G, and there are one or infinity servers.

Generation algorithm. The generation of homogeneous PPs is based on Burke's theorem stating that if the input of a queue is a homogeneous Poisson process, the output is a dependent Poisson process with the same intensity λ. Keilson and Servi (1994) stated that if the input is a nonhomogeneous Poisson process, the output is a Poisson process whose intensity is the convolution $\lambda_{out}(t) = \lambda(t) * g(t)$, where $g(t)$ is the density function of G. Then, the generation algorithm is:

1. Generation of the input process using a Poisson process with intensity $\lambda(t)$.
2. Generation of independent time services s_k with distribution G for each point i_k in the input process.
3. Generation of the output process using the generated input points and time services. If there is only one server, the output times o_k depend on the state of the queue: it is $o_k = o_{k-1} + s_k$ if the queue is not empty (that is if $o_{k-1} > i_k$), and $o_k = i_k + s_k$ otherwise. If there are infinity servers, there are no queues, and $o_k = i_k + s_k$.
4. The resulting output process is the input process of the following queue.

Steps 2 to 4 are repeated up to obtain d dependent Poisson processes.

The distribution G may be any distribution in **stats**; see `Distributions`. The length of the argument `lambda` fixes T, the length of the observed period, although in the homogeneous processes, we can fix the number of points to be generated instead (argument `nEv`). The vector of the output intensities $\lambda_{out}(t)$ is part of the output of the function. It is expected that low $\lambda(t)$ values and mean serving times lead to short queues and, consequently, high dependence between processes.

Estimation. Since the marginal processes are Poisson, they can be fitted and modeled using the package **NHPoisson**. Additionally, if the connection between the input and output points is known, the service times are the difference between them, and their distribution can be also easily estimated.

Marked Poisson processes with dependent marks

`DepNHPPMarked` generates d dependent processes, which are the marginal processes of an MPP with marks generated by a d-state Markov chain. An MPP is a Poisson process in which a variable, called a mark, is attached to each point. In this case, the marks are discrete variables taking values in $\{1, \ldots, d\}$, which determine in which of the d marginal processes N_j, a point occurs. Given the Markov chain structure, defined by a transition matrix $P = (p_{ij})$, only adjacent marks are dependent. The marginal processes are Poisson if and only if the marks are independent.

Generation algorithm. Applying the previous definition, the generation of algorithm is simple.

1. Generation of the points in a Poisson process with a given intensity $\lambda(t)$.
2. Generation of marks by a Markov chain. It implies an iterative generation of values in $1, \ldots, d$, given the previous mark, using a multinomial distribution with probabilities given by P.
3. Each marginal process N_j includes the points in the Poisson process with marks j.

`SpecGap` calculates the spectral gap, a measure of the dependence generated by a Markov chain, which assesses the convergence speed of the transition matrix to a matrix with the same stationary distribution and equal rows (that is, with independent marks). Processes with a lower spectral gap yield more dependent marginal processes. Independent Poisson processes can be generated using `IndNHPP` or a transition matrix with equal rows in `DepNHPPMarked`.

Estimation. Given that the process of all the points in the marginal processes is Poisson, $\lambda(t)$ can be estimated using **NHPoisson**. `TranM` estimates the transition matrix of the Markov chain using the MLE based on count data. Then, the estimators of the marginal intensities are $\hat{\lambda}_j(t) = \hat{\lambda}(t) \sum_{i=1}^{d} \hat{p}_{ij}$.

Inference based on computational statistical methods

There are many parameters of potential interest in a vector of point processes, where inference tools based on exact or asymptotic distributions are not available. Inference based on computational statistical methods such as Monte Carlo (MC) or parametric bootstrap is a useful alternative in those cases. `IntMPP` uses these methods to implement point estimation and calculation of confidence intervals and envelopes of a parameter, or vector of parameters, related to a vector of PPs. The only requirement

for the parameters of interest is that it must be possible to estimate them from the observed processes. Some examples are the vector of the number of points in each process in a given time period or the time of occurrence of the k-th point in the vector.

The idea to construct confidence intervals or envelopes using computational statistical methods is simple when the distribution of the vector of processes is completely known (Monte Carlo approach). In real problems, the parameters of the distribution of the vector of processes are rarely known, and parametric bootstrap methods, where the parameters are estimated from the sample, have to be used. The basic idea is to generate a sample of n_s vectors of processes with the distribution. A value of the statistic of interest is calculated from each generated vector so that a sample of size n_s of values of the statistic is obtained. The lower and upper bounds of an interval with a $(1-\alpha)\%$ confidence level are the $\alpha/2$ and $1-\alpha/2$ quantiles of the generated sample, and the point estimator is the sample mean. Standard tests of hypothesis can be implemented using those intervals in the usual way.

The two main arguments of `IntMPP` are `fun.name`, a function to define the estimator of the parameter, and `funMPP.name`, a model to generate the vectors of processes. The estimator in `fun.name` must be a function of the points in the vector of PPs (defined as a list which must be the first argument of the function) and any number of additional arguments provided by argument `fun.args`. The models in `funMPP.name` can be `DepNHCPSP`, `DepNHPNeyScot`, `DepNHqueue`, and `DepNHPPMarked`, or any other implemented by the user. The only requirement of those models is that the first element in the output has to be a list with d elements defining the vector of PPs. Additional arguments for the models are given in `funMPP.args`. Parallel computation is implemented in this function.

Analysis of the occurrence of extreme heat events in three locations using IndTestPP

This section illustrates how the package **IndTestPP** can be used to carry out all the steps in the analysis of the occurrence of the extreme heat events (EHEs) in three Spanish locations, Barcelona (B), Zaragoza (Z), and Huesca (H) using a vector of point processes.

Data

The series TxB, TxH, and TxZ are the daily maximum temperature, in Celsius degrees, during the warm season (May to September) from 1951 to 2016 at Barcelona, Huesca, and Zaragoza, respectively. The series were provided by the Spanish Meteorological Office (AEMET), and they are stored in the data frame $TxBHZ$ in the data set `TxBHZ`, available in the package. The days which are not observed in the three series are considered as missing observations so that three series with 8262 complete observations are available. The date (*day*, *month*, and *year*), day within year (*dayyear*) of the observations, and some variables representing the general temperature evolution are also available in the data set. The three locations are sited in a triangle where Barcelona is in the East, around 250km away from the others, and Huesca is 67 km to the North of Zaragoza.

Using the peak over threshold (POT) approach, an EHE is defined as a run of consecutive days where the temperature is over an extreme threshold, and its occurrence point is the day of maximum temperature in the run. The threshold is the 95^{th} percentile of the series in a reference period (months of June, July, and August in 1981-2010), being 31.3, 36.4, and 37.8° C in Barcelona, Huesca, and Zaragoza, respectively. The series are recorded at a discrete time scale, but given that the time unit is short compared with the length of the observed period and that the occurrence intensity of EHEs is quite low, the use of the continuous point processes to model the occurrence of EHEs in a series is justified. The EHEs may affect the three locations depending on the type of atmospheric situations that caused them. Then, our aim is to model the occurrence of the EHEs in the three series using a vector of point processes which take into account the dependence between them and identify the factors causing that dependence.

Processing data and preliminary analysis

To identify the occurrence times of the EHEs in a series using the POT approach, the function `POTevents.fun` in **NHPoisson** is used. The case of Zaragoza is shown as an example, and their 104 occurrence times are stored in *posZ*. Analogously, the 106 and 121 occurrences in Barcelona and Huesca are stored in *posB* and *posH*.

```
R> library(NHPoisson)
R> library(IndTestPP)
R> data(TxBHZ)
```

```
R> attach(TxBHZ)

R> auxZ<-POTevents.fun(TxZ, thres=37.8)

  Number of events:  104
  Number of excesses over threshold 37.8 : 176

R>  posZ<-auxZ$Px
```

Then, `PlotMargP` is used to plot the points in the three processes:

```
R> T<-length(TxZ)
R> PlotMargP(list(posB, posH, posZ), T=T, cex.axis=0.6,cex=0.6,
             cex.main=0.7, cex.lab=0.7)
```

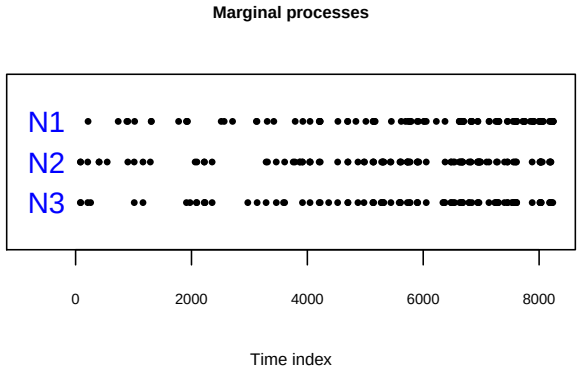


Figure 2: Plot from `PlotMargP`: Point processes of the occurrences times of the EHEs in Barcelona ($N1$), Huesca ($N2$), and Zaragoza ($N3$).

The temperature series are highly correlated, with Pearson coefficients $\rho_{BH} = 0.76$, $\rho_{BZ} = 0.73$, and $\rho_{HZ} = 0.94$, but to measure their extremal dependence, a more specific measure, such as the extremal dependence coefficients are used. The functions for the analysis between TxZ and TxB are shown as an example, but the pairwise dependence between the three locations is analyzed.

```
R> aux<-depchi(TxB,TxZ,indgraph=FALSE,xlegend='topright',
               thresval=c(9000:9975)/10000)
```

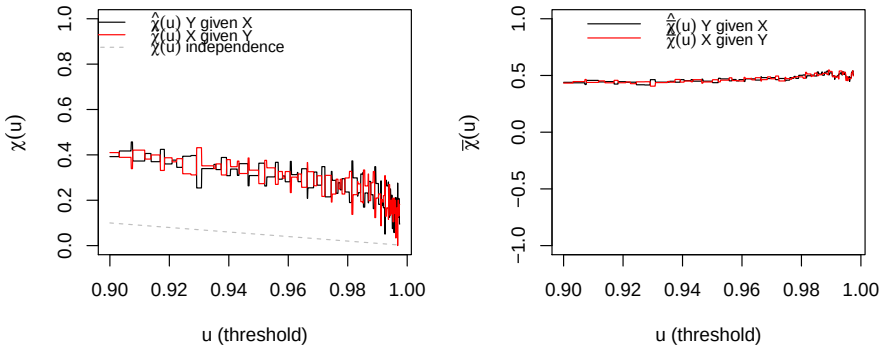


Figure 3: Plots from `depchi` of $\hat{\chi}_{x|y}(u)$ and $\hat{\bar{\chi}}_{x|y}(u)$ to estimate the extremal dependence coefficients.

The estimators $\hat{\chi}_{B|Z} = \hat{\chi}_{Z|B} = 0$ and $\hat{\bar{\chi}}_{B|Z} = \hat{\bar{\chi}}_{Z|B} = 0.5$ suggest an asymptotic independence between TxB and TxZ. However, $\hat{\bar{\chi}}_{Z|B} > 0$ suggests dependence at extreme levels, in particular in the threshold $\hat{\chi}_{Z|B}(0.95) \approx 0.38$. Similar conclusions are obtained for TxB and TxH, while TxH and TxZ are asymptotically dependent, with $\hat{\chi}_{H|Z} = \hat{\chi}_{Z|H} = 0.5$, $\hat{\bar{\chi}}_{Z|H} = \hat{\bar{\chi}}_{H|H} = 1$, and $\hat{\chi}_{Z|H}(0.95) \approx 0.7$.

The functions `CountingCor` and `BinPer` calculate another extremal dependence measures, the correlation coefficient between the number of EHEs in intervals of a given length ll, and the percentage of concordant intervals,

```
R> aux<-CountingCor(posB,posZ, ll=10, T=T, method='kendall')
R> aux

  tau
  0.3554213

R> aux<-BinPer(posB,posZ, ll=10, T=T)

  Percentage of concordant intervals:  0.272
```

with $ll = 10$ days, to measure short-term dependence. All the correlations, $\varrho^{10}_{BZ} = 0.36$, $\varrho^{10}_{BH} = 0.33$, and $\varrho^{10}_{ZH} = 0.68$ are significantly different from zero. Similar conclusions are provided by the percentage of concordant intervals, $BP^{10}_{BZ} = 0.27$, $BP^{10}_{BH} = 0.25$, and $BP^{10}_{ZH} = 0.55$.

The dependence given the empirical intensity of one process (obtained by function `emplambda.fun` in **NHPoisson**) can be graphically analyzed using the Dutilleul plot. Figure 4 shows the plot for Zaragoza-Barcelona, resulting from the following commands, and the plots for Barcelona-Huesca and Zaragoza-Huesca. All the previous results and the plots show that there exists a pairwise dependence between the three locations and that it is stronger between Zaragoza and Huesca.

```
R> lambdaEB<-emplambda.fun(posE=posB, t=c(1:T), lint=100, plot=F)$emplambda
R> aux<-DutilleulPlot(posZ, posB, lambdaEB, main="Zaragoza-Barcelona")
```

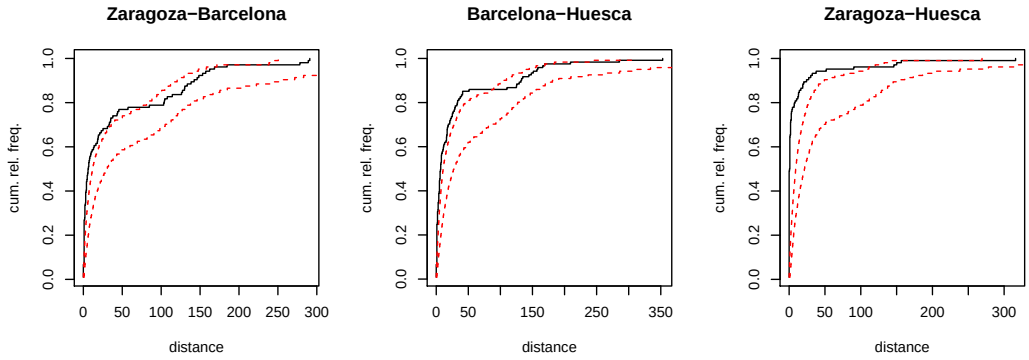


Figure 4: Dutilleul plot between the pairs of the EHE processes in the three locations, given the empirical intensities.

Testing independence and analyzing dependence factors

Our next aim is to identify the factors which cause the dependence. To that end, the independence tests given the marginal intensities are applied. The first step is to model each process individually. This has a twofold objective: first, to identify the factors that influence the occurrence of EHEs in each series, which may cause the dependence, and second to estimate the marginal intensities of the processes. The second step is to check if the occurrence processes are independent given the fitted intensities. If the tests do not reject the null hypothesis, it can be concluded that the dependence between the EHE processes is explained by the considered covariates since once its effect is removed, the processes are independent. The rejection of independence gives evidence that there are other non-identified factors causing dependence, which have not been included as predictors in the intensities. In those cases, a multivariate model allowing dependence should be used.

Step 1. To model the occurrence of the EHEs in each series, we consider a nonhomogeneous Poisson process with an intensity that is a function of a harmonic term (to model the seasonal behavior) and the available covariate, which represents the local atmospheric situation (Abaurrea et al., 2015). After the modeling process, based on a likelihood ratio test, the harmonic term, the covariate, and the squared covariate are selected in Zaragoza. The same terms are included in Huesca, and the same plus interaction between the covariate and the harmonic in Barcelona. These models are fitted using `fitPP.fun` in **NHPoisson**. The fit of Zaragoza is shown as an example, and the others are carried analogously to obtain *lambdaH* and *lambdaB*.

```
R> ss<-sin(2*pi*dayyear/366)
R> cc<-cos(2*pi*dayyear/366)
```

```
R> covZ<-cbind(ss,cc, Txm15Z, Txm15Z**2 )
R> dimnames(covZ)<-list(NULL, c("Sin", "Cos", "Txm15", "Txm152"))
R> ModZ<-fitPP.fun(covariates = covZ, posE = posZ,  inddat = auxZ$inddat,
               dplot=F, tit = "Sin+Cos+Txm15+Txm152",
               start = list(b0 = 1, b1=-1,b2=1, b3=0, b4=0))

  Number of observations  not used in the estimation process:  72
  Total number of time observations:  8262
  Number of events:  104
  Convergence code:  0
  Convergence attained
  Loglikelihood:  -430.087

  Estimated coefficients:
         b0      b1      b2      b3      b4
    -54.209   0.190  -2.496   2.434  -0.029
  Full coefficients:
         b0      b1      b2      b3      b4
    -54.209   0.190  -2.496   2.434  -0.029
  attr(,"TypeCoeff")
  [1] "Fixed: No  fixed parameters"

R> lambdaZ<-ModZ@lambdafit
```

The three fitted models are satisfactorily validated using `globalval.fun` in **NHPoisson**.

Step 2. The independence tests are used to study the pairwise independence given the fitted intensities. Since it can be assumed that the marginal processes are Poisson, the three families of tests POISSON, CLOSE, and CROSS can be applied. In the CLOSE family, only the PaB test is applied since the processes are nonhomogeneous; in the others, the most powerful test, according to Cebrián et al. (2020), is selected, that is the Normal test and the K test. Only the functions for the analysis between TxZ and TxB are shown, but all the pairwise comparisons are summarized in Table 1.

POISSON family. The Normal test is applied using an interval length $r = 15$ that guarantees the Normal approximation of the statistic.

```
R> aux<-CondTest(posZ, posB, lambday=lambdaB, r=15)

  WARNING: there are overlapping intervals. The independence hypothesis
           is not guaranteed.
  The intervals have been shortened to obtain disjoint  intervals.
  The length of the intersection priods are:
  [1] 23 21 28 19 11 12 27 26 20 22 15 27 22 18 18 22 28 16 26 25 17 26 12 26  6
  [26] 8 24 17 28 20 27 20 17 13 17 27 24 23 23 18 27  5 28 10  7 22 27 27 28 23
  [51] 27 26 24 21 19 26 14 14
  The shortest length of the considered intervals is:  3
  The median of the mui values is:  0.5

R> aux$pvN

  Normal p-value
       0.6859921
```

CLOSE family. In the PaB test, the parametric marginal model of the second process, the Poisson process fitted to Barcelona in this case, has to be specified.

```
R> PBZB<-TestIndNH(posZ, posB, nsim = 5000, type = "Poisson",
          lambdaMarg =cbind(lambdaB), fixed.seed=35)
R> PBZB$pv

  p-value
     0.2107578
```

CROSS family. The K test is implemented using an r-grid with values from 1 to 15, the value selected with the help of the plot of the estimated $K(r)$. Both the p-value and Figure 5 suggest the independence between Z-B given the intensities, since all the values $\hat{K}(r)$ lie inside the confidence band. On the other hand, the plot for Z-H, also shown in Figure 5, rejects independence for short-term dependence.

	Z-B			B-H			Z-H		
	Normal	PaB	K	Normal	PaB	K	Normal	PaB	K
pv	.69	.21	.0.16	.44	.25	.00 (0.29)	.03	.00	.03

Table 1: P-values of the tests to assess pairwise independence between the occurrence of EHEs in Zaragoza, Barcelona, and Huesca.

```
R> auxZB<-NHK(lambdaZ, lambdaB, posC=posZ, posD=posB, r=c(1:15),
         typePlot='Kfun', cores=2,fixed.seed=36)
R> auxZB$pv

  p-value
    0.1558442
```

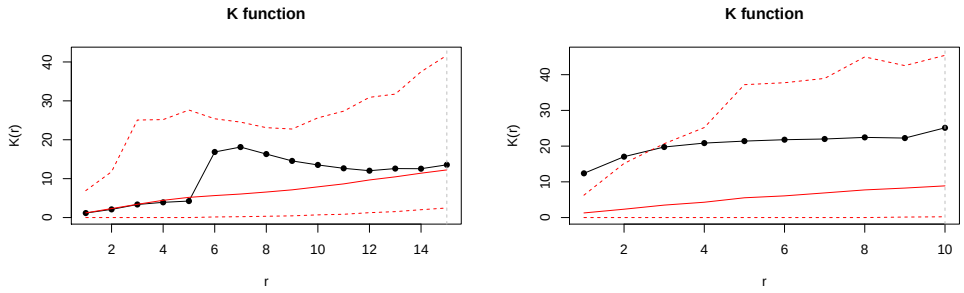


Figure 5: Plot from NHK: Estimation of the K function and confidence band under independence for Zaragoza-Barcelona (left) and Zaragoza-Huesca (right).

Table 1 summarizes the three pairwise comparisons. The three tests lead to the non-rejection of independence between the occurrences in B-Z, and to the rejection between Z-H. On the other hand, in pair B-H, the K test rejects the null while all the other tests do not. It is found that the high value of the K statistic is only due to the occurrence of a point in Huesca in $t = 901$ and in Barcelona in $t = 902$ when the intensity in both locations is low. In order to analyze the influence of this event, the point in Barcelona is removed, and the resulting p-value, 0.29, does not reject the independence anymore. This suggests that the K test is more sensitive than the others to the existence of an influential point. Given that all the tests are built by conditioning on the occurrences of the first process, the tests are also applied, changing the order of the locations, and the same conclusions are obtained.

These results are graphically confirmed by the Dutilleul plots given the fitted intensities, where only the plot between Zaragoza and Huesca gives evidence of dependence.

```
R< aux<-DutilleulPlot(posZ, posB, ModB@lambdafit,main="Barcelona-Zaragoza",
       cex.main=0.9)
```

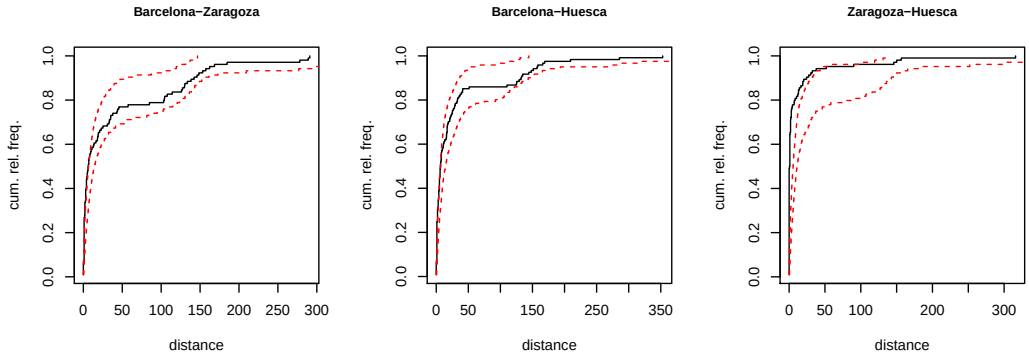


Figure 6: Dutilleul plot for the pairs of EHE processes given the fitted intensities.

The PaB test can also be used to test independence between the three processes simultaneously:

```
R> PBBHZ<-TestIndNH(posB, posH, posZ, nsim = 1000, type = "Poisson",
          lambdaMarg =cbind(lambdaH, lambdaZ), fixed.seed=65, cores=2)
R< PBBHZ$pv

  p-value
     0.002997003
```

Then, we conclude that, given the fitted intensities, the occurrence of the EHEs in Zaragoza-Barcelona and Barcelona-Huesca are independent, while there is dependence not explained by the covariates in Zaragoza-Huesca, which are the closest locations. Given these results, the best model for Barcelona is the previously fitted model, while the occurrence processes of Huesca and Zaragoza should be modeled by a vector of PPs taking into account the dependence between them. A model that allows us to include that dependence is a CPSP. The occurrences of the three indicator processes, the process of the events only in Huesca, only in Zaragoza, and the simultaneous events, are obtained by the function `CPSPPOTevents`. Then, the CPSP can be estimated by fitting a Poisson process to each of the three indicator processes using `fitPP.fun`; see Cebrián et al. (2015) for some examples.

Inference based on computational statistical methods

This section shows two examples of inference based on computational statistical tools using the function `IntMPP`. The first example uses the CPSP, which models the occurrence of EHEs in Huesca and Zaragoza, taking into account the dependence between them. It is fitted using **NHPoisson**, and the estimated intensities of the three indicator processes are the three last elements of the data.frame `TxBHZ`, `lambdaOZ`, `lambdaOH`, and `lambdaZH`.

In the first example, we calculate the point estimate and a confidence interval of the time of the first EHE in Zaragoza or Huesca. We need the function `firstt`, whose output is the minimum occurrence time in a vector of processes.

```
R> firstt<-function(posNH){minpos<-min(unlist(posNH))}
R> lambdaiZH<-cbind(lambdaOZ,lambdaOH,lambdaZH)
R> aux<-IntMPP(funMPP.name="DepNHCPSP",
       funMPP.args=list(lambdaiM=lambdaiZH, d=2, dplot=F),
       fun.name="firstt", fun.args=NULL, clevel=0.95, cores=2, fixed.seed=125)

  Lower  bound  of CI:  50.4648
  Point estimator:  116.7493
  Upper bound of CI:  233.4015
```

This type of inference also allows us to obtain confidence bands for two or more values, for example, the number of EHEs in Huesca and in Zaragoza in a given time interval I. To that end, we use the function `NumI`, included in the package, whose output is a vector containing the number of points in an interval I in each marginal process of a vector of processes. To see the evolution of the number of extremes, we consider two intervals, the three first and the three last years of the period. A clear increase in the number of EHEs is observed in the two locations.

```
R> aux<-IntMPP(funMPP.name="DepNHCPSP",
        funMPP.args=list(lambdaiM=lambdaiZH, d=2, dplot=F),
        fun.name="NumI", fun.args=list(I=c(1,459)),  fixed.seed=125)

  Lower  bound  of CI:  1 1
  Point estimator:  3.058 3.765
  Upper bound of CI:  6 7

R> aux<-IntMPP(funMPP.name="DepNHCPSP",
         funMPP.args=list(lambdaiM=lambdaiZH, d=2, dplot=FALSE),
         fun.name="NumI", fun.args=list(I=c(7803,8262)), fixed.seed=125)

  Lower  bound  of CI:  9 10
  Point estimator:  15.269 16.952
  Upper bound of CI:  22 24
```

Simulating and characterizing vectors of processes

In this section, some of the tools to generate vectors of processes in **IndTestPP** are used to characterize the effect of the dependence in the distribution of the nearest distances between two-point processes.

To that end, two dependent processes with a given dependence structure and two independent processes with the same marginal distribution that the previous ones are generated. The distributions of the samples of nearest distances are compared using histograms and qqplots.

We generate two dependent Neyman-Scott processes using `DepNHNeyScot`, with mean cluster size equal to 3 and 4, respectively, and $N(0,3)$ and $N(0,2)$ distributions for the distances to the center. The independent processes with the same marginal distribution are generated using `IndNHNeyScot`. The distribution of the nearest distances is very different in the two cases, as the qqplot shows. In the dependent processes, it is concentrated in low values, while in the independent ones the density decreases more smoothly.

```
R> set.seed(123)
R> lambdaParent<-runif(2000)/10

R> aux<-DepNHNeyScot(lambdaParent=lambdaParent, d=2, lambdaNumP=c(3,4),
       dist="normal", sigmaC=c(3,2),fixed.seed=123, dplot=F)
R> posxd<- aux$posNH$N1
R> posyd<- aux$posNH$N2

R> aux<-IndNHNeyScot(lambdaParent=lambdaParent, d=2, lambdaNumP=c(3,4),
        dist = "normal", sigmaC=c(3,2), fixed.seed=123, dplot=F)
R> posxi<- aux$N1
R> posyi<- aux$N2

R> par(mfrow=c(1,3))
R> distxyd<-nearestdist(posxd , posyd)
R> hist(distxyd , main='Dependent processes', xlab='Nearest dist',
       xlim=c(0,60), ylim=c(0,270),breaks=seq(0,60, by=4) )
R> distxyi<-nearestdist(posxi , posyi)
R> hist(distxyi , main='Independent processes', xlab='Nearest dist',
       xlim=c(0,60), ylim=c(0,270),breaks=seq(0,60, by=4) )
R> qqplot(distxyi, distxyd, xlab='Independent processes',
         ylab='Dependent processes')
R> lines(distxyd, distxyd, col="red")
```

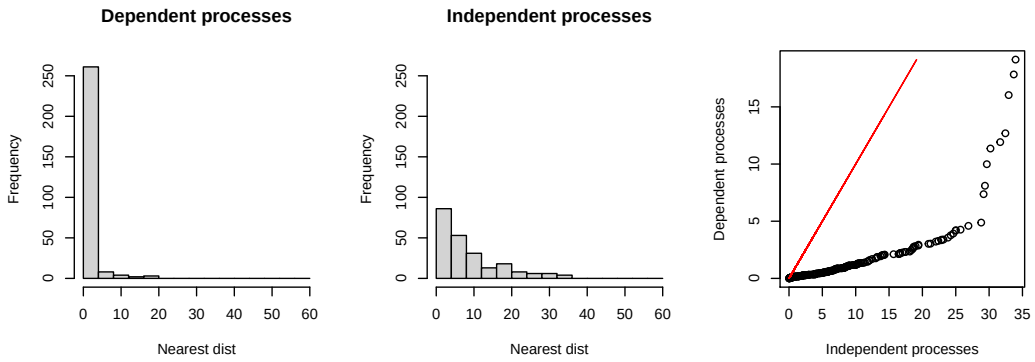


Figure 7: Histograms of nearest distance in two dependent and independent MNS processes and qqplot of the previous nearest distances.

Conclusions

Many modeling problems related to the occurrence of events require to analyze the dependence between two or more point processes in time. However, not many tools to carry out this type of analysis are available. **IndTestPP** provides a useful general framework for applications based on the modeling of a vector of point processes in time since it includes functions for processing data, estimating the marginal intensities of the processes, testing independence, identifying factors causing dependence, and making an inference. In particular, the three families of independence tests by Cebrián et al. (2020) are implemented. They are useful in different types of modeling problems since they cover a wide variety of processes, homogeneous and nonhomogeneous, Poisson processes,

processes with a parametric marginal model, point processes with known marginal intensities, etc. The package also provides functions to generate four different types of vectors of point processes, Common Poisson Shock processes, multivariate Neyman-Scott cluster processes, Poisson processes from queues in a tandem, and vectors of processes resulting from a marked Poisson process with discrete marks from a Markov chain. These generation functions are used to carry out inference based on computational statistical methods. The applicability of the package in real modeling problems is shown by analyzing the dependence between the occurrence of extreme temperature events in three Spanish locations, Zaragoza, Barcelona, and Huesca.

Acknowledgements

The authors are members of the research group Modelos Estocásticos (Gobierno de Aragón) and the project MTM2017-83812-P. They acknowledge J. Abaurrea and AEMET for the data and their advice.

Bibliography

J. Abaurrea, J. Asín, and A. C. Cebrián. Modeling and projecting the occurrence of bivariate extreme heat events using a non-homogeneous common Poisson shock process. *Stoch. Environ. Res. Risk. Assess.*, 29:309–322, 2015. [p499, 502, 505, 509]

M. Albert, Y. Bouret, M. Fromont, and P. Reynaud-Bouret. Bootstrap and permutation tests of independence for point processes. *Ann. Statist.*, 43(6):2537–2564, 2015. [p499]

A. Baddeley, E. Rubak, and R. Turner. *Spatial Point Patterns: Methodology and Applications with R*. Chapman & Hall/CRC Press, 2015. [p500]

A. C. Cebrián. *IndTestPP: Tests of Independence and Analysis of Dependence Between Point Processes in Time*, 2020. URL https://CRAN.R-project.org/package=IndTestPP. R package version 3.0. [p499]

A. C. Cebrián, J. Abaurrea, and J. Asín. NHPoisson: An R package for fitting and validating nonhomogeneous Poisson processes. *J. Stat. Softw.*, 64(6):1–25, 2015. [p500, 512]

A. C. Cebrián, J. Abaurrea, and J. Asín. Testing independence between two nonhomogeneous point processes in time. *Journal of Statistical Computation and Simulation*, 0(0):1–24, 2020. [p499, 501, 502, 510, 513]

S. Coles, J. Heffernan, and J. Tawn. Dependence measures for extreme value analysis. *Extremes*, 2(4): 339–65, 1999. [p504]

O. Cronie and M. N. M. van Lieshout. Summary statistics for inhomogeneous marked point processes. *Ann. Inst. Statist. Math.*, 68(4):905–928, 2016. [p499]

P. Dutilleul. *Spatio-temporal heterogeneity: Concepts and analyses*. CUP, 2011. [p503]

E. Gabriel, P. J. Diggle, B. Rowlingson, and F. J. Rodriguez-Cortes. *stpp: Space-Time Point Pattern Simulation, Visualisation and Analysis*, 2020. URL https://CRAN.R-project.org/package=stpp. R package version 2.0-4. [p500]

J. R. Giles, H. Salje, and J. Lessler. The IDSpatialStats R package: Quantifying spatial dependence of infectious disease spread. *The R Journal*, 11(2):308–327, 2019. [p500]

D. Harte. PtProcess: An R package for modelling marked point processes indexed by time. *Journal of Statistical Software*, 35(8):1–32, 2010. [p500]

H. Hino, K. Takano, Y. Yoshikawa, and N. Murata. *mmpp: Various Similarity and Distance Metrics for Marked Point Processes*, 2017. URL https://CRAN.R-project.org/package=mmpp. R package version 0.6. [p501]

J. Keilson and L. Servi. Networks of nonhomogeneous $M/G/\infty$ systems. *J. Appl. Probab.*, 31:157–68, 1994. [p506]

H. W. Lotwick and B. W. Silverman. Methods for analysing spatial processes of several types of points. *J.R. Statist. Soc. B*, 44:406–13, 1982. [p502]

M. Myllymäki, T. Mrkvicka, P. Grabarnik, H. Seijo, and U. Hahn. Global envelope tests for spatial processes. *J.R. Statist. Soc. B*, 79(2):381–404, 2017. [p499]

S. Ross. *Simulation*. Academic Press, 2006. [p502]

B. Rowlingson and P. Diggle. *splancs: Spatial and Space-Time Point Pattern Analysis*, 2017. URL https://CRAN.R-project.org/package=splancs. R package version 2.01-40. [p500]

P. Rubin-Delanchy and N. A. Heard. A test for dependence between two point processes on the real line, 2014a. [p499]

P. Rubin-Delanchy and N. A. Heard. *mppa: Statistics for analysing multiple simultaneous point processes on the real line*, 2014b. URL https://CRAN.R-project.org/package=mppa. R package version 1.0. [p501]

C. Tuleau-Malot, A. Rouis, F. Grammont, and P. Reynaud-Bouret. Multiple tests based on a Gaussian approximation of the unitary events method with delayed coincidence count. *Neural computation*, 26(7):1408–54, 2014. [p499]

Ana C. Cebrián
University of Zaragoza
Dpto. Métodos Estadísticos. Ed. Matemáticas. C Pedro Cerbuna, 12. Zaragoza 50009
Spain
ORCiD: 0000-0002-9052-9674
acebrian@unizar.es

Jesús Asín
University of Zaragoza
Dpto. Métodos Estadísticos. EINA. C María de Luna, 3. Zaragoza 50018
Spain

Conversations in Time: Interactive Visualization to Explore Structured Temporal Data

by Earo Wang and Dianne Cook

Abstract Temporal data often has a hierarchical structure, defined by categorical variables describing different levels, such as political regions or sales products. The nesting of categorical variables produces a hierarchical structure. The **tsibbletalk** package is developed to allow a user to interactively explore temporal data, relative to the nested or crossed structures. It can help to discover differences between category levels, and uncover interesting periodic or aperiodic slices. The package implements a shared `tsibble` object that allows for linked brushing between coordinated views, and a shiny module that aids in wrapping timelines for seasonal patterns. The tools are demonstrated using two data examples: domestic tourism in Australia and pedestrian traffic in Melbourne.

Introduction

Temporal data typically arrives as a set of many observational units measured over time. Some variables may be categorical, containing a hierarchy in the collection process, that may be measurements taken in different geographic regions, or types of products sold by one company. Exploring these multiple features can be daunting. Ensemble graphics (Unwin and Valero-Mora, 2018) bundle multiple views of a data set together into one composite figure. These provide an effective approach for exploring and digesting many different aspects of temporal data. Adding interactivity to the ensemble can greatly enhance the exploration process.

This paper describes new software, the **tsibbletalk** package, for exploring temporal data using linked views and time wrapping. We first provide some background to the approach based on setting up data structures and workflow, and give an overview of interactive systems in R. The section following introduces the **tsibbletalk** package. We explain the mechanism for constructing interactivity, to link between multiple hierarchical data objects and hence plots, and describe the set up for interactively slicing and dicing time to wrap a series on itself to investigate periodicities.

Background: tidy temporal data and workflow

The **tsibble** package (Wang et al., 2020) introduced a unified temporal data structure, referred to as a `tsibble`, to represent time series and longitudinal data in a tidy format (Wickham, 2014). A `tsibble` extends the `data.frame` and `tibble` classes with the temporal contextual metadata: `index` and `key`. The `index` declares a data column that holds time-related indices. The `key` identifies a collection of related series or panels observed over the `index`-defined period, which can comprise multiple columns. An example of a `tsibble` can be found in the monthly Australian retail trade turnover data (`aus_retail`), available in the **tsibbledata** package (O'Hara-Wild et al., 2020c), shown below. The `Month` column holds year-months as the `index`. `State` and `Industry` are the identifiers for these 152 series, which form the `key`. Note that the column `Series ID` could be an alternative option for setting up the `key`, but `State` and `Industry` are more readable and informative. The `index` and `key` are "sticky" columns to a `tsibble`, forming critical pieces for fluent downstream temporal data analysis.

```
#> # A tsibble: 64,532 x 5 [1M]
#> # Key:       State, Industry [152]
#>   State               Industry                  `Series ID`    Month Turnover
#>   <chr>               <chr>                     <chr>          <mth>    <dbl>
#> 1 Australian Capital ~ Cafes, restaurants and cat~ A3349849A   1982 Apr      4.4
#> 2 Australian Capital ~ Cafes, restaurants and cat~ A3349849A   1982 May      3.4
#> 3 Australian Capital ~ Cafes, restaurants and cat~ A3349849A   1982 Jun      3.6
#> 4 Australian Capital ~ Cafes, restaurants and cat~ A3349849A   1982 Jul      4
#> 5 Australian Capital ~ Cafes, restaurants and cat~ A3349849A   1982 Aug      3.6
#> # ... with 64,527 more rows
```

In the spirit of tidy data from the **tidyverse** (Wickham et al., 2019), the **tidyverts** suite features `tsibble` as the foundational data structure, and helps to build a fluid and fluent pipeline for time series analysis. Besides **tsibble**, the **feasts** (O'Hara-Wild et al., 2020b) and **fable** (O'Hara-Wild et al.,

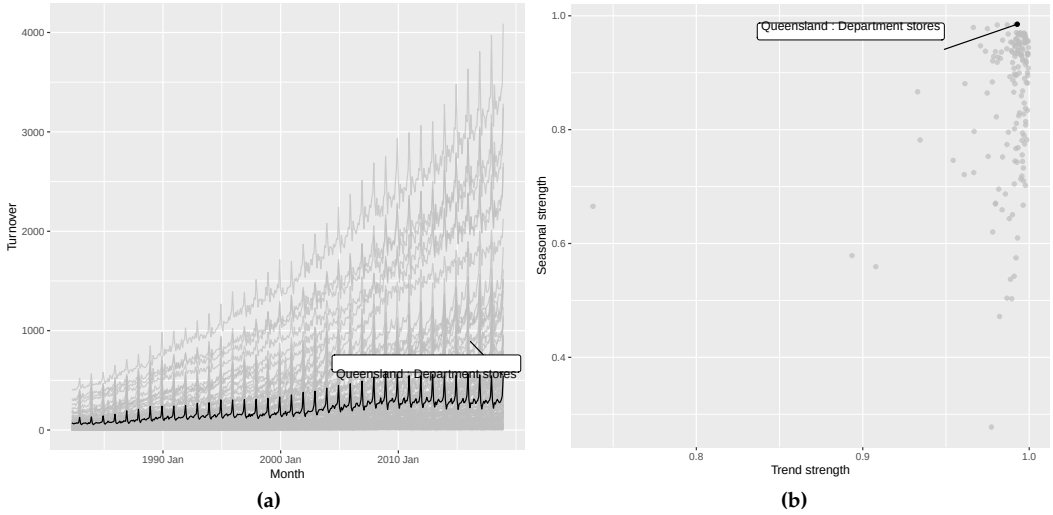


Figure 1: Plots for the `aus_retail` data, with the series of strongest seasonal strength highlighted. (a) An overlaid time series plot. (b) A scatter plot drawn from their time series features, where each dot represents a time series from (a).

2020a) packages fill the role of statistical analysis and forecasting in the **tidyverts** ecosystem. During all the steps of a time series analysis, the series of interest, denoted by the `key` variable, typically persist, through the trend modeling and also forecasting. We would typically want to examine the series across all of the keys.

Figure 1 illustrates examining temporal data with many keys. The data has 152 series corresponding to different industries in retail data. The multiple series are displayed using an overlaid time series plot, along with a scatterplot of two variables (trend versus seasonal strength) from feature space, where each series is represented by a dot. The feature space is computed using the `features()` function from **feasts**, which summarises the original data for each series using various statistical features. This function along with other **tidyverts** functions is `tsibble`-aware, and outputs a table in a reduced form where each row corresponds to a series, which can be graphically displayed as in Figure 1b.

Figure 1 has also been highlighted to focus on the one series with the strongest seasonality. To create this highlighting, one needs to first filter the interesting series from the features table, and join back to the original `tsibble` in order to examine its trend in relation to others. This procedure can soon grow cumbersome if many series are to be explored. It illustrates a need to query interesting series on the fly. Although these two plots are static, we can consider them as linked views because the common `key` variables link between the two data tables producing the two plots. This motivates the work in this package, described in this paper, to enable interactivity of `tsibble` and `tsibble`-derived objects for rapid exploratory data analysis.

Overview of interactivity

There is a long history of interactive data visualization research and corresponding systems. Within R, the systems can be roughly divided into systems utilizing web technology and those that do not.

R **shiny** (Chang et al., 2020) and **htmlwidgets** (Vaidyanathan et al., 2019) provide infrastructure connecting R with HTML elements and JavaScript that support the interactivity. The **htmlwidgets** package makes it possible to embed JavaScript libraries into R so that users are able to write only R code to generate web-based plots. Many JavaScript charting libraries have been ported to R as HTML widgets, including **plotly** (Sievert, 2020), **rbokeh** (Hafen and Continuum Analytics, Inc., 2020), and **leaflet** (Cheng et al., 2019) for maps. Interactions between different widgets can be achieved with **shiny** or **crosstalk** (Cheng, 2020). The **crosstalk** extends **htmlwidgets** with shared R6 instances to support linked brushing and filtering across widgets, without relying on **shiny**.

Systems without the web technology include **grDevices**, **loon** (Waddell and Oldford, 2020), based on Tcl/Tk, and **cranvas** (Xie et al., 2014) based on Qt. They offer a wide array of pre-defined interactions, such as selecting and zooming, to manipulate plots via mouse action, keyboard strokes, and menus. The **cranvastime** package (Cheng et al., 2016) is an add-on to **cranvas**, which provides specialized interactions for temporal data, such as wrapping and mirroring.

The techniques implemented in the work described in this paper utilize web technology, including **crosstalk**, **plotly**, and R **shiny**.

Using a shared temporal data object for interactivity

The **tsibbletalk** package introduces a shared tsibble instance built on a `tsibble`. This allows for seamless communication between different plots of temporal data. The `as_shared_tsibble()` function turns a `tsibble` into a shared instance, `SharedTsibbleData`, which is a subclass of `SharedData` from **crosstalk**. This is an R6 object driving data transmission across multiple views, due to its mutable and lightweight properties. The **tsibbletalk** package aims to streamline interactive exploration of temporal data, with the focus of temporal elements and structured linking.

Linking between plots

As opposed to one-to-one linking, **tsibbletalk** defaults to categorical variable linking, where selecting one or more observations in one category will broadcast to all other observations in this category. That is, linking is by key variables: within the time series plot, click on any data point, and the whole line will be highlighted in response. The `as_shared_tsibble()` uses `tsibble`'s `key` variables to achieve these types of linking.

The approach can also accommodate temporal data of nesting and crossing structures. These time series are referred to as hierarchical and grouped time series in the literature (Hyndman and Athanasopoulos, 2017). The `aus_retail` above is an example of grouped time series. Each series in the data corresponds to all possible combinations of the `State` and `Industry` variables, which means they are intrinsically crossed with each other. When one key variable is nested within another, such as regional areas within a state, this is considered to be a hierarchical structure.

The `spec` argument in `as_shared_tsibble()` provides a means to construct hybrid linking, that incorporates hierarchical and categorical linking. A symbolic formula can be passed to the `spec` argument, to define the crossing and/or nesting relationships among the key variables. Adopting Wilkinson and Rogers (1973)'s notation for factorial models, the `spec` follows the `/` and `*` operator conventions to declare nesting and crossing variables, respectively. The `spec` for the `aus_retail` data is therefore specified as `State * Industry` or `Industry * State`, which is the default for the presence of multiple `key` variables. If there is a hierarchy in the data, using `/` is required to indicate the parent-child relation, for a strictly one directional `parent/child`.

To illustrate nesting and crossing we use the `tourism_monthly` dataset (Tourism Research Australia, 2020) packaged in **tsibbletalk**. It contains monthly domestic overnight trips across Australia. The `key` is comprised of three identifying variables: `State`, `Region`, and `Purpose` (of the trip), in particular `State` nesting of `Region`, crossed together with `Purpose`. This specification can be translated as follows:

```
library(tsibble)
library(tsibbletalk)
tourism_shared <- tourism_monthly %>%
  as_shared_tsibble(spec = (State / Region) * Purpose)
```

There is a three-level hierarchy: the root node is implicitly Australia, geographically disaggregated to states, and lower-level tourism regions. A new handy function `plotly_key_tree()` has been implemented to help explore the hierarchy. It interprets hierarchies in the shared tsibble's `spec` as a tree view, built with **plotly**. The following code line produces the linked tree diagram (left panel of Figure 2). The visual for the tree hierarchy detangles a group of related series and provides a bird's eye view of the data organization.

```
p_l <- plotly_key_tree(tourism_shared, height = 1100, width = 800)
```

The tree plot provides the graphics skeleton, upon which the rest of the data plots can be attached. In this example, small multiples of line plots are placed at the top right of Figure 2 to explore the temporal trend across regions by the trip purpose. The shared tsibble data can be directly piped into **ggplot2** code to create this.

```
library(ggplot2)
p_tr <- tourism_shared %>%
  ggplot(aes(x = Month, y = Trips)) +
  geom_line(aes(group = Region), alpha = .5, size = .4) +
  facet_wrap(~ Purpose, scales = "free_y") +
  scale_x_yearmonth(date_breaks = "5 years", date_labels = "%Y")
```

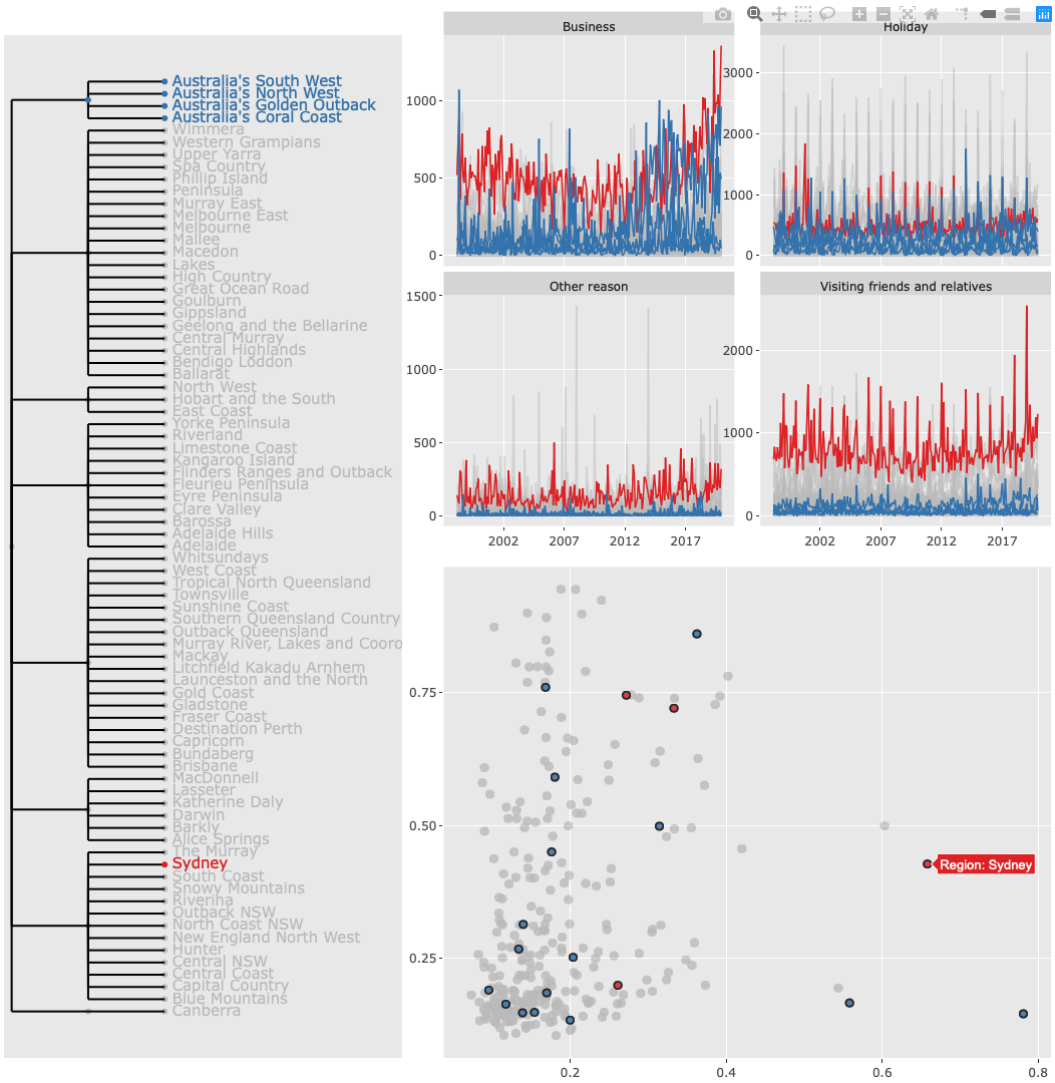


Figure 2: Snapshot of exploring an ensemble of linked plots of the Australian tourism data, built on a `tourism_shared` object. It also illustrates persistent linked brushing to compare two groups.

These line plots are heavily overplotted. To tease apart structure in the multiple time series, the `features()` function computes interesting characteristics, including the measures of trend and seasonality. These are displayed in the scatterplot at the bottom right, where one dot represents one series.

```
library(feasts)
tourism_feat <- tourism_shared %>%
  features(Trips, feat_stl)
p_br <- tourism_feat %>%
  ggplot(aes(x = trend_strength, y = seasonal_strength_year)) +
  geom_point(aes(group = Region), alpha = .8, size = 2)
```

There is one final step, to compose the three plots into an ensemble of coordinated views for exploration, shown in Figure 2. (This is the interactive realization of Figure 1).

```
library(plotly)
subplot(p_l,
  subplot(
    ggplotly(p_tr, tooltip = "Region", width = 1100),
    ggplotly(p_br, tooltip = "Region", width = 1100),
    nrows = 2),
  widths = c(.4, .6)) %>%
  highlight(dynamic = TRUE)
```

Since all plots are created from one shared tsibble data source, they are self-linking views. Nodes, lines, and points are hoverable and clickable. Given the spec, clicking either one element in any plot highlights all points that match the `Region` category, that is, categorical linking. Figure 2 is a static view of an interactive exploration. The steps in getting to this point were:

1. A branch of the tree corresponding to Western Australia was first selected. (The names of the regions are a little odd, which is a quirk of the data set, but all four areas, Australia's South West, ..., correspond to tourist destinations in Western Australia. Hovering over the node on the branch brings up the state name.) This generated the response in the line plots and the scatterplot that colored corresponding time series and points as blue.
2. To enable persistent selection, in oder to compare regions or states, "Shift" and click on the tree was done, after switching the color to red. This generated the response that points and time series corresponding to Sydney were highlighted in red.
3. Hovering over the points brings up the label for Sydney.

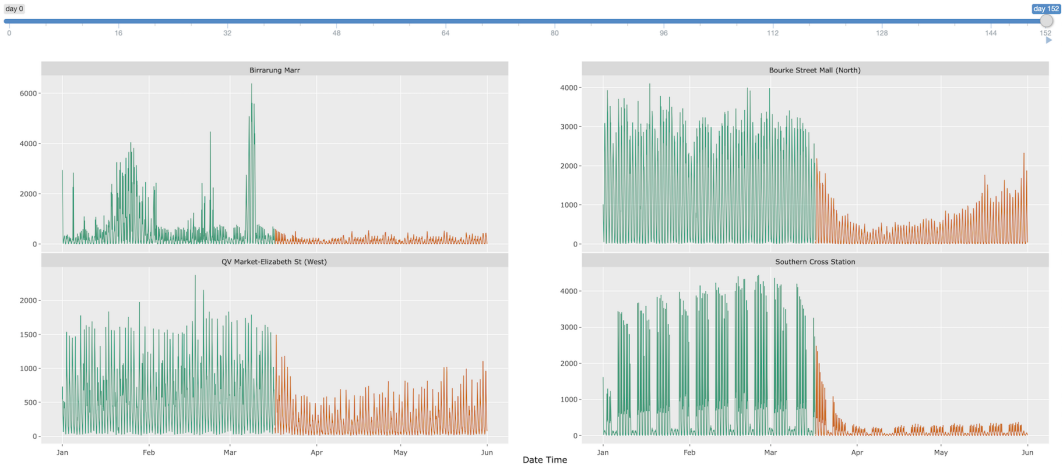
Domestic tourism sees Sydney as one of the most popular destinations in the realm of business and friends visiting over the years. Despite the relatively weaker performance in Western Australia, Australia's North West region sees a strongest upward trend in business, bypassing Sydney in some years.

In summary, shared tsibble data nicely bridges between the **crosstalk** and **tidyverts** ecosystems for temporal data using the common "key". The `as_shared_tsibble()` provides a symbolic user interface for the effortless construction of a hybrid of hierarchical and categorical linking between plots. The `plotly_key_tree()` function, in turn, decodes the hierarchical specification to plot a tree for data overview and navigation, when accompanied by more detailed plots.

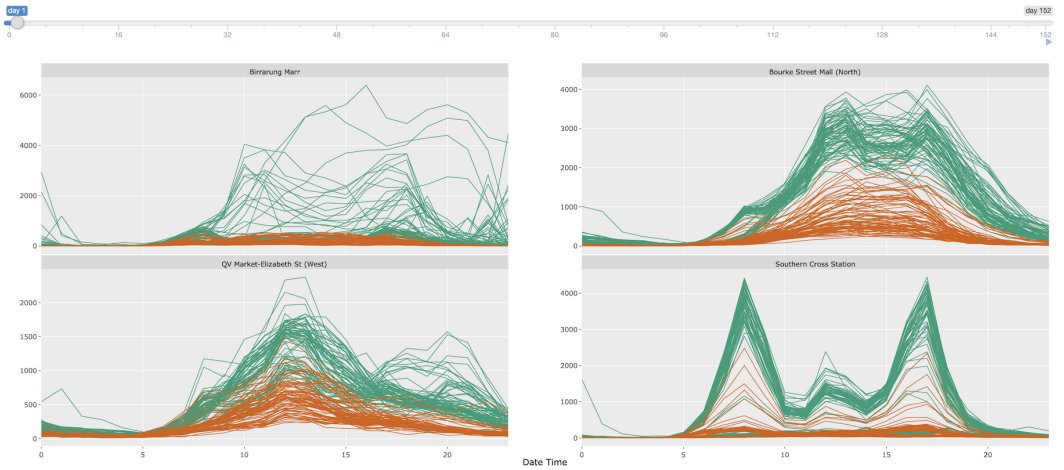
Slicing and dicing time

An important aspect of temporal data is the time context. Time has a cyclical structure, that may correspond to seasonal patterns to be discovered. The `index` component of the (shared) tsibble data forms the basis for exploring seasonality. To investigate for periodic or aperiodic patterns, series should be wrapped on themselves, where the index is broken into temporal components like quarter or day. We shall explore this with pedestrian traffic in Melbourne, Australia.

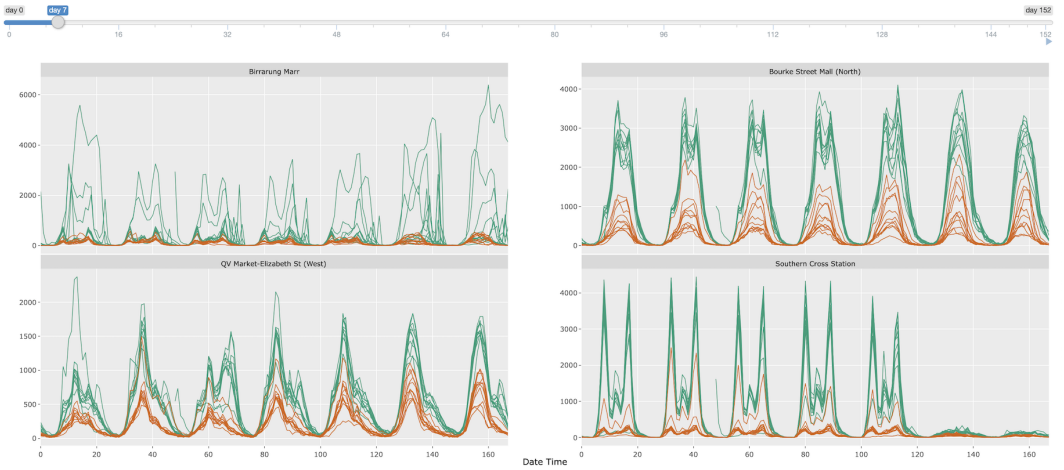
The city of Melbourne has sensors installed at various locations, to record hourly counts of pedestrians, in order to capture the daily rhythms of the downtown (City of Melbourne, 2020). Figure 3 shows the first five months of 2020 foot traffic at four different locations, for three different time slices, daily, weekly and full five months. Plot 3a shows hourly counts from January to May on an absolute timeline, facetted by locations. The stage 3 COVID-19 lockdown, on March 16, is marked by a change of color. (The pre-lockdown period is colored with dark green and lockdown with orange.) We can see a significant decline in foot traffic at all four locations. QV Market is less affected probably because this is a major produce market, an essential service that continued to operate. Bourke St, a



(a) Initial overview state



(b) 1-day state



(c) 7-day state, anchoring to Monday

Figure 3: Snapshots wrapping after slicing the `pedestrian20` data at different intervals, (a) none, (b) daily and (c) weekly. This type of interaction is made possible with Shiny elements.

major shopping center, sees a gradual uptick in the last weeks of the period indicating that people were getting back into the shops.

Figure 3b and 3c show the slicing and wrapping of the series into daily and weekly sections, respectively. Multiple seasonalities pop out. There tends to be a daily pattern, especially visible at the main train station, Southern Cross Station. There is also a weekday vs weekend pattern, also most visible at Southern Cross Station. These seasonal patterns are still present during the lockdown, but the magnitude is greatly reduced. Numbers are also down at the produce market and the shopping center. Birrarung Marr is the most affected. This is the location of special events, and it is clear that these have completely disappeared during the lockdown.

The wrapping procedure involves slicing the time index into seasonal periods of interest, and the result is diced time. For example, hourly pedestrian data can be decomposed into 24-hour blocks, which then overlays the counts for all respective days, as done in plot 3b. For exploration, this slice position should be controlled interactively, so that many different slices can be examined rapidly. This can be achieved using shiny, with the functions provided in the **tsibbletalk**.

This shiny module, decoupled to `tsibbleWrapUI()` and `tsibbleWrapServer()`, presents a clean interface and forms a reusable component that could be embedded in any shiny application. In general, a shiny module provides a vehicle for modularising shiny applications, relevant for both users and developers. As with all shiny modules, the first argument in both functions in **tsibbletalk** requires a user-supplied id string that must be unique. The UI function `tsibbleWrapUI()` simply shows a slider that animates or controls the number of periods to be diced. The workhorse is the server function `tsibbleWrapServer()`, encapsulating the algorithm that transforms data and sends messages to update the plot accordingly. The `plot` argument expects a `ggplot` or `plotly` object, where one can plot data using either lines or other graphical elements (such as boxplots). As the function name suggests, a (shared) tsibble is needed to start the engine, so that the time `index` can be retrieved for dissection. The `period` option semantically takes a desired number of seasonal periods to be shifted, for example data shifted by "1 day", "2 days", or "1 week", etc. In other words, the `period` defines the grind level. For date-times (represented by `POSIXt`), the granularity ranges from fine "day" to a much coarser "year". The following code snippet generates Figure 3. The creation of the `pedestrian20` data is available in supplementary R files.

```
library(shiny)
p_line <- pedestrian20 %>%
  ggplot(aes(x = Date_Time, y = Count, colour = Lockdown)) +
  geom_line(size = .3) +
  facet_wrap(~ Sensor, scales = "free_y") +
  labs(x = "Date Time") +
  scale_colour_brewer(palette = "Dark2") +
  theme(legend.position = "none")

ui <- fluidPage(
  tsibbleWrapUI("dice")
)
server <- function(input, output, session) {
  tsibbleWrapServer("dice", ggplotly(p_line, height = 700), period = "1 day")
}
shinyApp(ui, server)
```

Figure 3a corresponds to the initial state, with the slider incremented by 1-day units. The "play" button near the end of the slider can automatically animate the slicing and dicing process, walking the viewer through all 24 hours of the 152 days. Alternatively, users can drag the slider to examine selected slices.

In response to the slider input, the plot will be updated and loaded with newly transformed data. At its core, keeping the application as performant as possible is the top priority. Without completely redrawing the plot, the `plotlyProxy()` react method is invoked internally for talking to shiny. The underlying tsibble data is being called back and processed in R. Only transformed data gets fed back to the shiny server, for updating with resetting the x-axis ranges and breaks. The other plot configurations, such as marks, y-axes, and layouts, are cached and used as is.

The new shiny module exploits the temporal aspect for a tsibble object, available through the `index` attribute. It allows users to slide through relative periods to digest seasonal behaviors, with a nimble user experience.

Summary

At the heart of the **tsibbletalk** package is a blending of the best bits from **tsibble**, **crosstalk**, **plotly**, and **shiny**.

The `as_shared_tsibble()` turns a `tsibble` object to a shared data class, with an option to express any nesting and crossing structures from the `key` attribute. If nesting is found in the data, the `plotly_key_tree()` creates an interactive hierarchical tree to help with the data overview. This sets the stage for hierarchical and categorical linking between multiple views from one shared tsibble.

A new shiny module, `tsibbleWrapUI()` and `tsibbleWrapServer()`, provides a lens for looking at temporal aspects of a `tsibble`, in particular seasonal or cyclical variations. The slicing and dicing technique efficiently wrap time lines for user-defined plots. The `plotlyProxy()` react method makes it possible to send wrapped data to the server and amend the plot straight way.

Bibliography

W. Chang, J. Cheng, J. Allaire, Y. Xie, and J. McPherson. *shiny: Web Application Framework for R*, 2020. URL https://CRAN.R-project.org/package=shiny. R package version 1.5.0. [p517]

J. Cheng. *crosstalk: Inter-Widget Interactivity for HTML Widgets*, 2020. URL https://CRAN.R-project.org/package=crosstalk. R package version 1.1.0.1. [p517]

J. Cheng, B. Karambelkar, and Y. Xie. *leaflet: Create Interactive Web Maps with the JavaScript 'Leaflet' Library*, 2019. URL https://CRAN.R-project.org/package=leaflet. R package version 2.0.3. [p517]

X. Cheng, D. Cook, and H. Hofmann. Enabling interactivity on displays of multivariate time series and longitudinal data. *Journal of Computational and Graphical Statistics*, 25(4):1057–1076, 2016. ISSN 1061-8600, 1537-2715. URL https://www.tandfonline.com/doi/full/10.1080/10618600.2015.1105749. [p517]

City of Melbourne. *Pedestrian Volume in Melbourne*, 2020. URL http://www.pedestrian.melbourne.vic.gov.au. [p520]

R. Hafen and Continuum Analytics, Inc. *rbokeh: R Interface for Bokeh*, 2020. URL https://CRAN.R-project.org/package=rbokeh. R package version 0.5.1. [p517]

R. J. Hyndman and G. Athanasopoulos. *Forecasting: Principles and Practice*. OTexts, Melbourne, Australia, 2017. URL OTexts.org/fpp2. [p518]

M. O'Hara-Wild, R. Hyndman, and E. Wang. *fable: Forecasting Models for Tidy Time Series*, 2020a. URL https://CRAN.R-project.org/package=fable. R package version 0.2.1. [p516]

M. O'Hara-Wild, R. Hyndman, and E. Wang. *feasts: Feature Extraction and Statistics for Time Series*, 2020b. URL https://CRAN.R-project.org/package=feasts. R package version 0.1.5. [p516]

M. O'Hara-Wild, R. Hyndman, and E. Wang. *tsibbledata: Diverse Datasets for 'tsibble'*, 2020c. URL https://CRAN.R-project.org/package=tsibbledata. R package version 0.2.0. [p516]

C. Sievert. *Interactive Web-Based Data Visualization with R, plotly, and shiny*. Chapman and Hall/CRC, 2020. ISBN 9781138331457. URL https://plotly-r.com. [p517]

Tourism Research Australia. *Australian domestic overnight trips*, 2020. URL https://www.tra.gov.au. [p518]

A. Unwin and P. Valero-Mora. Ensemble Graphics. *Journal of Computational and Graphical Statistics*, 27 (1):157–165, 2018. ISSN 1061-8600, 1537-2715. URL https://www.tandfonline.com/doi/full/10.1080/10618600.2017.1383264. [p516]

R. Vaidyanathan, Y. Xie, J. Allaire, J. Cheng, and K. Russell. *htmlwidgets: HTML Widgets for R*, 2019. URL https://CRAN.R-project.org/package=htmlwidgets. R package version 1.5.1. [p517]

A. Waddell and R. W. Oldford. *loon: Interactive Statistical Data Visualization*, 2020. URL https://CRAN.R-project.org/package=loon. R package version 1.3.1. [p517]

E. Wang, D. Cook, and R. J. Hyndman. A new tidy data structure to support exploration and modeling of temporal data. *Journal of Computational and Graphical Statistics*, 29(3):466–478, 2020. doi: 10.1080/10618600.2019.1695624. [p516]

H. Wickham. Tidy data. *Journal of Statistical Software*, 59(10):1–23, 2014. [p516]

H. Wickham, M. Averick, J. Bryan, W. Chang, L. D. McGowan, R. François, G. Grolemund, A. Hayes, L. Henry, J. Hester, M. Kuhn, T. L. Pedersen, E. Miller, S. M. Bache, K. Müller, J. Ooms, D. Robinson, D. P. Seidel, V. Spinu, K. Takahashi, D. Vaughan, C. Wilke, K. Woo, and H. Yutani. Welcome to the tidyverse. *Journal of Open Source Software*, 4(43):1686, 2019. URL https://doi.org/10.21105/joss.01686. [p516]

G. N. Wilkinson and C. E. Rogers. Symbolic description of factorial models for analysis of variance. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 22(3):392–399, 1973. ISSN 00359254, 14679876. URL http://www.jstor.org/stable/2346786. [p518]

Y. Xie, H. Hofmann, and X. Cheng. Reactive programming for interactive graphics. *Statistical Science*, 29(2):201–213, 2014. ISSN 0883-4237. URL http://projecteuclid.org/euclid.ss/1408368571. [p517]

Earo Wang
The University of Auckland
Department of Statistics
earo.wang@auckland.ac.nz

Dianne Cook
Monash University
Department of Econometrics and Business Statistics
dicook@monash.edu

ROCnReg: An R Package for Receiver Operating Characteristic Curve Inference With and Without Covariates

by María Xosé Rodríguez-Álvarez and Vanda Inácio

Abstract This paper introduces the package **ROCnReg** that allows estimating the pooled ROC curve, the covariate-specific ROC curve, and the covariate-adjusted ROC curve by different methods, both from (semi) parametric and nonparametric perspectives and within Bayesian and frequentist paradigms. From the estimated ROC curve (pooled, covariate-specific, or covariate-adjusted), several summary measures of discriminatory accuracy, such as the (partial) area under the ROC curve and the Youden index, can be obtained. The package also provides functions to obtain ROC-based optimal threshold values using several criteria, namely, the Youden index criterion and the criterion that sets a target value for the false positive fraction. For the Bayesian methods, we provide tools for assessing model fit via posterior predictive checks, while the model choice can be carried out via several information criteria. Numerical and graphical outputs are provided for all methods. This is the only package implementing Bayesian procedures for ROC curves.

Introduction

The receiver operating characteristic (ROC) curve (Metz, 1978) is, unarguably, the most popular tool used for evaluating the discriminatory ability of continuous-outcome diagnostic tests. The ROC curve displays the false positive fraction (FPF) against the true positive fraction (TPF) for all possible threshold values that can be used to dichotomize the test result. The ROC curve thus provides a global description of the trade-off between the FPF and the TPF of the test as the threshold changes. Plenty of parametric and semi/nonparametric methods are available for estimating the ROC curve, either from frequentist or Bayesian viewpoints, and we refer the interested reader to Pepe (1998, Chapter 5), Zhou et al. (2011, Chapter 4), Inácio et al. (2020), and references therein.

It is known that in many situations, the outcome of a test and, possibly, its discriminatory capacity can be affected by covariates. Two different ROC-based measures that incorporate covariate information have been proposed: the covariate-specific or conditional ROC curve (see, e.g., Pepe, 2003, Chapter 6) and the covariate-adjusted ROC curve (Janes and Pepe, 2009). The formal definition of both curves is given in Section Notation and definitions. Succinctly, a covariate-specific ROC curve is an ROC curve that conditions on a specific covariate value, thus describing the accuracy of the test in the 'subpopulation' defined by that covariate value. On the other hand, the covariate-adjusted ROC curve is a weighted average of covariate-specific ROC curves. Regarding estimation, since the seminal paper of Pepe (1998), a plethora of methods have been proposed in the literature for the estimation of the covariate-specific ROC curve and associated summary measures. Without being exhaustive, we mention the work of Faraggi (2003), Rodríguez-Álvarez et al. (2011a,b), Inácio de Carvalho et al. (2013), and Inácio de Carvalho et al. (2017). A detailed review can be found in Rodríguez-Álvarez et al. (2011c), Pardo-Fernández et al. (2014), and Inácio et al. (2020). With respect to the covariate-adjusted ROC curve, estimation has been discussed in Janes and Pepe (2009), Rodríguez-Álvarez et al. (2011a), Guan et al. (2012), and Inácio de Carvalho and Rodríguez-Álvarez (2018).

A few R packages for ROC curve analysis are available on the Comprehensive R Archive Network and, as far as we are aware, all of them implementing frequentist approaches. The package **sROC** (Wang, 2012) contains functions to perform nonparametric, kernel-based, estimation of ROC curves. **pROC** (Robin et al., 2011) offers a set of tools to visualize, smooth, and compare ROC curves, and **nsROC** (Pérez Fernández et al., 2018) also allows estimating ROC curves, building confidence bands as well as comparing several curves both for dependent and independent data (i.e., data arising from paired and unpaired study designs, respectively). However, covariate information cannot be explicitly taken into account in any of these packages. The packages **ROCRegression** (available at `https://bitbucket.org/mxrodriguez/rocregression`) and **npROCRegression** (Rodriguez-Alvarez and Roca-Pardinas, 2017) provide routines to estimate semiparametrically and nonparametrically, under a frequentist framework, the covariate-specific ROC curve. We also mention **OptimalCutpoints** (López-Ratón et al., 2014) and **ThresholdROC** (Perez Jaume et al., 2017) that provide a collection of functions for point and interval estimation of optimal thresholds for continuous diagnostic tests. To the best of our knowledge, there is no statistical software package implementing Bayesian inference for ROC curves and associated summary indices and optimal thresholds.

To close this gap, in this paper we introduce the **ROCnReg** package that allows conducting

Method	Description
Pooled ROC curve	
emp	(Frequentist) empirical estimator (Hsieh and Turnbull, 1996).
kernel	(Frequentist) kernel-based approach (Zou et al., 1997).
BB	Bayesian bootstrap method (Gu et al., 2008).
dpm	Nonparametric Bayesian approach based on a Dirichlet process mixture of normal distributions (Erkanli et al., 2006).
Covariate-specific ROC curve	
sp	(Frequentist) parametric and semiparametric induced ROC regression approach (Pepe, 1998; Faraggi, 2003)
kernel	Nonparametric (kernel-based) induced ROC regression approach (Rodríguez-Álvarez et al., 2011a).
bnp	Nonparametric Bayesian model based on a single-weights dependent Dirichlet process mixture of normal distributions (Inácio de Carvalho et al., 2013).
Covariate-adjusted ROC curve	
sp	(Frequentist) semiparametric method (Janes and Pepe, 2009).
kernel	Nonparametric (kernel-based) induced ROC regression approach (Rodríguez-Álvarez et al., 2011a).
bnp	Nonparametric Bayesian model based on a single-weights dependent Dirichlet process mixture of normal distributions and the Bayesian bootstrap (Inácio de Carvalho and Rodríguez-Álvarez, 2018).

Table 1: Overview of ROC estimation methods included in the **ROCnReg** package.

Bayesian inference for the (pooled or marginal) ROC curve, the covariate-specific ROC curve, and the covariate-adjusted ROC curve. For the sake of generality, frequentist approaches are also implemented. Specifically, in what concerns estimation of the pooled ROC curve, **ROCnReg** implements the frequentist empirical estimator described in Hsieh and Turnbull (1996), the kernel-based approach proposed by Zou et al. (1997), the Bayesian Bootstrap method of Gu et al. (2008), and the Bayesian nonparametric method based on a Dirichlet process mixture of normal distributions model proposed by Erkanli et al. (2006). Regarding the covariate-specific ROC curve, **ROCnReg** implements the frequentist normal method of Faraggi (2003) and its semiparametric counterpart as described in Pepe (1998), the kernel-based approach of Rodríguez-Álvarez et al. (2011a), and the Bayesian nonparametric model based on a single-weights dependent Dirichlet process mixture of normal distributions proposed by Inácio de Carvalho et al. (2013). As for the covariate-adjusted ROC curve, the **ROCnReg** package allows estimation using the frequentist semiparametric approach of Janes and Pepe (2009), the frequentist nonparametric method discussed in Rodríguez-Álvarez et al. (2011a), and the recently proposed Bayesian nonparametric estimator of Inácio de Carvalho and Rodríguez-Álvarez (2018). Table 1 shows a summary of all methods implemented in the package. In addition, **ROCnReg** also provides functions to obtain ROC-based optimal thresholds to perform the classification/diagnosis of individuals as, say, diseased or nondiseased, using two different criteria, namely, the Youden index and the criterion that sets a target value for the false positive fraction. These are implemented for both the ROC curve, the covariate-specific, and the covariate-adjusted ROC curve.

The remainder of the paper is organized as follows. In Section Notation and definitions, we formally introduce the (pooled or marginal) ROC curve, the covariate-specific ROC curve, and the covariate-adjusted ROC curve. The description of the Bayesian estimation methods implemented in the **ROCnReg** package is given in Section Methods. In Section Package presentation and illustration, the usage of the main functions and capabilities of **ROCnReg** are described and illustrated using a synthetic dataset mimicking endocrine data. The paper concludes with a discussion in Section Summary and future plans.

Notation and definitions

This section sets out the formal definition of the pooled or marginal ROC curve, the covariate-specific ROC curve, and the covariate-adjusted ROC curve. It also describes the most commonly used summary measures of discriminatory accuracy, namely, the area under the ROC curve, the partial area under the ROC curve, and the Youden Index. For conciseness, we intentionally avoid giving too many details

and refer the interested reader to Pepe (2003) (and references therein) for an extensive account of many aspects of ROC curves with and without covariates.

In what follows, we denote as Y the outcome of the diagnostic test and as D the binary variable indicating the presence ($D = 1$) or absence ($D = 0$) of disease. We also assume that along with Y and the true disease status D, a covariate vector $\mathbf{X}$ is also available and that it may encompass both continuous and categorical covariates. For ease of notation, the covariate vector $\mathbf{X}$ is assumed to be the same in both the diseased ($D = 1$) and nondiseased ($D = 0$) populations, although this is not always necessarily the case in practice (e.g., disease stage is, obviously, a disease-specific covariate). By a slight abuse of notation, we use the subscripts D and $\bar{D}$ to denote (random) quantities conditional on, respectively, $D = 1$ and $D = 0$. For example, Y_D and $Y_{\bar{D}}$ denote the test outcomes in the diseased and nondiseased populations, respectively.

Pooled ROC curve

In the case of a continuous-outcome diagnostic test, the classification is usually made by comparing the test result Y against a threshold c. If the outcome is equal or above the threshold, $Y \geq c$, the subject will be diagnosed as diseased. On the other hand, if the test result is below the threshold, $Y < c$, he or she will be classified as nondiseased. The ROC curve is then defined as the set of all possible pairs of false positive fractions, $\text{FPF}(c) = \Pr(Y \geq c \mid D = 0) = \Pr(Y_{\bar{D}} \geq c)$, and true positive fractions, $\text{TPF}(c) = \Pr(Y \geq c \mid D = 1) = \Pr(Y_D \geq c)$, that can be obtained by varying the threshold value c, i.e.,

$$\{(\text{FPF}(c), \text{TPF}(c)) : c \in \mathbb{R}\}.$$

It is common to represent the ROC curve as $\{(p, \text{ROC}(p)) : p \in [0,1]\}$, where

$$p = \text{FPF}(c) = 1 - F_{\bar{D}}(c), \quad \text{ROC}(p) = 1 - F_D\left\{F_{\bar{D}}^{-1}(1-p)\right\}, \tag{1}$$

with $F_{\bar{D}}(y) = \Pr(Y_{\bar{D}} \leq y)$ and $F_D(y) = \Pr(Y_D \leq y)$ denoting the cumulative distribution function (CDF) of Y in the nondiseased and diseased groups, respectively. Several indices can be used as global summary measures of the accuracy of a test. The most widely used is the area under the ROC curve (AUC), defined as

$$\text{AUC} = \int_0^1 \text{ROC}(p)\, \mathrm{d}p. \tag{2}$$

In addition to its geometric definition, the AUC has also a probabilistic interpretation (see, e.g., Pepe, 2003, p. 78)

$$\text{AUC} = \Pr(Y_D \geq Y_{\bar{D}}), \tag{3}$$

that is, the AUC is the probability that a randomly selected diseased subject has a higher test outcome than that of a randomly selected nondiseased subject. The AUC takes values between 0.5, in the case of an uninformative test that classifies individuals no better than chance, and 1.0 for a perfect test. We note that an AUC below 0.5 simply means that the classification rule should be reversed. As it is clear from its definition, the AUC integrates the ROC curve over the whole range of FPFs. However, depending on the clinical circumstances, interest might lie only on a relevant interval of FPFs or TPFs, which leads to the notion of the partial area under the ROC curve (pAUC). The pAUC over a range of FPFs $(0, u_1)$, where u_1 is typically low and represents the largest acceptable FPF, is defined as

$$\text{pAUC}(u_1) = \int_0^{u_1} \text{ROC}(p)\, \mathrm{d}p. \tag{4}$$

On the other hand, the pAUC over a range of TPFs $(v_1, 1)$, where v_1 is typically large and represents the lowest acceptable TPF, is defined as

$$\text{pAUC}_{\text{TPF}}(v_1) = \int_{v_1}^1 \text{ROC}_{\text{TNF}}(p)\, \mathrm{d}p, \tag{5}$$

where ROC_{TNF} is a 270° rotation of the ROC curve, which can be expressed as

$$\text{ROC}_{\text{TNF}}(p) = F_{\bar{D}}\{F_D^{-1}(1-p)\}. \tag{6}$$

The curve in (6) is referred to as the true negative fraction (TNF) ROC curve since TNF ($= 1 - \text{FPF}$) is plotted on the y-axis. We shall highlight that the argument p in the ROC curve stands for a false positive fraction, whereas in the ROC_{TNF} curve, it stands for a true positive fraction. In Figure 1, we graphically illustrate the two partial areas.

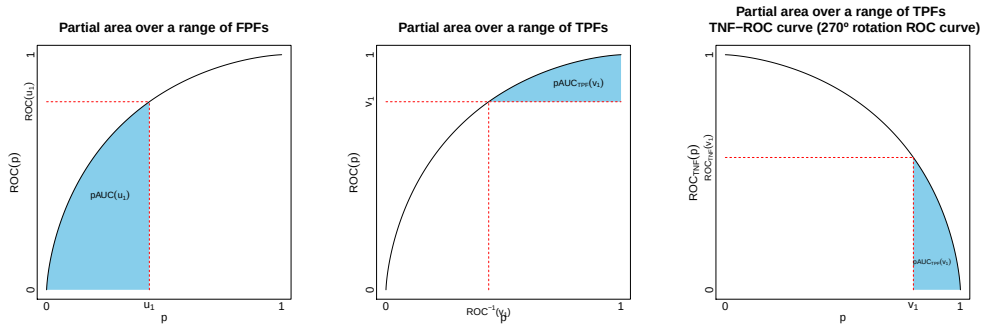


Figure 1: (a) Shaded area in blue represents the partial area under the ROC curve over the interval $(0, u_1)$ of FPFs. (b) Shaded area in blue represents the partial area under the ROC curve over the interval $(v_1, 1)$ of TPFs. (c) The same as in (b) but now represented as an area under the true negative fraction ROC curve.

Another summary index of diagnostic accuracy is the Youden index (Shapiro, 1999; Youden, 1950)

$$\begin{aligned} \text{YI} &= \max_c \{\text{TPF}(c) - \text{FPF}(c)\} && (7)\\ &= \max_c \{F_{\bar{D}}(c) - F_D(c)\} && (8)\\ &= \max_p \{\text{ROC}(p) - p\}. && (9) \end{aligned}$$

The YI ranges from 0 to 1, taking the value of 0 in the case of an uninformative test and 1 for a perfect test. As for the AUC, a YI below 0 means that the classification rule should be reversed. The value c^*, which maximizes Equation (7) (or, equivalently, Equation (8)), is frequently used in practice to classify subjects as diseased or nondiseased. It should be noted that the Youden index is equivalent to the Kolmogorov–Smirnov measure of distance between the distributions of Y_D and $Y_{\bar{D}}$ (Pepe, 2003, p. 80).

Covariate-specific ROC curve

The conditional or covariate-specific ROC curve, given a covariate value $\mathbf{x}$, is defined as

$$\text{ROC}(p \mid \mathbf{x}) = 1 - F_D\{F_{\bar{D}}^{-1}(1 - p \mid \mathbf{x}) \mid \mathbf{x}\}, \quad (10)$$

where $F_{\bar{D}}(y \mid \mathbf{x}) = \Pr(Y_{\bar{D}} \leq y \mid \mathbf{X}_{\bar{D}} = \mathbf{x})$ and $F_D(y \mid \mathbf{x}) = \Pr(Y_D \leq y \mid \mathbf{X}_D = \mathbf{x})$ are the conditional CDFs of the test in the nondiseased and diseased groups, respectively. In this case, a number of possibly different ROC curves (and therefore discriminatory accuracies) may be obtained for different values of $\mathbf{x}$. Thus, the covariate-specific ROC curve is an important tool that helps to understand and determine the optimal and suboptimal populations where to apply the tests on. That is, the covariate-specific ROC curve allows determining the populations, defined by or homogeneous with respect to $\mathbf{x}$, where the diagnostic test has a 'good' or 'poor' discriminatory capacity. Similarly to the unconditional case, the covariate-specific TNF-ROC curve is given by

$$\text{ROC}_{\text{TNF}}(p \mid \mathbf{x}) = F_{\bar{D}}\{F_D^{-1}(1 - p \mid \mathbf{x}) \mid \mathbf{x}\}, \quad (11)$$

and the covariate-specific AUC, pAUC, and Youden index are

$$\text{AUC}(\mathbf{x}) = \int_0^1 \text{ROC}(p \mid \mathbf{x})\text{d}p, \tag{12}$$

$$\text{pAUC}(u_1 \mid \mathbf{x}) = \int_0^{u_1} \text{ROC}(p \mid \mathbf{x})\text{d}p, \tag{13}$$

$$\text{pAUC}_{\text{TPF}}(v_1 \mid \mathbf{x}) = \int_{v_1}^1 \text{ROC}_{\text{TNF}}(p \mid \mathbf{x})\text{d}p, \tag{14}$$

$$\text{YI}(\mathbf{x}) = \max_c |\text{TPF}(c \mid \mathbf{x}) - \text{FPF}(c \mid \mathbf{x})| \tag{15}$$

$$= \max_c |F_{\bar{D}}(c \mid \mathbf{x}) - F_D(c \mid \mathbf{x})| \tag{16}$$

$$= \max_p |\text{ROC}(p \mid \mathbf{x}) - p|. \tag{17}$$

The value $c^*_{\mathbf{x}}$ that achieves the maximum in (15) (or (16)) is called the optimal covariate-specific YI threshold and can be used to classify a subject, with covariate value $\mathbf{x}$, as diseased or nondiseased.

Covariate-adjusted ROC curve

The covariate-specific ROC curve and associated AUC, pAUCs, and YI described in the previous section depict the accuracy of the test for specific covariate values. However, it would be undoubtedly useful to have a global summary measure that also takes covariate information into account. Such summary measure was developed by Janes and Pepe (2009), who proposed the covariate-adjusted ROC (AROC) curve, defined as

$$\text{AROC}(p) = \int \text{ROC}(p \mid \mathbf{x})\text{d}H_D(\mathbf{x}), \tag{18}$$

where $H_D(\mathbf{x}) = \Pr(\mathbf{X}_D \leq \mathbf{x})$ is the CDF of $\mathbf{X}_D$. That is, the AROC curve is a weighted average of covariate-specific ROC curves, weighted according to the distribution of the covariates in the diseased group. Equivalently, as shown by Janes and Pepe (2009), the AROC curve can also be expressed as

$$\begin{aligned}\text{AROC}(p) &= \Pr\{Y_D > F^{-1}_{\bar{D}}(1-p \mid \mathbf{X}_D)\} \\ &= \Pr\{1 - F_{\bar{D}}(Y_D \mid \mathbf{X}_D) \leq p\}.\end{aligned} \tag{19}$$

As will be seen in Section Methods, Expression (19) is very convenient when it comes to estimating the AROC curve. Also, it emphasizes that the AROC curve at an FPF of p is the overall TPF when the thresholds used for defining a positive test result are covariate-specific and chosen to ensure that the FPF is p in each subpopulation defined by the covariate values.

In contrast to the pooled ROC curve (see Expressions (1) and (6)) and the covariate-specific ROC curve (see Expressions (10) and (11)), the AROC curve (and its 270° rotation) cannot be expressed in terms of the (conditional) CDFs of the test in each group. This does not, however, preclude the possibility of defining AROC-based summary accuracy measures, yet more care is needed. Thus, for the AROC curve, the area under the AROC, as well as the partial areas and YI, are expressed as follows

$$\text{AAUC} = \int_0^1 \text{AROC}(p)\text{d}p, \tag{20}$$

$$\text{pAAUC}(u_1) = \int_0^{u_1} \text{AROC}(p)\text{d}p, \tag{21}$$

$$\text{pAAUC}_{\text{TPF}}(v_1) = \int_{\text{AROC}^{-1}(v_1)}^1 \text{AROC}(p)\text{d}p - \{1 - \text{AROC}^{-1}(v_1)\}v_1, \tag{22}$$

$$\text{YI}_{\text{AROC}} = \max_p \{\text{AROC}(p) - p\}. \tag{23}$$

Note, in particular, that the expressions for both the partial area under the AROC curve over a range of TPFs (see also Figure 1b) and for the YI are defined in terms of the AROC curve. For the YI, once the value that achieves the maximum in (23) is obtained, say p^*, covariate-specific threshold values can be calculated as follows

$$c^*_{\mathbf{x}} = F^{-1}_{\bar{D}}(1 - p^* \mid \mathbf{X}_D = \mathbf{x}).$$

Note that, by construction, these threshold values will ensure that the FPF is p^* in each subpopulation defined by the covariate values. However, the TPF may vary with the covariate values, i.e.,

$$\text{TPF}\left(c^*_{\mathbf{x}}\right) = 1 - F_D\left(c^*_{\mathbf{x}} \mid \mathbf{X}_D = \mathbf{x}\right).$$

To finish this part, we mention that when the accuracy of a test is not affected by covariates, this does not necessarily mean that the covariate-specific ROC curve (which, in this case, is the same for all covariate values) coincides with the pooled ROC curve. It does coincide, however, with the AROC curve (see Janes and Pepe, 2009; Pardo-Fernández et al., 2014; Inácio de Carvalho and Rodríguez-Álvarez, 2018, for more details). As such, in all cases where covariates affect the test results, even though they might not affect its discriminatory capacity, inferences based on the pooled ROC curve might be misleading. In such cases, the AROC curve should be used instead. This also applies to the selection of (optimal) threshold values, which might be covariate-specific (i.e., possibly different for different covariate values).

Methods

For space reasons, we focus ourselves here on the Bayesian methods for ROC curve inference (with and without covariates) implemented in the **ROCnReg** package. A detailed description, as well as usage examples, of the frequentist approaches are available as Supplementary Material at `https://bitbucket.org/mxrodriguez/rocnreg`.

Pooled ROC curve

In what follows, let $\{y_{\bar{D}i}\}_{i=1}^{n_{\bar{D}}}$ and $\{y_{Dj}\}_{j=1}^{n_D}$ be two independent random samples of test outcomes from the nondiseased and diseased groups of size $n_{\bar{D}}$ and n_D, respectively.

Bayesian bootstrap based estimator

The function `pooledROC.bb` implements the Bayesian bootstrap (BB) approach proposed by Gu et al. (2008). Their estimator relies on the notion of placement value (Pepe, 2003, Chapter 5), which is simply a standardization of the test outcomes with respect to a reference group. Specifically, $U_D = 1 - F_{\bar{D}}(Y_D)$ is to be interpreted as a standardization of a diseased test outcome with respect to the distribution of test results in the nondiseased population. The ROC curve can be regarded as the CDF of U_D

$$\Pr(U_D \leq p) = \Pr\{1 - F_{\bar{D}}(Y_D) \leq p\} = 1 - F_D\{F_{\bar{D}}^{-1}(1-p)\} = \text{ROC}(p), \quad 0 \leq p \leq 1. \tag{24}$$

The representation of the ROC given in (24) provides the rationale for the two-step algorithm of Gu et al. (2008), which can be described as follows. Let S be the number of iterations.

Step 1: Computation of the placement value based on the BB.
For $s = 1, \ldots, S$, let

$$U_{Dj}^{(s)} = \sum_{i=1}^{n_{\bar{D}}} q_{1i}^{(s)} I\left(y_{\bar{D}i} \geq y_{Dj}\right), \quad j = 1, \ldots, n_D,$$

where $\left(q_{11}^{(s)}, \ldots, q_{1n_{\bar{D}}}^{(s)}\right) \sim \text{Dirichlet}(n_{\bar{D}}; 1, \ldots, 1)$.

Step 2: Generate a realization of the ROC curve. Based on (24), generate a realization of $\text{ROC}^{(s)}(p)$, the cumulative distribution function of $(U_{D1}^{(s)}, \ldots, U_{Dn_D}^{(s)})$, where

$$\text{ROC}^{(s)}(p) = \sum_{j=1}^{n_D} q_{2j}^{(s)} I\left(U_{Dj}^{(s)} \leq p\right), \quad \left(q_{21}^{(s)}, \ldots, q_{2n_D}^{(s)}\right) \sim \text{Dirichlet}(n_D; 1, \ldots, 1).$$

The BB estimate of the ROC curve is obtained by averaging over the ensemble of ROC curves $\{\text{ROC}^{(1)}(p), \ldots, \text{ROC}^{(S)}(p)\}$, that is,

$$\widehat{\text{ROC}}^{\text{BB}}(p) = \frac{1}{S} \sum_{s=1}^{S} \text{ROC}^{(s)}(p),$$

and a $(1-\alpha) \times 100\%$ pointwise credible band can be obtained from the $\alpha/2 \times 100\%$ and $(1-\alpha/2) \times 100\%$ percentiles of the same ensemble ($\alpha \in (0,1)$). Note that these pointwise credible bands for the ROC curve are to be interpreted as credible intervals for the corresponding constituents TPFs.

The Bayesian bootstrap estimator leads to closed-form expressions for the AUC and pAUC, which

are, respectively, given by

$$\mathrm{AUC}^{(s)} = \int_0^1 \mathrm{ROC}^{(s)}(p)\mathrm{d}p = 1 - \sum_{j=1}^{n_D} q_{2j}^{(s)} U_{Dj}^{(s)},$$

$$\mathrm{pAUC}^{(s)}(u_1) = \int_0^{u_1} \mathrm{ROC}^{(s)}(p)\mathrm{d}p = u_1 - \sum_{j=1}^{n_D} q_{2j}^{(s)} \min\left\{u_1, U_{Dj}^{(s)}\right\}.$$

It is easy to show that

$$\mathrm{pAUC}_{\mathrm{TPF}}^{(s)}(v_1) = \int_{v_1}^1 \mathrm{ROC}_{\mathrm{TNF}}^{(s)}(p)\,\mathrm{d}p = \sum_{i=1}^{n_{\bar{D}}} q_{1i}^{(s)} \max\left\{v_1, U_{\bar{D}i}^{(s)}\right\} - v_1,$$

where

$$U_{\bar{D}i}^{(s)} = \sum_{j=1}^{n_D} q_{2j}^{(s)} I\left(y_{Dj} \geq y_{\bar{D}i}\right), \quad i = 1, \ldots, n_{\bar{D}},$$

and it is also easy to demonstrate that the $\mathrm{ROC}_{\mathrm{TNF}}$ curve is the survival function of the placement value $U_{\bar{D}} = 1 - F_D(Y_{\bar{D}})$. With respect to the Youden index, it is obtained by maximising, over a grid of possible threshold values, the following expression

$$\mathrm{YI}^{(s)} = \max_c \left\{F_{\bar{D}}^{(s)}(c) - F_D^{(s)}(c)\right\},$$

where

$$F_{\bar{D}}^{(s)}(c) = \sum_{i=1}^{n_{\bar{D}}} q_{1i}^{(s)} I\left(y_{\bar{D}i} \leq c\right) \quad \text{and} \quad F_D^{(s)}(c) = \sum_{j=1}^{n_D} q_{2j}^{(s)} I\left(y_{Dj} \leq c\right).$$

As for the ROC curve, point estimates for the AUC, pAUC, $\mathrm{pAUC}_{\mathrm{TPF}}$, YI, and c^* can be obtained by averaging over the respective ensembles of S realizations, with credible bands derived from the percentiles of such ensembles.

Dirichlet process mixture of normal distributions based estimator

The Bayesian nonparametric approach, based on a Dirichlet process mixture (DPM) of normal distributions, for estimating the pooled ROC curve (Erkanli et al., 2006) is implemented in the `pooledROC.dpm` function. In this case, as implicit by the name, the CDFs of the test outcomes in each group are estimated via a Dirichlet process mixture of normal distributions. That is, it is assumed that the CDF, say in the diseased group (the one in the nondiseased group, $\bar{D}$, follows analogously), is of the form

$$F_D(y) = \int \Phi(y \mid \mu, \sigma^2)\mathrm{d}G_D(\mu, \sigma^2), \qquad G_D \sim \mathrm{DP}(\alpha_D, G_D^*(\mu, \sigma^2)), \tag{25}$$

where $\Phi(y \mid \mu, \sigma^2)$ denotes the CDF of the normal distribution with mean μ and variance σ^2 evaluated at y. Here, $G_D \sim \mathrm{DP}(\alpha_D, G_D^*)$ is used to denote that the mixing distribution G_D follows a Dirichlet process (DP) (Ferguson, 1973) with centering distribution G_D^*, for which $E(G_D) = G_D^*$, and precision parameter α_D. Usually, due to conjugacy reasons, $G_D^*(\mu, \sigma^2) \equiv \mathrm{N}(\mu \mid m_{D0}, S_{D0})\Gamma(\sigma^{-2} \mid a_D, b_D)$, and this is the centering distribution used by the `pooledROC.dpm` function. Note that here, S_{D0} denotes the variance of the normal distribution, and a_D and b_D are, respectively, the shape and rate parameters of the gamma distribution. All hyperparameter values are fixed.

For ease of posterior simulation and because it provides a highly accurate approximation, we make use of the truncated stick-breaking representation of the DP (Ishwaran and James, 2001), according to which G_D can be written as

$$G_D(\cdot) = \sum_{l=1}^{L_D} \omega_{Dl} \delta_{(\mu_{Dl}, \sigma_{Dl}^2)}(\cdot),$$

where $(\mu_{Dl}, \sigma_{Dl}^2) \overset{\mathrm{iid}}{\sim} G_D^*(\mu, \sigma^2)$, for $l = 1, \ldots, L_D$, and the weights follow the so-called (truncated) stick-breaking construction: $\omega_{D1} = v_{D1}$, $\omega_{Dl} = v_{Dl}\prod_{r<l}(1 - v_{Dr})$, $l = 2, \ldots, L_D$, and $v_{D1}, \ldots, v_{D,L_D-1} \overset{\mathrm{iid}}{\sim} \mathrm{Beta}(1, \alpha_D)$. Further, one must set $v_{DL_D} = 1$ in order to ensure that the weights add up to one. The CDF in (25) can therefore be written as

$$F_D(y) = \sum_{l=1}^{L_D} \omega_{Dl} \Phi(y \mid \mu_{Dl}, \sigma_{Dl}^2),$$

where we shall note that L_D is not the exact number of components expected to be observed, but

rather an upper bound on it, as some of the components may be unoccupied. Some comments are in order regarding the specification of the hyperparameters' values. In what concerns the centering distribution, m_{D0} represents the prior belief about the components' means, and S_{D0} represents the confidence in such prior belief. Similarly, the values of a_D and b_D can be chosen to represent the prior belief about the components' variance. Of course, when setting these parameters, it is crucial to consider the measurement scale of the data. By default, test outcomes are standardized (so that the resulting mean is zero and the variance is one) in the `pooledROC.dpm` function and the default values are as follows

$$m_{D0} = 0, \quad S_{D0} = 10, \quad a_D = 2, \quad b_D = 0.5.$$

Because test outcomes are standardized, we expect the means of the components to be near zero and hence $m_{D0} = 0$. The parameter S_{D0} then controls where the drawn μ_{Dl} can lie, and the value of 10 implies that approximately 95% of the values roughly lie within -6 and 6. Further, note that $a_D = 2$ leads to a prior with an infinite variance that is centered around a finite mean ($b_D = 0.5$) and therefore favors variances less than one. Considering that the standardized data have a variance of one, it is reasonable to expect the within component variance to be smaller than the overall variance. The option of not standardizing the test outcomes is also available in `pooledROC.dpm`, and in such a case, the defaults for the centering distribution hyperparameters' values are as following

$$m_{D0} = \bar{y}_D, \quad S_{D0} = 100 s_D^2/n_D, \quad a_D = 2, \quad b_D = s_D^2/2,$$

with $\bar{y}_D = \frac{1}{n_D}\sum_{j=1}^{n_D} y_{Dj}$ and $s_D^2 = \frac{1}{n_D-1}\sum_{j=1}^{n_D}(y_{Dj} - \bar{y}_D)^2$. Regarding the precision parameter of the DP, α_D, it has a direct relationship with the number of occupied mixture components. One possible strategy for specifying α_D is to fix it to a small value to favor a small number of occupied components relative to the sample size. In the `pooledROC.dpm` function, we set $\alpha_D = 1$, a commonly used default value (Gelman et al., 2013, p. 553). Lastly, by default, $L_D = 10$. Before proceeding, we shall emphasize that these two configurations of hyperparameters values (for standardized and not standardized test outcomes) have proved to work well for a different range of test outcomes distributions, but it is certainly not our goal to encourage users to use it blindly and indeed thought should be dedicated to this important task. Nevertheless, output from the function `pooledROC.dpm` may be post-processed, and (informal) model fit diagnostics obtained; see more in Section Package presentation and illustration and in the Supplementary Materials.

Because the full conditional distributions for all model parameters are available in closed-form, posterior simulation can be easily conducted through Gibbs sampler (see the details, for instance, in Ishwaran and James 2002). At iteration s of the Gibbs sampler procedure, the ROC curve is computed as

$$\text{ROC}^{(s)}(p) = 1 - F_D^{(s)}\left\{F_{\bar{D}}^{-1(s)}(1-p)\right\}, \quad s = 1, \ldots, S,$$

with

$$F_D^{(s)}(y) = \sum_{l=1}^{L_D} \omega_{Dl}^{(s)} \Phi\left(y \mid \mu_{Dl}^{(s)}, \sigma_{Dl}^{2(s)}\right), \quad F_{\bar{D}}^{(s)}(y) = \sum_{k=1}^{L_{\bar{D}}} \omega_{\bar{D}k}^{(s)} \Phi\left(y \mid \mu_{\bar{D}k}^{(s)}, \sigma_{\bar{D}k}^{2(s)}\right), \tag{26}$$

and where the inversion is performed numerically. There is a closed-form expression for the AUC (Erkanli et al., 2006) given by

$$\text{AUC}^{(s)} = \sum_{k=1}^{L_{\bar{D}}}\sum_{l=1}^{L_D} \omega_{\bar{D}k}^{(s)}\omega_{Dl}^{(s)} \Phi\left(\frac{b_{kl}^{(s)}}{\sqrt{1 + a_{kl}^{2(s)}}}\right), \quad b_{kl}^{(s)} = \frac{\mu_{Dl}^{(s)} - \mu_{\bar{D}k}^{(s)}}{\sigma_{Dl}^{(s)}}, \quad a_{kl}^{(s)} = \frac{\sigma_{\bar{D}k}^{(s)}}{\sigma_{Dl}^{(s)}}.$$

Also, when $L_D = L_{\bar{D}} = 1$, there are closed-form expressions for the pAUC and pAUC_{TPF} which are used in the package (see Hillis and Metz, 2012). For the pAUC/pAUC_{TPF}, when $L_D > 1$ or $L_{\bar{D}} > 1$, the integrals are approximated numerically using Simpson's rule. The Youden index/optimal threshold is computed as in the Bayesian bootstrap method, with the obvious difference that here the CDFs are expressed as in (26). At the end of the sampling procedure, we have an ensemble of S ROC curves and AUCs/pAUCs/pAUC_{TPF}s/YIs/optimal thresholds, which, as before, allows obtaining point and interval estimates.

Covariate-specific ROC curve

We now let $\{(\mathbf{x}_{\bar{D}i}, y_{\bar{D}i})\}_{i=1}^{n_{\bar{D}}}$ and $\{(\mathbf{x}_{Dj}, y_{Dj})\}_{j=1}^{n_D}$ be two independent random samples of test outcomes and covariates from the nondiseased and diseased groups of size $n_{\bar{D}}$ and n_D, respectively. Further, for all $i = 1, \ldots, n_{\bar{D}}$ and $j = 1, \ldots, n_D$, let $\mathbf{x}_{\bar{D}i} = (x_{\bar{D}i,1}, \ldots, x_{\bar{D}i,q})^\top$ and $\mathbf{x}_{Dj} = (x_{Dj,1}, \ldots, x_{Dj,q})^\top$ be q-dimensional vectors of covariates, which can be either continuous or categorical.

The function `cROC.bnp` implements the Bayesian nonparametric approach for conducting inference

about the covariate-specific ROC curve of Inácio de Carvalho et al. (2013), which is based on a single-weights dependent Dirichlet process mixture of normal distributions (De Iorio et al., 2009). Specifically, under this method, the conditional CDF in the diseased group is modeled as follows

$$F_D(y_{Dj} \mid \mathbf{x}_{Dj}) = \int \Phi(y_{Dj} \mid \mu_D(\mathbf{x}_{Dj}, \boldsymbol{\beta}), \sigma^2) \mathrm{d}G_D(\boldsymbol{\beta}, \sigma^2), \qquad G_D \sim \mathrm{DP}(\alpha_D, G_D^*(\boldsymbol{\beta}, \sigma^2)),$$

with the conditional CDF in the nondiseased group $\bar{D}$ following in an analogous manner. As in the no-covariate case, by making use of Sethuraman's truncated representation of the DP, we can write the conditional CDF as

$$\begin{aligned} &F_D(y_{Dj} \mid \mathbf{x}_{Dj}) = \sum_{l=1}^{L_D} \omega_{Dl} \Phi(y_{Dj} \mid \mu_D(\mathbf{x}_{Dj}, \boldsymbol{\beta}_{Dl}), \sigma_{Dl}^2), \\ &\omega_{D1} = v_{D1}, \quad \omega_{Dl} = v_{Dl} \prod_{r<l} (1 - v_{Dr}), \quad l = 2, \ldots, L_D, \\ &v_{Dl} \overset{\text{iid}}{\sim} \mathrm{Beta}(1, \alpha_D), \quad l = 1, \ldots, L_D - 1, \quad v_{DL_D} = 1. \end{aligned}$$

It is worth mentioning that although the variance of each component does not depend on covariates, the overall variance of the mixture does depend on covariates, as it can be written as

$$\mathrm{var}(y_{Dj} \mid \mathbf{x}_{Dj}) = \sum_{l=1}^{L_D} \omega_{Dl} \sigma_{Dl}^2 + \sum_{l=1}^{L_D} \omega_{Dl} \left\{ \mu_D(\mathbf{x}_{Dj}, \boldsymbol{\beta}_{Dl}) - \left(\sum_{l=1}^{L_D} \omega_{Dl} \mu_D(\mathbf{x}_{Dj}, \boldsymbol{\beta}_{Dl}) \right)^2 \right\}.$$

Note that by assuming that the weights, w_{Dl}, do not vary with covariates, the model might have limited flexibility in practice (MacEachern, 2000). This issue can, however, be largely mitigated by using a flexible formulation for $\mu_D(\mathbf{x}_{Dj}, \boldsymbol{\beta}_{Dl})$, which is needed not only for the model to be able to recover nonlinear trends but also to recover flexible shapes that might arise due to a dependence of the weights on the covariates. As such, the function `cROC.bnp` in **ROCnReg** allows modeling the mean function of each component using an additive smooth structure

$$\mu_D(\mathbf{x}_{Dj}, \boldsymbol{\beta}_{Dl}) = \beta_{Dl0} + f_{Dl1}(x_{Dj,1}) + \ldots + f_{Dlq}(x_{Dj,q}), \qquad l = 1, \ldots, L_D, \tag{27}$$

where the smooth functions, f_{Dlm} $(m = 1, \ldots, q)$, are approximated using a linear combination of cubic B-splines basis functions. To avoid notational burden, we have assumed that all q covariates are continuous and modeled in a flexible way. However, the function `cROC.bnp` can also deal with categorical covariates, linear effects of continuous covariates, as well as interactions. For the reasons mentioned before, we recommend that all continuous covariates are modeled as in (27). Nonetheless, posterior predictive checks, as illustrated in Section Package presentation and illustration, can also be used to informally validate the fitted model. We write

$$\mu_D(\mathbf{x}_{Dj}, \boldsymbol{\beta}_{Dl}) = \mathbf{z}_{Dj}^\top \boldsymbol{\beta}_{Dl}, \quad l = 1, \ldots, L_D, \quad j = 1, \ldots, n_D, \tag{28}$$

where $\mathbf{z}_{Dj}^\top$ is the jth row of the design matrix that contains the intercept, the continuous covariates that are modeled in a linear way (if any), the cubic B-splines basis representation for those modeled in a flexible way, the categorical covariates (if any), and their interaction(s) (if believed to exist). Also, $\boldsymbol{\beta}_{Dl}$ collects, for the lth component, the regression coefficients associated with the aforementioned covariates. For the covariate effects modeled using cubic B-splines, an important issue is the selection of the number and location of the knots at which to anchor the basis functions, as this has the potential to impact inferences, more so for the former than the latter. The selection of the number of knots can be assisted by a model selection criterion, for example, (the adaptation to the case of mixture models of) the deviance information criterion (DIC) (Celeux et al., 2006), the log pseudo marginal likelihood (LPML) (Geisser and Eddy, 1979), and the widely applicable information criterion (WAIC) (Gelman et al., 2014). In turn, for the location of the interior knots themselves, we follow Rosenberg (1995) and use the quantiles of the covariate values.

The regression coefficients and variances associated with each of the L_D components are sampled from the conjugate centering distribution $(\boldsymbol{\beta}_{Dl}, \sigma_{Dl}^{-2}) \overset{\text{iid}}{\sim} \mathrm{N}_{Q_D}(\mathbf{m}_D, \mathbf{S}_D)\Gamma(a_D, b_D)$, with conjugate hyper-priors $\mathbf{m}_D \sim \mathrm{N}(\mathbf{m}_{D0}, \mathbf{S}_{D0})$ and $\mathbf{S}_D^{-1} \sim \mathrm{Wishart}(\nu_D, (\nu_D \Psi_D)^{-1})$ (a Wishart distribution with degrees of freedom ν_D and expectation Ψ_D^{-1}), and where Q_D is the dimension of the vector $\mathbf{z}_{Dj}$. Hyperparameters $\mathbf{m}_{D0}$ and Ψ_D must be chosen to represent the prior belief about the regression coefficients associated to each mixture component and about their covariance matrix, respectively, whereas $\mathbf{S}_{D0}$ and ν_D are chosen to represent the confidence in the prior belief of $\mathbf{m}_{D0}$ and Ψ_D, respectively. As in the no-covariate case, by default, in `cROC.bnp`, test outcomes and covariates are standardized, which not only facilitates specification of the hyperparameter values but also improves the mixing of the Markov

chain Monte Carlo (MCMC) chains. The default values are as follows

$$\mathbf{m}_{D0} = \mathbf{0}_{Q_D}, \quad \mathbf{S}_{D0} = 10 I_{Q_D}, \quad \nu_D = Q_D + 2, \quad \Psi_D = I_{Q_D}, \quad a_D = 2, \quad b_D = 0.5.$$

When test outcomes and covariates are not standardized, the defaults are the following

$$\mathbf{m}_{D0} = \widehat{\boldsymbol{\beta}}_D, \quad \mathbf{S}_{D0} = \widehat{\boldsymbol{\Sigma}}_D, \quad \nu_D = Q_D + 2, \quad \Psi_D = 30\widehat{\boldsymbol{\Sigma}}_D, \quad a_D = 2, \quad b_D = \widehat{\sigma}_D^2/2,$$

where $\widehat{\boldsymbol{\beta}}_D$ and $\widehat{\sigma}_D$ are the least squares estimates from fitting the linear model $y_{Dj} = \mathbf{z}_{Dj}\boldsymbol{\beta}_D + \sigma_D \varepsilon_{Dj}$, where $E(\varepsilon_{Dj}) = 0$, $\text{var}(\varepsilon_{Dj}) = 1$, and $\widehat{\boldsymbol{\Sigma}}_D$ is the estimated covariance matrix of $\widehat{\boldsymbol{\beta}}_D$. With regard to the specification of α_D and L_D, as in the DPM model (no-covariate case), we set them, respectively, to 1 and 10. The blocked Gibbs sampler is used to simulate draws from the posterior distribution, and details about it can be found, for instance, in the Supplementary Materials of Inácio de Carvalho et al. (2017).

Similarly to the analogous model for the no-covariate case, at iteration s of the Gibbs sampler procedure, the covariate-specific ROC curve is computed as

$$\text{ROC}^{(s)}(p \mid \mathbf{x}) = 1 - F_D^{(s)}\left\{F_{\bar{D}}^{-1(s)}(1-p \mid \mathbf{x}) \mid \mathbf{x}\right\}, \quad s = 1, \ldots, S,$$

with

$$F_D^{(s)}(y \mid \mathbf{x}) = \sum_{l=1}^{L_D} \omega_{Dl}^{(s)} \Phi\left(y \mid \mathbf{z}^\top \boldsymbol{\beta}_{Dl}^{(s)}, \sigma_{Dl}^{2(s)}\right), \quad F_{\bar{D}}^{(s)}(y \mid \mathbf{x}) = \sum_{k=1}^{L_{\bar{D}}} \omega_{\bar{D}k}^{(s)} \Phi\left(y \mid \mathbf{z}^\top \boldsymbol{\beta}_{\bar{D}k}^{(s)}, \sigma_{\bar{D}k}^{2(s)}\right), \tag{29}$$

and where the inversion is performed numerically. A point estimate for $\text{ROC}(p \mid \mathbf{x})$ can be obtained by computing the mean of the ensemble $\{\text{ROC}^{(1)}(p \mid \mathbf{x}), \ldots, \text{ROC}^{(S)}(p \mid \mathbf{x})\}$, with pointwise credible bands derived from the percentiles of the ensemble. Although the results presented in Erkanli et al. (2006) can be extended to derive a closed-form expression for the covariate-specific AUC, for computational reasons, in **ROCnReg**, the integral in (12) is approximated using Simpson's rule, and the same applies for the partial areas. Conditionally on a specific covariate value, the computation of the Youden index and of the optimal threshold proceeds in a similar way as in the DPM model (see Inácio de Carvalho et al., 2017 for details). As for the covariate-specific ROC curve, point and interval estimates can be obtained from the corresponding covariate-specific ensemble of each summary measure.

We finish this section by noting that a particular case of the above estimator arises when the effect of all continuous covariates is assumed to be linear and only one component is considered, i.e.,

$$F_D^{(s)}(y \mid \mathbf{x}) = \Phi\left(y \mid \tilde{\mathbf{x}}^\top \boldsymbol{\beta}_D^{(s)}, \sigma_D^{2(s)}\right), \quad \text{and} \quad F_{\bar{D}}^{(s)}(y \mid \mathbf{x}) = \Phi\left(y \mid \tilde{\mathbf{x}}^\top \boldsymbol{\beta}_{\bar{D}}^{(s)}, \sigma_{\bar{D}}^{2(s)}\right), \tag{30}$$

with $\tilde{\mathbf{x}}^\top = \left(1, \mathbf{x}^\top\right)$. In this case, it is easy to show that

$$\text{ROC}^{(s)}(p \mid \mathbf{x}) = 1 - \Phi\left\{a^{(s)}(\mathbf{x}) + b^{(s)}\Phi^{-1}(1-p)\right\}, \tag{31}$$

where

$$a^{(s)}(\mathbf{x}) = \tilde{\mathbf{x}}^\top \frac{\left(\boldsymbol{\beta}_{\bar{D}}^{(s)} - \boldsymbol{\beta}_D^{(s)}\right)}{\sigma_D^{(s)}}, \quad \text{and} \quad b^{(s)} = \frac{\sigma_{\bar{D}}^{(s)}}{\sigma_D^{(s)}}. \tag{32}$$

With this configuration, the model for the covariate-specific ROC curve can be regarded as a Bayesian counterpart of the induced ROC approach proposed by Faraggi (2003) (and detailed in the Supplementary Material). We denote it as the Bayesian normal linear model (for the test outcomes).

Covariate-adjusted ROC curve

The estimation of the AROC curve rests on the following three steps:

1. Estimation of the conditional distribution of test outcomes in the nondiseased group, $F_{\bar{D}}(y_{\bar{D}i} \mid \mathbf{x}_{\bar{D}i})$.
2. Computation of the placement value $U_D = 1 - F_{\bar{D}}(Y_D \mid \mathbf{X}_D)$, where, by a slight abuse of notation, we are designating it by the same letter used for the unconditional case.
3. Estimation of the cumulative distribution function of U_D.

The approach proposed by Inácio de Carvalho and Rodríguez-Álvarez (2018) for estimating the AROC curve is implemented in function `AROC.bnp`, and it combines a single-weights dependent Dirichlet

process mixture of normal distributions in Step 1 and the Bayesian bootstrap in Step 3. Again, here, in Step 1, we also recommend using cubic B-splines transformations of all continuous covariates. Using the same notation as before, we model the conditional density as

$$F_{\bar{D}}(y_{\bar{D}i} \mid \mathbf{x}_{\bar{D}i}) = \sum_{l=1}^{L_{\bar{D}}} \omega_{\bar{D}l} \Phi(y_{\bar{D}i} \mid \mathbf{z}_{\bar{D}i}^{\top} \boldsymbol{\beta}_{\bar{D}l}, \sigma_{\bar{D}l}^2).$$

The same prior distributions and default values as in the `cROC.bnp` function are adopted for $\boldsymbol{\beta}_{\bar{D}l}$ and $\sigma_{\bar{D}l}^2$. Once Step 1 has been completed, and given a posterior sample from the parameters of interest, the corresponding realization of the placement value of a diseased subject in the nondiseased population is easily computed as

$$U_{Dj}^{(s)} = 1 - F_{\bar{D}}^{(s)}(y_{Dj} \mid \mathbf{x}_{Dj}) = \sum_{l=1}^{L_{\bar{D}}} \omega_{\bar{D}l}^{(s)} \Phi\left(y_{Dj} \mid \mathbf{z}_{Dj}^{\top} \boldsymbol{\beta}_{\bar{D}l}^{(s)}, \sigma_{\bar{D}l}^{2(s)}\right), \quad j = 1, \ldots, n_D, \quad s = 1, \ldots, S.$$

Finally, in Step 3, the cumulative distribution function of $\left\{U_{Dj}^{(s)}\right\}_{j=1}^{n_D}$ is estimated through the Bayesian bootstrap

$$\mathrm{AROC}^{(s)}(p) = \sum_{j=1}^{n_D} q_j^{(s)} I\left(U_{Dj}^{(s)} \leq p\right), \quad (q_1^{(s)}, \ldots, q_{n_D}^{(s)}) \sim \mathrm{Dirichlet}(n_D; 1, \ldots, 1).$$

As before, closed-form expressions do exist for the AAUC and pAAUC

$$\mathrm{AAUC}^{(s)} = \int_0^1 \mathrm{AROC}^{(s)}(p)\mathrm{d}p = 1 - \sum_{j=1}^{n_D} q_j^{(s)} U_{Dj}^{(s)},$$

$$\mathrm{pAAUC}^{(s)}(u_1) = \int_0^{u_1} \mathrm{AROC}^{(s)}(p)\mathrm{d}p = u_1 - \sum_{j=1}^{n_D} q_j^{(s)} \min\left\{u_1, U_{Dj}^{(s)}\right\},$$

and the $\mathrm{pAAUC}_{\mathrm{TNF}}$ (Equation (22)) is computed using numerical integration methods. With regards to the YI, it is obtained by directly plugging in $\mathrm{AROC}^{(s)}(p)$ in Expression (23).

A point estimate for $\mathrm{AROC}(p)$ can be obtained by computing the mean of the ensemble $\{\mathrm{AROC}^{(1)}(p), \ldots, \mathrm{AROC}^{(S)}(p)\}$, that is,

$$\widehat{\mathrm{AROC}}(p) = \frac{1}{S} \sum_{s=1}^{S} \mathrm{AROC}^{(s)}(p),$$

and the percentiles of the ensemble can be used to provide pointwise credible bands/credible intervals. The same applies for the AAUC, pAAUC, and YI.

Package presentation and illustration

This section describes the main functions in the **ROCnReg** package and illustrates their usage using, due to confidentiality reasons, a synthetic dataset mimicking endocrine data from a cross-sectional study carried out by the Galician Endocrinology and Nutrition Foundation. A detailed description of the original dataset can be found in Tomé Martínez de Rituerto et al. (2009). The original data have also been previously analyzed in Rodríguez-Álvarez et al. (2011a,b) and Inácio de Carvalho and Rodríguez-Álvarez (2018). The synthetic data can be found in the **ROCnReg** package under the name `endosyn`, and a summary of it follows.

```
R> library("ROCnReg")
R> data("endosyn")
R> summary(endosyn)
```

```
   cvd_idf            age          gender          bmi
Min.   :0.0000   Min.   :18.25   Men  :1317   Min.   :12.60
1st Qu.:0.0000   1st Qu.:29.57   Women:1523   1st Qu.:23.19
Median :0.0000   Median :39.28                Median :26.24
Mean   :0.2433   Mean   :41.43                Mean   :26.69
3rd Qu.:0.0000   3rd Qu.:50.84                3rd Qu.:29.74
Max.   :1.0000   Max.   :84.66                Max.   :46.20
```

The dataset is comprised of 2840 individuals (1317 men and 1523 women, variable `gender`), with an age range between 18 and 85 years old. Variable `bmi` contains the body mass index (BMI) values,

and cvd_idf is the variable that indicates the presence (1) or absence (0) of two or more cardiovascular disease (CVD) risk factors. Following previous studies, the CVD risk factors considered include raised triglycerides, reduced HD-cholesterol, raised blood pressure, and raised fasting plasma glucose. Note that from the 2840 individuals, about 24% present two or more CVD risk factors.

Using the **ROCnReg** package, in the subsequent sections, we will illustrate how to ascertain, through the pooled ROC curve, the discriminatory capacity of the BMI (which acts as our diagnostic test in this example) in differentiating individuals with two or more CVD risk factors (those belonging to the diseased class D) from those having none or just one CVD risk factor (and that therefore belong to the nondiseased group $\bar{D}$). We will also show how to evaluate, through the covariate-specific ROC curve, the possible modifying effect of age and gender on the discriminatory capacity of the BMI. Finally, the last part of this section focuses on the covariate-adjusted ROC curve as a global summary measure of the BMI discriminatory ability when taking the age and gender effects into account. In the Supplementary Material, we show the usage of the package for those methods not described here in the main text.

Pooled ROC curve

The **ROCnReg** package allows estimating the pooled ROC curve by means of the four methods listed in Table 1. Here, we only present the syntax for the functions pooledROC.BB and pooledROC.dpm that correspond, respectively, to the Bayesian bootstrap estimator and the approach based on a Dirichlet process mixture (of normal distributions). The function pooledROC.emp, which implements an empirical estimator, and the function pooledROC.kernel, which is based on kernel methods, are illustrated in the Supplementary Material. The input arguments in the functions are method-specific (details can be found in the manual accompanying the package), but in all cases, numerical and graphical summaries can be obtained by calling the functions print.pooledROC, summary.pooledROC, and plot.pooledROC, which can be abbreviated by print, summary, and plot. Recall that our aim is to ascertain, using the endosyn dataset, the discriminatory capacity of the BMI in differentiating individuals with two or more CVD risk factors from those having just one or none CVD risk factors.

```
R> set.seed(123, "L'Ecuyer-CMRG") # for reproducibility
R> pROC_dpm <- pooledROC.dpm(marker = "bmi", group = "cvd_idf", tag.h = 0,
+ data = endosyn, standardise = TRUE, p = seq(0, 1, l = 101), ci.level = 0.95,
+ compute.lpml = TRUE, compute.WAIC = TRUE, compute.DIC = TRUE,
+ pauc = pauccontrol(compute = TRUE, focus = "FPF", value = 0.1),
+ density = densitycontrol(compute = TRUE),
+ prior.h = priorcontrol.dpm(L = 10), prior.d = priorcontrol.dpm(L = 10),
+ mcmc = mcmccontrol(nsave = 8000, nburn = 2000, nskip = 1),
+ parallel = "snow", ncpus = 2, cl = NULL)
```

Before describing in detail the previous call, we first present the control functions that are used. In particular,

```
pauccontrol(compute = FALSE, focus = c("FPF", "TPF"), value = 1)
```

can be used to indicate whether the pAUC should be computed (by default it is not computed), and in case it is computed (i.e., compute = TRUE), whether the focus should be placed on restricted FPFs (pAUC; see (4)) or on restricted TPFs (pAUC_{TPF}; see (5)). In both cases, the upper bound u_1 (if focus is the FPF) or the lower bound v_1 (if focus is the TPF) should be indicated in the argument value. In addition to the pooled ROC curve, AUC, and pAUC (if required), the function pooledROC.dpm also allows computing the probability density function (PDF) of the test outcomes in both the diseased and nondiseased groups. In order to do so, we use

```
densitycontrol(compute = FALSE, grid.h = NA, grid.d = NA)
```

By default, PDFs are not returned by the function pooledROC.dpm, but this can be changed by setting compute = TRUE, and through grid.h and grid.d, the user can specify a grid of test results where the PDFs are to be evaluated in, respectively, the nondiseased and diseased groups. Value NA signals auto initialization, with default a vector of length 200 in the range of the test results. Regarding the hyperparameters for the Dirichlet process mixture of normals model (used for the estimation of the PDFs/CDFs of the test outcomes in each group), they can be controlled using

```
priorcontrol.dpm(m0 = NA, S0 = NA, a = 2, b = NA, alpha = 1, L = 10)
```

A detailed description of these hyperparameters is found in Section Methods. Finally, to set the various parameters controlling the MCMC procedure (which in our case is simply a Gibbs sampler), we use

```
mcmccontrol(nsave = 8000, nburn = 2000, nskip = 1)
```

Here, nsave is an integer value with the total number of scans to be saved, nburn is the number of burn-in scans, and nskip is the thinning interval. Unless due to memory usage reasons, we recommend not thinning and instead monitoring the effective sample size of the MCMC chain.

Coming back to the pooledROC.dpm function, through marker, the user specifies the name of the variable containing the test results. In our case, these are the values of the BMI. The name of the variable that distinguishes diseased (two or more CVD risk factors, D) from nondiseased individuals (none or one CVD risk factor, $\bar{D}$) is represented by the argument group, and the value codifying nondiseased individuals in group is specified by tag.h. The data argument is a data frame containing the data and all needed variables. Setting standardise = TRUE (the default) will standardize (i.e., subtract the mean and divide by the standard deviation) the test outcomes. The set of FPFs at which to estimate the pooled ROC curve is specified in the argument p, and argument ci.level allows specifying the level for the credible intervals (by default: 0.95). The LPML, WAIC, and DIC are computed by setting, respectively, the arguments compute.lpml, compute.WAIC, and compute.DIC to TRUE. Argument pauc is an (optional) list of values to replace the default values returned by the function pauccontrol. Here, we ask for the pAUC to be computed, with the focus on restricted FPFs and upper bound $u_1 = 0.1$. Similarly, the argument density is an (optional) list of values to replace the default values returned by the function densitycontrol, as it is the argument mcmc. Through prior.h and prior.d arguments, we specify the hyperparameters in the nondiseased and diseased groups, respectively. Again, both arguments are (optional) lists of values to replace the default values returned by the function priorcontrol.dpm. We shall remember that different hyperparameters' default values are set depending on whether test outcomes are standardized or not. Finally, arguments parallel, ncpus and cl allow performing parallel computations (based on the R-package **parallel**). In particular, through parallel, the user specifies the type of parallel operation: either "no" (default), "multicore" (not available on Microsoft Windows operating systems), or "snow". Argument ncpus is used to indicate the number of processes to be used in a parallel operation (when parallel = "multicore", or parallel = "snow"), and cl is an optional parallel or snow cluster to be used when parallel = "snow". If cl is not supplied (as in our example), a cluster on the local machine is created for the duration of the call.

A numerical summary of the fitted model can be obtained by calling the function summary, which provides, among other information, the estimated AUC (posterior mean) and 95% credible interval (recall that we set in the call to the function ci.level = 0.95) and, if required, the LPML, WAIC, and DIC, separately, in the nondiseased (denoted here as Group H) and diseased (Group D) groups.

```
R> summary(pROC_dpm)

Call:
pooledROC.dpm(marker = "bmi", group = "cvd_idf", tag.h = 0, data = endosyn,
    standardise = TRUE, p = seq(0, 1, l = 101), ci.level = 0.95,
    compute.lpml = TRUE, compute.WAIC = TRUE, compute.DIC = TRUE,
    pauc = pauccontrol(compute = TRUE, focus = "FPF", value = 0.1),
    density = densitycontrol(compute = TRUE), prior.h = priorcontrol.dpm(L = 10),
    prior.d = priorcontrol.dpm(L = 10), mcmc = mcmccontrol(nsave = 8000,
        nburn = 2000, nskip = 1), parallel = "snow", ncpus = 2, cl = NULL)

Approach: Pooled ROC curve - Bayesian DPM
---------------------------------------------
Area under the pooled ROC curve: 0.759 (0.74, 0.777)*
Partial area under the pooled ROC curve (FPF = 0.1): 0.168 (0.139, 0.199)*
 * Credible level:  0.95

Model selection criteria:
                     Group H       Group D
WAIC               12490.485      4017.063
WAIC (Penalty)         8.431         5.468
LPML               -6245.247     -2008.541
DIC                12490.276      4016.920
DIC (Penalty)          8.326         5.396


Sample sizes:
                           Group H     Group D
Number of observations        2149         691
Number of missing data           0           0
```

To complement these numerical results, the **ROCnReg** package also provides graphical results that

can be used to further explore the fitted model. Specifically, the function `plot` depicts the estimated pooled ROC curve and AUC (posterior means), jointly with `ci.level`×100% (pointwise) credible intervals (here 95%)

```
R> plot(pROC_dpm, cex.main = 1.5, cex.lab = 1.5, cex.axis = 1.5, cex = 1.5)
```

The result of the above code is shown in Figure 2.

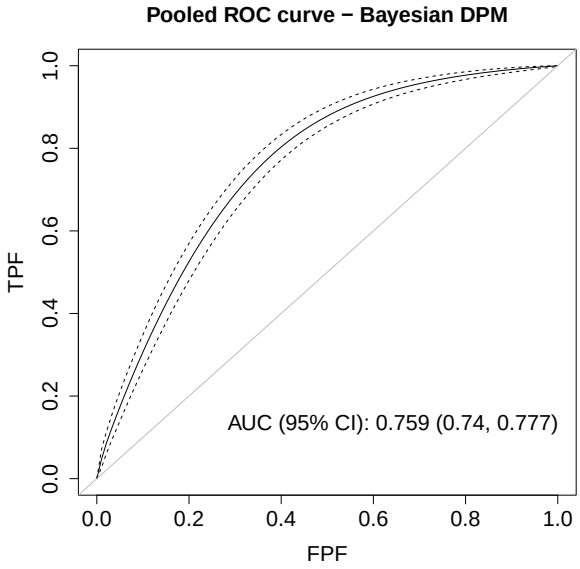
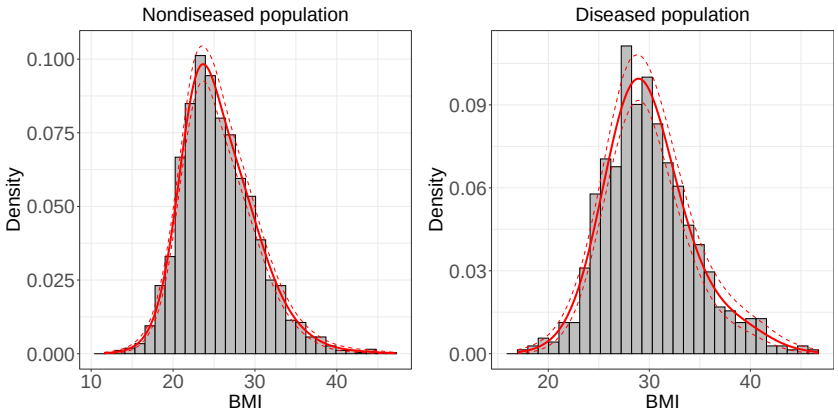


Figure 2: Graphical results as provided by the `plot.pooledROC` function for an object of class `pooledROC.dpm`. Posterior mean and 95% pointwise credible band for the pooled ROC curve and corresponding posterior mean and 95% credible interval for the AUC.

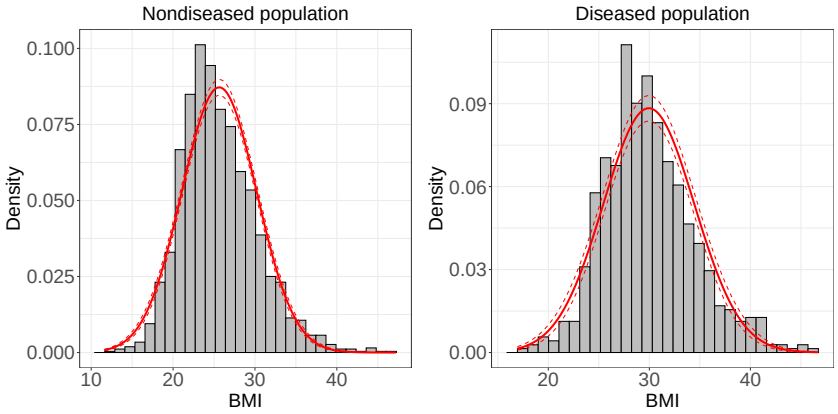
By means of `density = densitycontrol(compute = TRUE)` in the call to the function, the estimates of the PDFs of the BMI in both groups are to be returned. This information can be accessed through component `dens` in the object `pROC_dpm` (i.e., `pROC_dpm$dens`), which is a list with elements `h` and `d` associated with the nondiseased and diseased groups, respectively. Each of the two elements is itself another list of two components: (1) `grid`, a vector that contains the grid of test results at which the PDFs have been evaluated (estimated); and (2) `dens`, a matrix with the PDFs at each iteration of the MCMC procedure. We can use these results to plot, e.g., the posterior mean (and 95% pointwise credible bands) of the PDF of the BMI in the healthy and diseased populations (see Figure 3a obtained using the R package **ggplot2** by Wickham, 2016). As can be observed, the estimated densities obtained under the DPM method follow very closely the histograms of the data. Further, the estimated densities available in `dens` can be used, as advised by Gelman et al. (2013, p. 553), to monitor convergence of the MCMC chains. The well-known label switching problem often leads to poor mixing of the chains of the component-specific parameters, but this may not impact convergence and mixing of the induced density/distribution of interest. For instance, Figure 4 shows the trace plots of the MCMC iterations (after burn-in) of the PDFs of the BMI in the two groups for different (and randomly selected) values of the BMI, and Figure 5 depicts the corresponding effective sample sizes and Geweke statistics (obtained using the R package **coda** by Plummer et al., 2006). Note that all plots give evidence of a good mixing and do not suggest a lack of convergence. For conciseness, the R code for reproducing Figures 3a, 4, and 5 is not provided here but in the replication code that accompanies the paper.

It is worth noting that the function `pooledROC.dpm` also allows fitting a normal distribution in each group. This is just a particular case (for which $L_D = L_{\bar{D}} = 1$) of the more general DPM model. In order to fit such model, one simply needs to set $L = 1$ in the `prior.d` and `prior.h` arguments. The code follows.

```
R> set.seed(123, "L'Ecuyer-CMRG") # for reproducibility
R> pROC_normal <- pooledROC.dpm(marker = "bmi", group = "cvd_idf", tag.h = 0,
+ data = endosyn, standardise = TRUE, p = seq(0, 1, l = 101), ci.level = 0.95,
+ compute.lpml = TRUE, compute.WAIC = TRUE, compute.DIC = TRUE,
+ pauc = pauccontrol(compute = TRUE, focus = "FPF", value = 0.1),
+ density = densitycontrol(compute = TRUE),
```



(a) DPM model with 10 mixture components in each group



(b) Normal model in each group

Figure 3: Histogram of the (observed) BMI and posterior mean jointly along with 95% pointwise credible bands (red lines) of the PDF of the BMI obtained using (a) a Dirichlet process mixture of normals model (object `pROC_dpm`); and (b) a normal model (object `pROC_normal`). Left: Nondiseased individuals (none or one CVD risk factor). Right: Diseased individuals (two or more CVD risk factors).

```
+ prior.h = priorcontrol.dpm(L = 1), prior.d = priorcontrol.dpm(L = 1),
+ mcmc = mcmccontrol(nsave = 8000, nburn = 2000, nskip = 1),
+ parallel = "snow", ncpus = 2)
```

For the sake of space, we omit from the summary the call to the function

```
R> summary(pROC_normal)

Call: [...]

Approach: Pooled ROC curve - Bayesian DPM
-----------------------------------------------
Area under the pooled ROC curve: 0.748 (0.728, 0.768)*
Partial area under the pooled ROC curve (FPF = 0.1): 0.224 (0.194, 0.253)*
 * Credible level:  0.95

Model selection criteria:
                    Group H       Group D
WAIC              12639.952      4049.004
WAIC (Penalty)        2.431         2.267
LPML              -6319.976     -2024.502
DIC               12639.505      4048.714
DIC (Penalty)         1.986         1.987


Sample sizes:
```

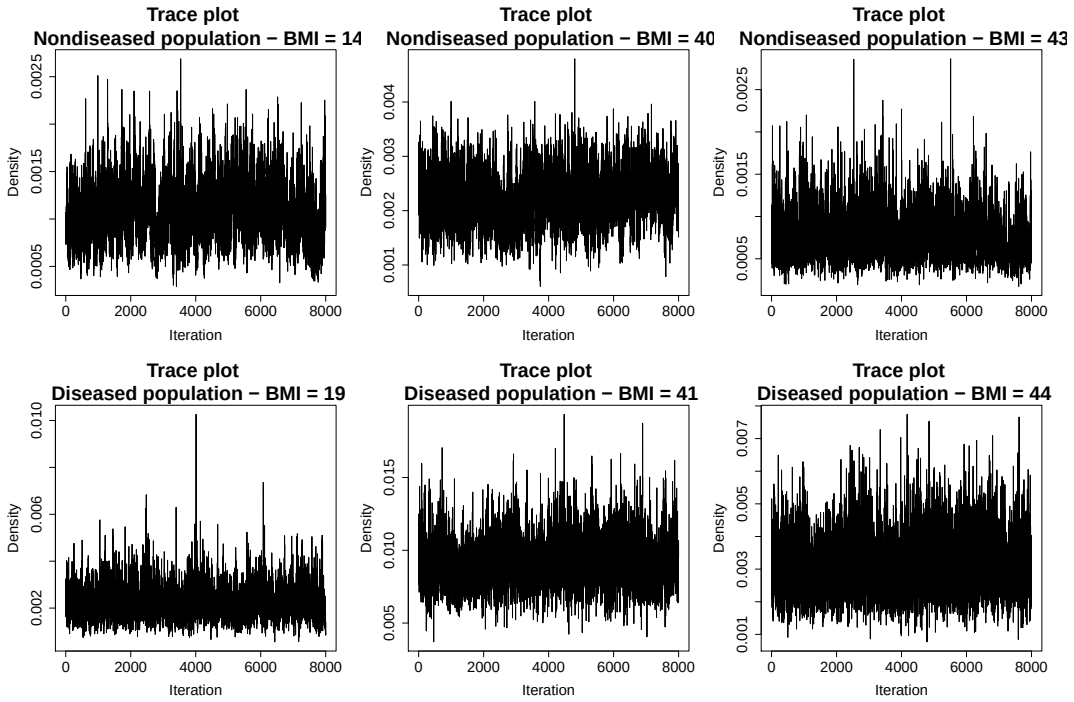


Figure 4: Trace plots of the MCMC draws (after burn-in) of the PDFs of the BMI based on the model `pROC_dpm`. Results are shown separately for the nondiseased and diseased populations and for different values of the BMI.

```
                          Group H     Group D
Number of observations       2149         691
Number of missing data          0           0
```

The fit of the DPM and normal models in each group can be compared on the basis of the WAIC, DIC, and/or the LPML. Remember that for the LPML, the higher its value, the better the model fit, while for the WAIC and DIC, it is the other way around. By comparing these values, provided in the summary of each fitted model, we can conclude that the three criteria favor, in both the diseased and (especially in the) nondiseased groups, the more general DPM model. This is also corroborated by the plot of the fitted densities in each group shown in Figure 3b.

We now estimate the pooled ROC curve using the Bayesian bootstrap estimator (function `pooledROC.BB`), and comparisons with the results obtained using the DPM approach are provided.

```
R> set.seed(123, "L'Ecuyer-CMRG") # for reproducibility
R> pROC_BB <- pooledROC.BB(marker = "bmi", group = "cvd_idf", tag.h = 0, data = endosyn,
+ p = seq(0, 1, l = 101), pauc = pauccontrol(compute = TRUE, focus = "TPF", value = 0.8),
+ B = 5000, ci.level = 0.95, parallel = "snow", ncpus = 2)

R> summary(pROC_BB)

Call: [...]

Approach: Pooled ROC curve - Bayesian bootstrap
-----------------------------------------------
Area under the pooled ROC curve: 0.76 (0.74, 0.779)*
Partial area under the pooled ROC curve (FPF = 0.1): 0.17 (0.14, 0.201)*
 * Credible level:  0.95

Sample sizes:
                          Group H     Group D
Number of observations       2149         691
Number of missing data          0           0
```

Note that the posterior means for the AUC and pAUC obtained using the DPM method (0.759 and

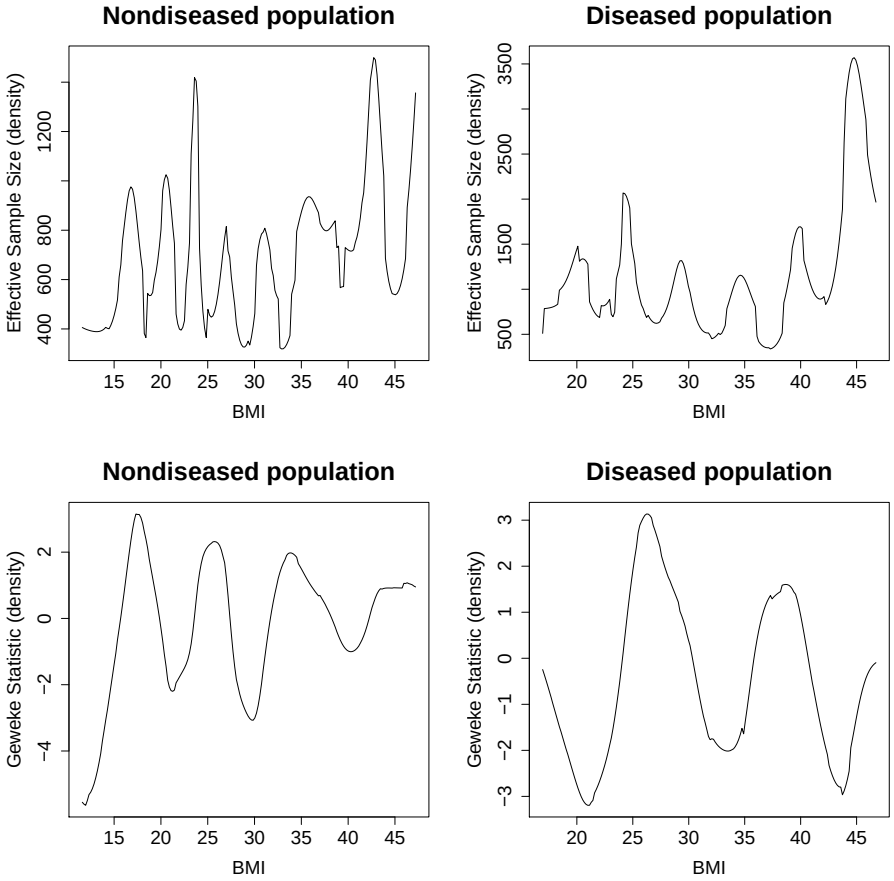


Figure 5: Effective sample size and Geweke statistic of the MCMC chains (after burn-in) and of the PDFs of the BMI based on model `pROC_dpm` in the nondiseased and diseased populations. In both cases, results are shown along BMI values.

0.168, respectively) and the Bayesian bootstrap approach (0.760 and 0.170) are almost identical. This can also be observed when plotting the estimated ROC curves under the two methods (Figure 6).

We finish this section by showing how to use **ROCnReg** to obtain an (optimal) threshold value which could be further used to 'diagnose' an individual as diseased (two or more CVD risk factors) or healthy/nondiseased (none or only one CVD risk factor). To that aim, and for `pooledROC` objects (i.e., those obtained using functions `pooledROC.dpm`, `pooledROC.BB`, `pooledROC.emp`, and `pooledROC.kernel`), we use the function `compute.threshold.pooledROC`, which allows obtaining (optimal) threshold values using two criteria: the YI and the one that sets a target value for the FPF. For illustration, we show here the results using the YI criterion.

```
R> th_pROC_dmp <- compute.threshold.pooledROC(pROC_dpm, criterion = "YI",
+ ci.level = 0.95, parallel = "snow", ncpus = 2)
R> th_pROC_dmp

$call
compute.threshold.pooledROC(object = pROC_dpm, criterion = "YI",
    ci.level = 0.95, parallel = "snow", ncpus = 2)

$thresholds
     est       ql       qh
26.46877 26.07129 26.85029

$YI
      est        ql        qh
0.4045776 0.3721684 0.4366298

$FPF
```

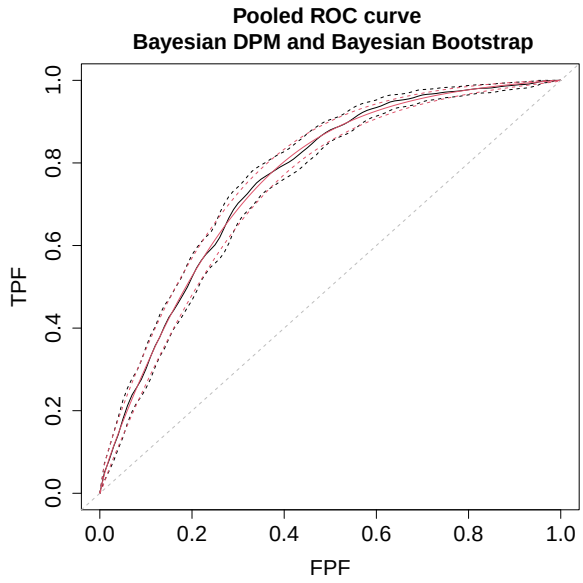


Figure 6: Estimated ROC curve using the Bayesian bootstrap approach (in black) and the DPM method (in red). Solid lines represent the posterior means and dashed lines the 95% pointwise credible bands.

```
      est        ql        qh
0.3808336 0.3469580 0.4159478

$TPF
      est        ql        qh
0.7854112 0.7528575 0.8161865
```

The function returns the posterior mean (`est`) and `ci.level`×100% (here 95% since `ci.level = 0.95`) credible interval (lower bound: `ql`, upper bound: `qh`) for the YI and associated threshold value, as well as for the FPF and TPF associated with this cutoff value. For our example, the (posterior mean of the) YI is 0.40, and the YI-based threshold value is a BMI value of 26.5, which falls in the nutritional status defined as pre-obesity by the World Health Organization. By using this YI-based threshold value, we would have an FPF of 0.38 and a TPF of 0.79.

Covariate-specific ROC curve

We now turn our attention to the inclusion of covariates in ROC analysis. As shown in Table 1, with **ROCnReg**, the user can estimate the covariate-specific ROC curve by means of three approaches. As for the functions in **ROCnReg** for estimating the pooled ROC curve, the input arguments are method-specific, and we refer the reader to the manual for details. For all methods, numerical and graphical summaries are obtained using functions `print.cROC`, `summary.cROC`, and `plot.cROC`. Here, we describe how to use the function `cROC.bnp` that implements the Bayesian nonparametric approach for estimating the covariate-specific ROC curve detailed in Section Methods. Also, for objects of this class, **ROCnReg** provides the function `predictive.checks`, which implements tools for assessing model fit via posterior predictive checks.

Recall that, when including covariate information in ROC analysis, interest resides in evaluating if and how the discriminatory capacity of the test varies with such covariates. In particular, in our endocrine study, we aim at evaluating the possible effect of both `age` and `gender` in the discriminatory capacity of the BMI. In what follows, with this aim in mind, two different models are fitted using the function `cROC.bnp`. One which considers a normal distribution in each group and that incorporates the `age` effect in a linear way and a second one which caps the maximum number of mixture components in each group at 10 (i.e., $L_D = L_{\bar{D}} = 10$) and that models the `age` effect using cubic B-splines (and thus allows for a nonlinear effect of age). Following Rodríguez-Álvarez et al. (2011a,b), both models consider the interaction between `age` and `gender`. For clarity, we first focus on the code that models the `age` effect in a linear way and use it to describe in detail the different arguments of the `cROC.bnp` function.

```
R> # Dataframe for predictions
```

```
R> agep <- seq(22, 80, l = 30)
R> endopred <- data.frame(age = rep(agep,2), gender = factor(rep(c("Women", "Men"),
+ each = length(agep))))

R> set.seed(123, "L'Ecuyer-CMRG") # for reproducibility
R> cROC_bp <- cROC.bnp(formula.h = bmi ~ gender*age, formula.d = bmi ~ gender*age,
+ group = "cvd_idf", tag.h = 0, data = endosyn, newdata = endopred,
+ standardise = TRUE, p = seq(0, 1, l = 101), ci.level = 0.95, compute.lpml = TRUE,
+ compute.WAIC = TRUE, compute.DIC = TRUE, pauc = pauccontrol(compute = FALSE),
+ prior.h = priorcontrol.bnp(L = 1), prior.d = priorcontrol.bnp(L = 1),
+ density = densitycontrol(compute = TRUE),
+ mcmc = mcmccontrol(nsave = 8000, nburn = 2000, nskip = 1),
+ parallel = "snow", ncpus = 2)
```

As can be seen, many arguments coincide with those of the function `pooledROC.dpm` (described in the previous section). We thus focus here on those that are specific to `cROC.bnp`. The arguments `formula.h` and `formula.d` are `formula` objects specifying the model for the regression function (see Equation (28)) in, respectively, the nondiseased and diseased groups. They are similar to the formula used with the `glm` function, except that nonlinear functions (modeled by means of cubic B-splines) can be added using function `f` (an example will follow later in this section). Note that in both cases, the left-hand side of the formulas should include the name of the test/marker (in our case `bmi`). In our application, and for both groups, the model for the component's means includes, in addition to the linear effect of age and gender, the (linear) interaction between these two covariates (i.e., `gender*age` $\equiv$ `gender + age + gender:age`). Through the `newdata` argument, the user can specify a new data frame containing the values of the covariates at which the covariate-specific ROC curve and AUC (and also pAUC and PDFs, if required) are to be computed. Finally, `prior.h` (the same holds for `prior.d`) is a (optional) list of values to replace the defaults returned by `priorcontrol.bnp`, which allows setting the hyperparameters for the single-weights dependent Dirichlet process mixture of normals model (see Section Methods and the manual accompanying the package for more details)

```
priorcontrol.bnp(m0 = NA, S0 = NA, nu = NA, Psi = NA, a = 2, b = NA,
   alpha = 1, L = 10)
```

In our example, we only modified the upper bound for the number of components in the mixture model, which by default is 10, and set it to 1.

In this case, the `summary` of the fitted model provides the following information.

```
R> summary(cROC_bp)

Call: [...]

Approach: Conditional ROC curve - Bayesian nonparametric
-----------------------------------------------------------

Parametric coefficients
Group H:
                   Post. mean    Post. quantile 2.5%    Post. quantile 97.5%
(Intercept)           26.1459                25.8765                 26.4096
genderWomen           -0.9160                -1.2726                 -0.5680
age                    1.1949                 0.9180                  1.4690
genderWomen:age        1.1948                 0.8455                  1.5394


Group D:
                   Post. mean    Post. quantile 2.5%    Post. quantile 97.5%
(Intercept)           29.1865                28.7625                 29.6115
genderWomen            2.0826                 1.3705                  2.7665
age                    0.6578                 0.2162                  1.0904
genderWomen:age       -0.7711                -1.4655                 -0.0956


ROC curve:
                   Post. mean    Post. quantile 2.5%    Post. quantile 97.5%
(Intercept)           -0.6959                -0.8177                 -0.5776
genderWomen           -0.6863                -0.8695                 -0.5046
age                    0.1229                 0.0045                  0.2415
```

```
genderWomen:age        0.4499              0.2745              0.6245
b                      0.9391              0.8824              0.9975


Model selection criteria:
                     Group H       Group D
WAIC               12174.986      4007.980
WAIC (Penalty)         6.283         5.646
LPML               -6087.492     -2003.990
DIC                12173.664      4007.329
DIC (Penalty)          4.994         5.053


Sample sizes:
                          Group H     Group D
Number of observations       2149         691
Number of missing data          0           0
```

The first aspect to note is that, in this case, the `summary` function does not provide the estimated AUC as there is one (possibly different) AUC for each combination of covariate values. Also, given that: (1) only one component has been considered for modeling the CDFs of test results in the diseased and nondiseased groups, and (2) covariate effects have been modeled in a linear way, the `summary` function provides the posterior mean (and quantiles) of the (parametric) coefficients associated with the regression functions (Equation (30)) and with the covariate-specific ROC curve (Equation (32)). We note that since in the call to the function we have specified `standardise = TRUE` (and consequently both the test outcomes and covariates are standardized), the regression coefficients are on the scale of the standardized covariates. If we focus on the coefficients for the covariate-specific ROC curve, it seems that the discriminatory capacity of the BMI decreases with age, with the decrease being more pronounced in women (note that the expression of the covariate-specific ROC curve in Equation (31) implies that positive coefficients correspond to a decrease in discriminatory capacity). These results are possibly better judged by plotting the estimated covariate-specific ROC curves and associated AUCs. This can be done using the `plot` function. For the covariate-specific ROC curve, the depicted graphics will depend on the number and nature of the covariates included in the analyses. In particular, for our application, we obtain, separately for men and women, the covariate-specific ROC curves (and AUCs) along age. These are shown in Figure 7, obtained using the code

```
R> op <- par(mfrow = c(2,2))
R> plot(cROC_sp, ask = FALSE)
R> par(op)
```

Although in this example we have modeled the `age` effect linearly and only one mixture component was considered, **ROCnReg** also allows for modeling the effect of continuous covariates in a nonlinear way, either using cubic B-spline basis expansions (through the function `cROC.bnp`) or kernel-based smoothers (via the function `cROC.kernel` which is described in the Supplementary Material). Also, as noted before, using only one mixture component for the single-weights dependent Dirichlet process mixture of normals model (function `cROC.bnp`) is equivalent to considering a (Bayesian) normal model, which might be too restrictive for most data applications. In what follows, we provide more flexibility to the model for the covariate-specific ROC curve by means of (1) increasing the number of mixture components and (2) modeling the `age` effect in a nonlinear way (recall our considerations in Section Covariate-specific ROC curve about the lack of flexibility of the single-weights dependent Dirichlet process mixture of normals model when covariates effects on the components' means are modeled linearly). The former is done by modifying the value of `L` in the arguments `prior.h` and `prior.d`, with 10 being the default value. Regarding the latter, this is done by making use of the function `f` when specifying the component's mean functions through `formula.h` and `formula.d`. In particular, in our application we are interested in modeling the factor-by-curve interaction between `age` and `gender` (i.e., we model the `age` effect 'separately' for men and women). This is done using, e.g., `bmi ~ gender + f(age,by = gender,K = c(3,5))`. Through argument `K`, we indicate the number of internal knots used for constructing the cubic B-spline basis used to approximate the nonlinear effect of `age` (with the quantiles of `age` used to anchor the knots). Note that we can specify a different number of internal knots for men and women (`K = c(3,5)`), where the order of vector `K` should match the ordering of the levels of the factor `gender`. We also note that to assist in the selection of the number of interior knots (in **ROCnReg**, the location is always based on the quantiles of the corresponding covariates), the user can make use of the WAIC, DIC, and/or LPML. For instance, for this application, we fitted different models with a different number of internal knots, and we have chosen the model that provided the lowest WAIC (this was done in both the nondiseased and diseased populations, and we remark that

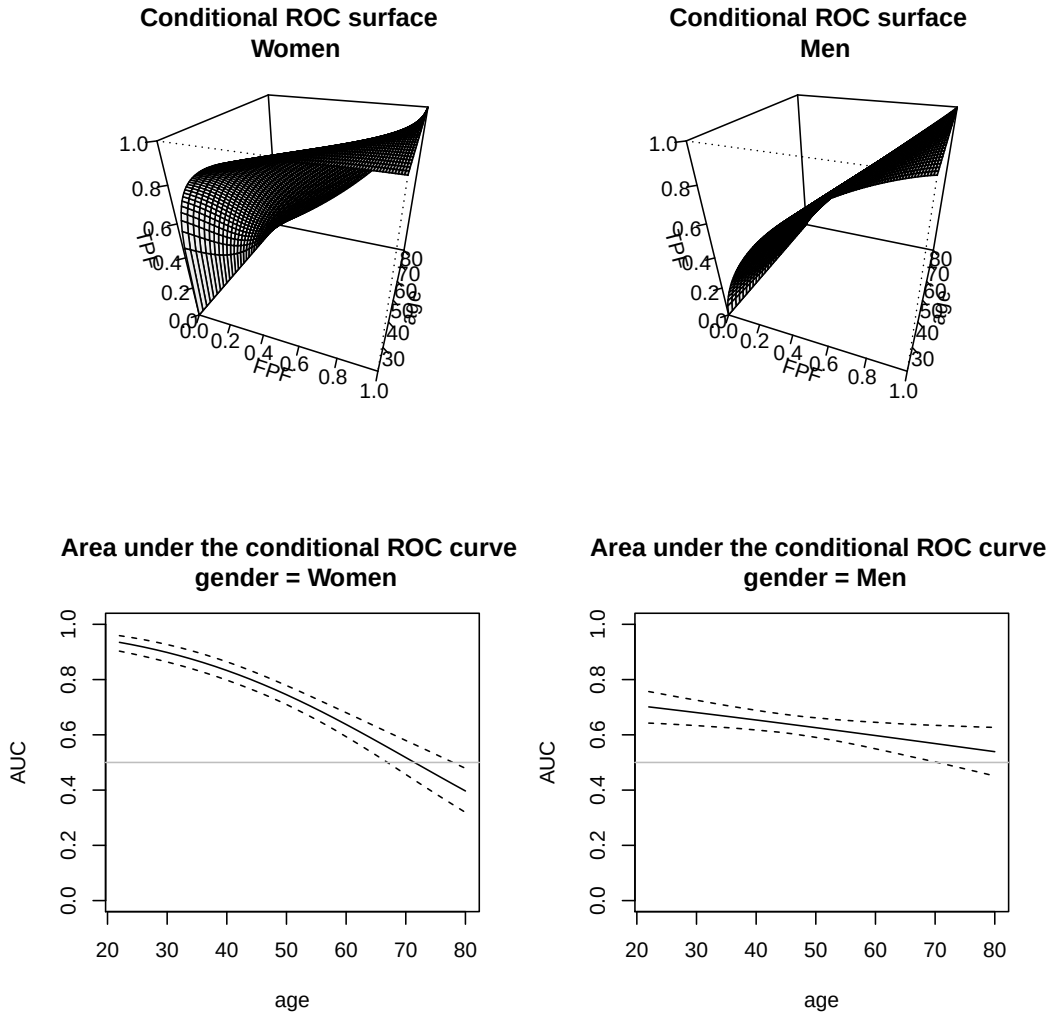


Figure 7: Graphical results as provided by the `plot.cROC` function for an object of class `cROC.bnp`. Results for the model that includes the linear interaction between age and gender and one mixture component. Top row: Posterior mean of the covariate-specific ROC curve along age, separately for men and women. Bottom row: Posterior mean and 95% pointwise credible band for the covariate-specific AUC along age, separately for men and women.

the number of knots does not need to be the same in the two populations). The final model is shown below.

```
R> # Levels of gender, and its ordering.
R> # Needed if we want to specify different
R> # number of knots for men and women
R> levels(endosyn$gender)

[1] "Men"   "Women"

R> set.seed(123, "L'Ecuyer-CMRG") # for reproducibility
R> cROC_bnp <- cROC.bnp(
+ formula.h = bmi ~ gender + f(age, by = gender, K = c(0,0))
+ formula.d = bmi ~ gender + f(age, by = gender, K = c(4,4)),
+ group = "cvd_idf", tag.h = 0, data = endosyn, newdata = endopred,
+ standardise = TRUE, p = seq(0, 1, l = 101), ci.level = 0.95, compute.lpml = TRUE,
+ compute.WAIC = TRUE, compute.DIC = TRUE, pauc = pauccontrol(compute = FALSE),
+ prior.h = priorcontrol.bnp(L = 10), prior.d = priorcontrol.bnp(L = 10),
+ density = densitycontrol(compute = TRUE),
+ mcmc = mcmccontrol(nsave = 8000, nburn = 2000, nskip = 1),
```

```
+ parallel = "snow", ncpus = 2)

R> summary(cROC_bnp)

Call: [...]

Approach: Conditional ROC curve - Bayesian nonparametric
-----------------------------------------------------------

Model selection criteria:
                     Group H       Group D
WAIC               11833.000      3909.828
WAIC (Penalty)        31.236        38.583
LPML               -5916.766     -1955.449
DIC                11829.750      3904.532
DIC (Penalty)         29.611        35.934


Sample sizes:
                           Group H     Group D
Number of observations        2149         691
Number of missing data           0           0
```

```
R> op <- par(mfrow = c(2,2))
R> plot(cROC_sp, ask = FALSE)
R> par(op)
```

The graphical results are shown in Figure 8. Note that, especially for women, age displays a marked nonlinear effect. Recall that for objects of class `cROC.bnp`, and if required in the call to the function, the `summary` function provides, separately for the diseased and nondiseased/healthy groups, the WAIC, LPML, and DIC. Note that, in both cases, the three criteria support the use of the more flexible model that uses cubic B-splines and 10 mixture components for modeling the distribution of the BMI (model `cROC_bnp`) over the more restrictive Bayesian normal linear model (model `cROC_bp`). Because the WAIC, LPML, and DIC are relative criteria, posterior predictive checks are also available in **ROCnReg** through the function `predictive.checks`. Specifically, the function generates replicated datasets from the posterior predictive distribution in the two groups D and $\bar{D}$ and compares them to the test values (BMI values in our application) using specific statistics. For the choice of such statistics, we follow Gabry et al. (2019), who suggest choosing statistics that are 'orthogonal' to the model parameters. Since we are using a location-scale mixture of normals model for the test outcomes, we use the skewness and kurtosis here and check how well the posterior predictive distribution captures these two quantities.

```
R> op <- par(mfrow = c(2,3))
R> pc_cROC_bp <- predictive.checks(cROC_bp,
+ statistics = c("kurtosis", "skewness"), devnew = FALSE)
R> par(op)

R> op <- par(mfrow = c(2,3))
R> pc_cROC_bnp <- predictive.checks(cROC_bnp,
+ statistics = c("kurtosis", "skewness"), devnew = FALSE)
R> par(op)
```

Results are shown in Figure 9. As can be seen, the model that includes the factor-by-curve interaction between age and gender and 10 mixture components performs quite well in capturing both quantities, while the Bayesian normal linear model fails to do so. Also shown in Figure 9 (and provided by function `predictive.checks`) are the kernel density estimates of 500 randomly selected datasets drawn from the posterior predictive distribution, in each group, compared to the kernel density estimate of the observed BMI (in each group). Again, the more flexible model, as opposed to the Bayesian normal linear model, is able to simulate data that are very much similar to the observed BMI values.

As for the pooled ROC curve, **ROCnReg** also provides a function that allows obtaining (optimal) threshold values for the covariate-specific ROC curve. For illustration, instead of the threshold values based on the Youden index, we now use the criterion that sets a target value for the FPF. The code for model `cROC_bnp`, when setting the FPF $= 0.3$, is as follows.

```
R> th_fpf_cROC_bnp <- compute.threshold.cROC(cROC_bnp, criterion = "FPF", FPF = 0.3,
+ newdata = endopred, ci.level = 0.95, parallel = "snow", ncpus = 2)
```

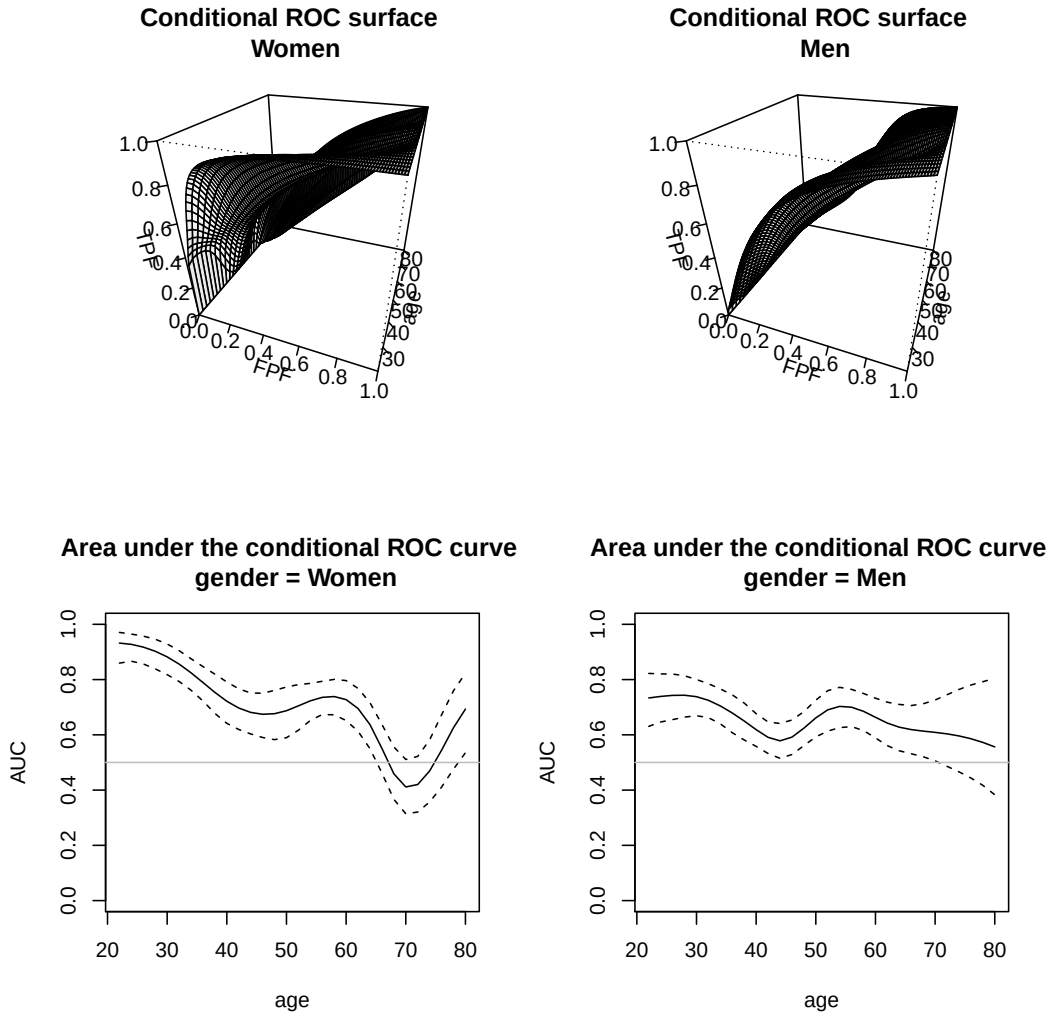
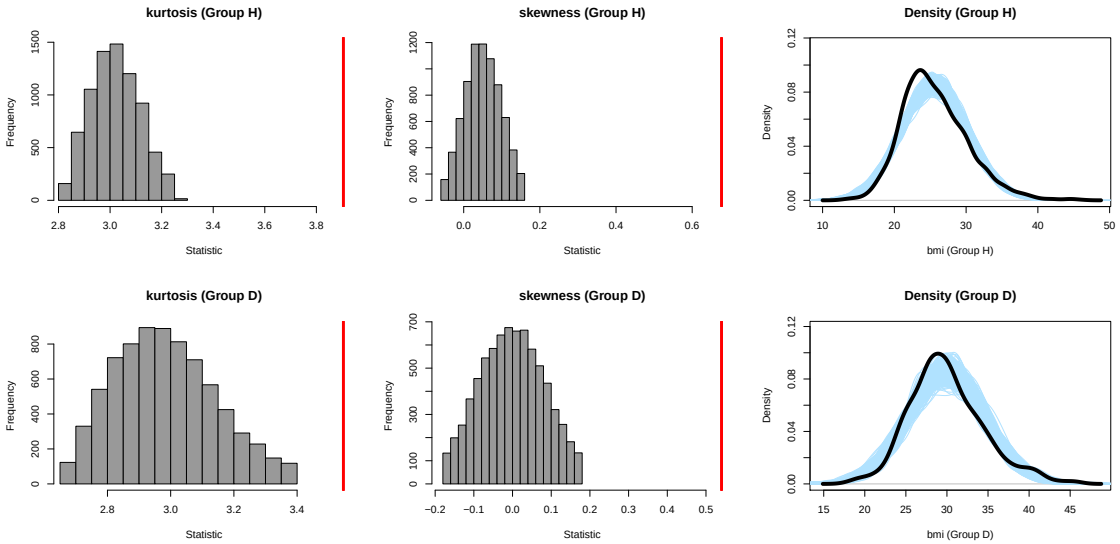


Figure 8: Graphical results as provided by the `plot.cROC` function for an object of class `cROC.bnp`. Results for the model that includes the factor-by-curve interaction between `age` and `gender` and 10 mixture components. Top row: Posterior mean of the covariate-specific ROC curve along age, separately for men and women. Bottom row: Posterior mean and 95% pointwise credible band for the covariate-specific AUC along age, separately for men and women.

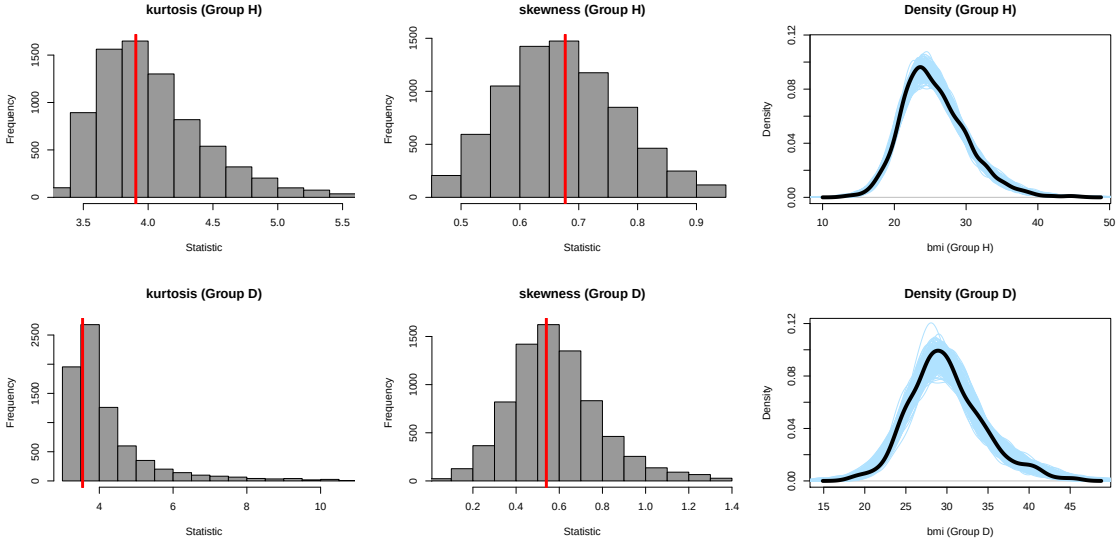
```
R> names(th_fpf_cROC_bnp)

[1] "newdata"    "thresholds" "TPF"    "FPF"    "call"
```

In addition to the data frame `newdata` containing the covariate values at which the thresholds are computed, the function `compute.threshold.cROC` also returns the covariate-specific `thresholds` corresponding to the specified FPF as well as the covariate-specific `TPF` attached to these thresholds. In both cases, the function returns the posterior mean and the `ci.level`$\times$100% (here 95%) pointwise credible intervals. Although **ROCnReg** does not provide a function for plotting the results obtained using `compute.threshold.cROC`, graphical results can be easily obtained. For simplicity, we only show here the code for the covariate-specific threshold values (`thresholds`), but a similar code can be used to plot the covariate-specific TPFs. Both plots are shown in Figure 10. As can be observed, for an FPF of 0.3, the BMI age-specific thresholds tend to increase with age both for men and women, although for the latter, there is a slight decrease after the age of about 70 years old. The age-specific TPFs corresponding to the thresholds for which the FPF is 0.3 show a nonlinear behavior, and these are in general higher for women than for men (of the same age).

```
R> df <- data.frame(age = th_fpf_cROC_bnp$newdata$age,
+ gender = th_fpf_cROC_bnp$newdata$gender, y = th_fpf_cROC_bnp$thresholds[[1]][,"est"],
+ ql = th_fpf_cROC_bnp$thresholds[[1]][,"ql"],
```



(a) Model including the linear interaction between age and gender and one component

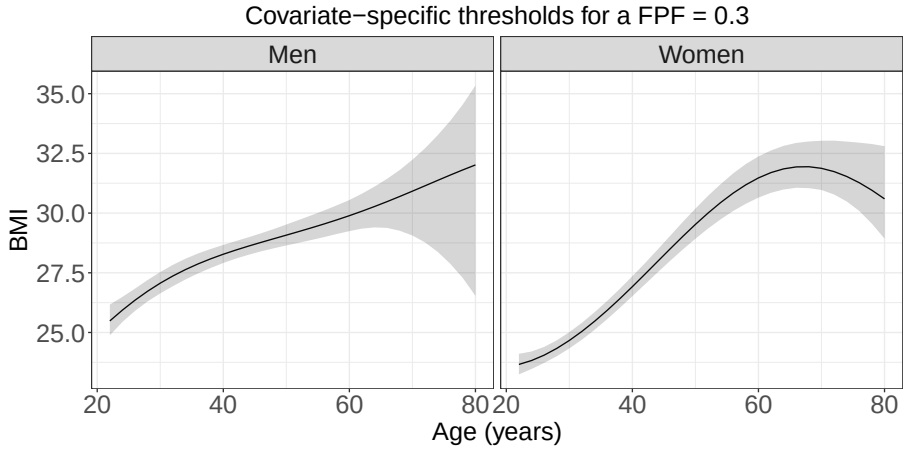


(b) Model including the factor-by-curve interaction between age and gender and 10 mixture components

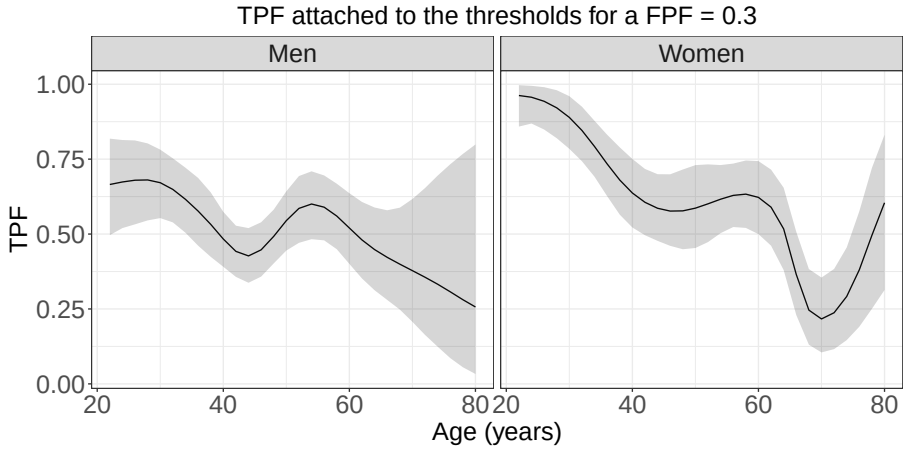
Figure 9: Graphical results as provided by the `predictive.checks` function for an object of class `cROC.bnp`. Histograms of the statistics *skewness* and *kurtosis* computed from 8000 draws from the posterior predictive distribution in the diseased and nondiseased groups. The red line is the estimated statistic from the observed BMI values. The right-hand side plots show the kernel density estimate of the observed BMI (solid black line), jointly with the kernel density estimates for 500 simulated datasets drawn from the posterior predictive distributions.

```
+ qh = th_fpf_cROC_bnp$thresholds[[1]][,"qh"])

R> g0 <- ggplot(df, aes(x = age, y = y, ymin = ql, ymax = qh)) + geom_line() +
+ geom_ribbon(alpha = 0.2) +
+ labs(title = "Covariate-specific thresholds for an FPF = 0.3",
+ x = "Age (years)", y = "BMI") +
+ theme(strip.text.x = element_text(size = 20),
+ plot.title = element_text(hjust = 0.5, size = 20),
+ axis.text = element_text(size = 20),
+ axis.title = element_text(size = 20)) + facet_wrap(~gender)
R> print(g0)
```

For conciseness, we have not shown here how to perform convergence diagnostics of the MCMC



(a) BMI threshold values



(b) True positive fractions

Figure 10: Top row: Posterior mean (solid black line) and 95% pointwise credible band for the BMI threshold values, along age, corresponding to an FPF of 0.3. Bottom row: Posterior mean (solid black line) and 95% pointwise credible band of the TPFs, along age, corresponding to the BMI threshold values for which FPF $= 0.3$.

chains for models fitted using the function `cROC.bnp`. In very much the same way as shown in the previous section for the object `pROC_dpm`, using the information contained in component `dens` in the list of returned values (if required), one can produce trace plots of the conditional densities at some sampled values, as well as obtain the corresponding effective sample sizes and Geweke statistics. Some results are provided in the Supplementary Material, and the associated code can be found in the replication code that accompanies this paper.

Covariate-adjusted ROC curve

In this section, we illustrate how to conduct inference about the covariate-adjusted ROC curve using **ROCnReg**. Similar to the covariate-specific ROC curve, three approaches are available for estimating the AROC curve. The function `AROC.bnp` is illustrated below, while `AROC.sp` and `AROC.kernel` are exemplified in the Supplementary Material.

Recall that the AROC curve is a global summary measure of diagnostic accuracy that takes covariate information into account. In the context of our endocrine application, we seek to study the overall discriminatory capacity of the BMI for detecting the presence of CVD risk factors when adjusting for age and gender. Here, we focus on how to estimate the AROC curve using the `AROC.bnp` function. The function syntax is exactly similar to the one of `cROC.bnp`, with the only difference being that we only need to specify the arguments related to the nondiseased population. The code and respective summary follow.

```
R> set.seed(123, "L'Ecuyer-CMRG") # for reproducibility
```

```
R> AROC_bnp <- AROC.bnp(
+ formula.h = bmi ~ gender + f(age, by = gender, K = c(0,0))
+ group = "cvd_idf", tag.h = 0, data = endosyn, standardise = TRUE,
+ p = seq(0, 1, l = 101), ci.level = 0.95, compute.lpml = TRUE, compute.WAIC = TRUE,
+ compute.DIC = TRUE, pauc = pauccontrol(compute = FALSE),
+ prior.h = priorcontrol.bnp(L = 10), density = densitycontrol(compute = TRUE),
+ mcmc = mcmccontrol(nsave = 8000, nburn = 2000, nskip = 1),
+ parallel = "snow", ncpus = 2)

R> summary(AROC_bnp)

Call: [...]

Approach: AROC Bayesian nonparametric
---------------------------------------------
Area under the covariate-adjusted ROC curve: 0.656 (0.629, 0.684)*
 * Credible level:  0.95

Model selection criteria:
                     Group H
WAIC               11833.000
WAIC (Penalty)        31.236
LPML               -5916.766
DIC                11829.750
DIC (Penalty)         29.611


Sample sizes:
                         Group H     Group D
Number of observations      2149         691
Number of missing data         0           0
```

The area under the AROC curve is 0.656 (95% credible interval: $(0.629, 0.684)$) thus revealing a reasonable good ability of the BMI to detect the presence of CVD risk factors when teasing out the age and gender effects. As for the pooled ROC curve and the covariate-specific ROC curve, a `plot` function is also available (result in Figure 11a).

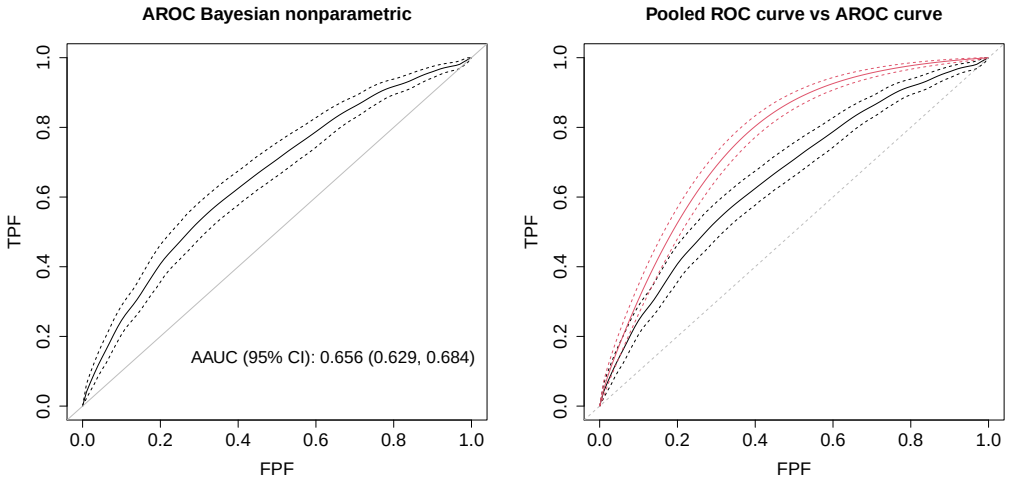
```
R> plot(AROC_bnp, cex.main = 1.5, cex.lab = 1.5, cex.axis = 1.5, cex = 1.3)
```

Finally, we compare the AROC curve with the pooled ROC curve that was obtained earlier by using a DPM model with 10 components in each group. In Figure 11b, we show the plots of the two curves, and, as can be noticed, the pooled ROC curve lies well above the AROC curve, thus evidencing the need for incorporating covariate information into the analysis.

```
R> plot(AROC_bnp$p, AROC_bnp$ROC[,1], type = "l", xlim = c(0,1), ylim = c(0,1),
+ xlab = "FPF", ylab = "TPF", main = "Pooled ROC curve vs AROC curve", cex.main = 1.5,
+ cex.lab = 1.5, cex.axis = 1.5, cex = 1.5)
R> lines(AROC_bnp$p, AROC_bnp$ROC[,2], col = 1, lty = 2)
R> lines(AROC_bnp$p, AROC_bnp$ROC[,3], col = 1, lty = 2)
R> lines(pROC_dpm$p, pROC_dpm$ROC[,1], col = 2)
R> lines(pROC_dpm$p, pROC_dpm$ROC[,2], col = 2, lty = 2)
R> lines(pROC_dpm$p, pROC_dpm$ROC[,3], col = 2, lty = 2)
R> abline(0, 1, col = "grey", lty = 2)
```

Computational aspects

We finish this section with some comments on computational aspects. In our experience, the methods with the largest computing times are those implemented in `cROC.bnp` when $L_{\bar{D}} > 1$ and in `cROC.kernel` when confidence bands are to be constructed. In the first case, the main reason behind the computational burden is the need to invert $F_{\bar{D}}(\cdot \mid \mathbf{x})$ in order to obtain the covariate-specific ROC curve (see equation (10)). Note that when $L_{\bar{D}} > 1$, the conditional distribution function in the nondiseased group is given by a mixture of normal distributions, and the corresponding quantile function needs to be computed for each covariate(s) value we might be interested in and for each iteration of the Gibbs sampler procedure. Regarding the `cROC.kernel` function, the computing time is mainly driven by the number of bootstrap samples used for constructing the confidence bands. In Table 2, we show the



(a) AROC curve

(b) AROC vs Poooled ROC curve

Figure 11: (a) Age and gender-adjusted ROC curve: posterior mean and 95% pointwise credible band. (b) Age and gender-adjusted ROC curve (in black) and pooled ROC curve (estimated using a DPM of normals model) (in red). Solid lines represent the posterior means and dashed lines the 95% pointwise credible bands.

time, in seconds, needed for fitting the pooled, the covariate-specific, and the covariate-adjusted ROC curve using the Bayesian nonparametric and the kernel approaches for the synthetic endocrine data and when both parallel (with 2 and 4 processes) and no parallel options are used. We note that for the Bayesian approaches, we also computed the densities/conditional densities, as well as the WAIC, LPML, and DIC, which further increase the computing time (in the case of the AROC curve, these were only computed in the nondiseased population). With respect to the kernel-based approach (in this case, the fit is done separately for men and women and the corresponding results are presented in the Supplementary Material), we have used 500 bootstrap samples to construct the confidence bands. As it can be appreciated, for these two intense tasks, using 4 processes drastically improves the computation time. All computations were performed in a iMac with 3.6GHz quad core Intel i7 processor and 32GB RAM running under a macOS Catalina 10.15.5 operating system.

	No parallel	Snow (2 cores)	Snow (4 cores)
`pooledROC.dpm`	138	118	111
`pooledROC.kernel`	376	196	105
`cROC.bnp`	2052	1117	680
`cROC.kernel`	Men: 1159 Women: 1885	Men: 528 Women: 916	Men: 279 Women: 466
`AROC.bnp`	126	115	112
`AROC.kernel`	Men: 847 Women: 1707	Men: 404 Women: 833	Men: 214 Women: 438

Table 2: Time in seconds (rounded to the nearest second) needed to fit the pooled, the covariate-specific, and the covariate-adjusted ROC curve for the Bayesian nonparametric and kernel approaches.

Summary and future plans

In this paper, we have introduced the capabilities of the R package **ROCnReg** for conducting inference about the pooled ROC curve, the covariate-specific ROC curve, and the covariate-adjusted ROC curve and their associated summary indices. As we have illustrated, the current version of the package provides several options for estimating ROC curves, both under frequentist and Bayesian paradigms, either parametrically, semiparametrically, or nonparametrically. To the best of our knowledge, this is the first software package implementing Bayesian inference for ROC curves. Several additions/extensions are planned in the future, and these, among others, include:

- Implement the most time-consuming parts in C or C++.
- Incorporate methods for non-binary disease status (e.g., no disease, mild disease, severe disease). That is, implement ROC surface models.
- Implement new (optimal) threshold criteria (e.g., YI including costs).

Computational details

The results in this paper were obtained using R 4.0.3 with the **ROCnReg** 1.0-5 package. The **ROCnReg** package has several dependencies: **graphics**, **grDevices**, **parallel**, **splines**, **stats**, **moments** (Komsta and Novomestky, 2015), **nor1mix** (Maechler, 2019), **Matrix** (Bates and Maechler, 2019), **spatstat** (Baddeley and Turner, 2005), **np** (Hayfield and Racine, 2008), **lattice** (Sarkar, 2008), **MASS** (Venables and Ripley, 2002), and **pbivnorm** (Genz and Kenkel, 2015). R itself and all packages used are available from the Comprehensive R Archive Network (CRAN) at https://CRAN.R-project.org/.

Acknowledgements

We acknowledge the reviewer for their constructive comments that led to an improved version of the article. MX Rodríguez-Álvarez was funded by project MTM2017-82379-R (AEI/FEDER, UE), by the Basque Government through the BERC 2018-2021 program and Elkartek project 3KIA (KK-2020/00049) and by the Spanish Ministry of Science, Innovation, and Universities (BCAM Severo Ochoa accreditation SEV-2017-0718).

Bibliography

A. Baddeley and R. Turner. spatstat: An R package for analyzing spatial point patterns. *Journal of Statistical Software*, 12(6):1–42, 2005. URL http://doi.org/10.18637/jss.v012.i06. [p552]

D. Bates and M. Maechler. *Matrix: Sparse and Dense Matrix Classes and Methods*, 2019. URL https://CRAN.R-project.org/package=Matrix. R package version 1.2-18. [p552]

G. Celeux, F. Forbes, C. P. Robert, and D. M. Titterington. Deviance information criteria for missing data models. *Bayesian Analysis*, 1(4):651–673, 2006. URL https://doi.org/10.1214/06-BA122. [p533]

M. De Iorio, W. O. Johnson, P. Müller, and G. L. Rosner. Bayesian nonparametric nonproportional hazards survival modeling. *Biometrics*, 65(3):762–771, 2009. URL https://doi.org/10.1111/j.1541-0420.2008.01166.x. [p533]

A. Erkanli, M. Sung, E. Jane Costello, and A. Angold. Bayesian semi-parametric ROC analysis. *Statistics in Medicine*, 25(22):3905–3928, 2006. URL https://doi.org/10.1002/sim.2496. [p526, 531, 532, 534]

D. Faraggi. Adjusting receiver operating characteristic curves and related indices for covariates. *Journal of the Royal Statistical Society D*, 52(2):179–192, 2003. URL https://doi.org/10.1111/1467-9884.00350. [p525, 526, 534]

T. S. Ferguson. A Bayesian analysis of some nonparametric problems. *The Annals of Statistics*, 1(2): 209–230, 1973. URL https://doi.org/10.1214/aos/1176342360. [p531]

J. Gabry, D. Simpson, A. Vehtari, M. Betancourt, and A. Gelman. Visualization in Bayesian workflow. *Journal of the Royal Statistical Society A*, 182(2):389–402, 2019. URL https://doi.org/10.1111/rssa.12378. [p546]

S. Geisser and W. F. Eddy. A predictive approach to model selection. *Journal of the American Statistical Association*, 74(365):153–160, 1979. URL https://doi.org/10.2307/2286745. [p533]

A. Gelman, J. B. Carlin, H. S. Stern, D. B. Dunson, A. Vehtari, and D. B. Rubin. *Bayesian Data Analysis*. Chapman and Hall/CRC, third edition, 2013. [p532, 538]

A. Gelman, J. Hwang, and A. Vehtari. Understanding predictive information criteria for Bayesian models. *Statistics and Computing*, 24(6):997–1016, 2014. URL https://doi.org/10.1007/s11222-013-9416-2. [p533]

A. Genz and B. Kenkel. *pbivnorm: Vectorized Bivariate Normal CDF*, 2015. URL https://CRAN.R-project.org/package=pbivnorm. R package version 0.6.0. [p552]

J. Gu, S. Ghosal, and A. Roy. Bayesian bootstrap estimation of ROC curve. *Statistics in Medicine*, 27: 5407–5420, 2008. URL https://doi.org/10.1002/sim.3366. [p526, 530]

Z. Guan, J. Qin, and B. Zhang. Information borrowing methods for covariate-adjusted ROC curve. *Canadian Journal of Statistics*, 40(3):569–587, 2012. URL https://doi.org/10.1002/cjs.11145. [p525]

T. Hayfield and J. S. Racine. Nonparametric econometrics: The np package. *Journal of Statistical Software*, 27(5), 2008. URL https://doi.org/10.18637/jss.v027.i05. [p552]

S. L. Hillis and C. E. Metz. An analytic expression for the binormal partial area under the ROC curve. *Academic Radiology*, 19(12):1491 – 1498, 2012. URL https://doi.org/10.1016/j.acra.2012.09.009. [p532]

F. Hsieh and B. W. Turnbull. Nonparametric and semiparametric estimation of the receiver operating characteristic curve. *The Annals of Statistics*, 24(1):25–40, 02 1996. URL https://doi.org/10.1214/aos/1033066197. [p526]

V. Inácio, M. X. Rodríguez-Álvarez, and P. Gayoso-Diz. Statistical evaluation of medical tests. *Annual Review of Statistics and Its Application*, 8, 2020. URL https://doi.org/10.1146/annurev-statistics-040720-022432. [p525]

V. Inácio de Carvalho and M. X. Rodríguez-Álvarez. Bayesian nonparametric inference for the covariate-adjusted ROC curve. *arXiv:1806.00473 [stat.ME]*, 2018. URL https://arxiv.org/abs/1806.00473. [p525, 526, 530, 534, 535]

V. Inácio de Carvalho, A. Jara, T. E. Hanson, and M. de Carvalho. Bayesian nonparametric ROC regression modeling. *Bayesian Analysis*, 8(3):623–646, 2013. URL https://doi.org/10.1214/13-BA825. [p525, 526, 533]

V. Inácio de Carvalho, M. de Carvalho, and A. J. Branscum. Nonparametric Bayesian covariate-adjusted estimation of the Youden index. *Biometrics*, 73(4):1279–1288, 2017. URL https://doi.org/10.1111/biom.12686. [p525, 534]

H. Ishwaran and L. F. James. Gibbs sampling methods for stick-breaking priors. *Journal of the American Statistical Association*, 96(453):161–173, 2001. URL https://doi.org/10.1198/016214501750332758. [p531]

H. Ishwaran and L. F. James. Approximate Dirichlet process computing in finite normal mixtures: Smoothing and prior information. *Journal of Computational and Graphical Statistics*, 11(3):508–532, 2002. URL https://doi.org/10.1198/106186002411. [p532]

H. Janes and M. S. Pepe. Adjusting for covariate effects on classification accuracy using the covariate-adjusted receiver operating characteristic curve. *Biometrika*, 96(2):371–382, 2009. URL https://doi.org/10.1093/biomet/asp002. [p525, 526, 529, 530]

L. Komsta and F. Novomestky. *moments: Moments, cumulants, skewness, kurtosis and related tests*, 2015. URL https://CRAN.R-project.org/package=moments. R package version 0.14. [p552]

M. López-Ratón, M. X. Rodríguez-Álvarez, C. Cadarso-Suárez, and F. Gude-Sampedro. OptimalCutpoints: an R package for selecting optimal cutpoints in diagnostic tests. *Journal of Statistical Software*, 61(8):1–36, 2014. URL https://doi.org/10.18637/jss.v061.i08. [p525]

S. N. MacEachern. Dependent Dirichlet processes. *Unpublished manuscript, Department of Statistics, The Ohio State University*, pages 1–40, 2000. [p533]

M. Maechler. *nor1mix: Normal aka Gaussian (1-d) Mixture Models (S3 Classes and Methods)*, 2019. URL https://CRAN.R-project.org/package=nor1mix. R package version 1.3-0. [p552]

C. E. Metz. Basic principles of ROC analysis. *Seminars in Nuclear Medicine*, 8(4):283 – 298, 1978. URL https://doi.org/10.1016/S0001-2998(78)80014-2. [p525]

J. C. Pardo-Fernández, M. X. Rodríguez-Álvarez, and I. van Keilegom. A review on ROC curves in the presence of covariates. *REVSTAT–Statistical Journal*, 12(1):21–41, 2014. [p525, 530]

M. S. Pepe. Three approaches to regression analysis of receiver operating characteristic curves for continuous test results. *Biometrics*, 54(1):124–135, 1998. URL https://doi.org/10.2307/2534001. [p525, 526]

M. S. Pepe. *The Statistical Evaluation of Medical Tests for Classification and Prediction*. Oxford Statistical Sciences Series. Oxford University Press, New York, 2003. [p525, 527, 528, 530]

S. Pérez Fernández, P. Martínez Camblor, P. Filzmoser, and N. Corral Blanco. nsROC: An R package for non-standard ROC curve analysis. *The R Journal*, 10(2), 2018. URL https://doi.org/10.32614/RJ-2018-043. [p525]

S. Perez Jaume, K. Skaltsa, N. Pallarès, and J. L. Carrasco. ThresholdROC: Optimum threshold estimation tools for continuous diagnostic tests in R. *Journal of Statistical Software*, 82(4):1–21, 2017. URL https://doi.org/10.18637/jss.v082.i04. [p525]

M. Plummer, N. Best, K. Cowles, and K. Vines. Coda: Convergence diagnosis and output analysis for MCMC. *R News*, 6(1):7–11, 2006. URL https://journal.r-project.org/archive/. [p538]

X. Robin, N. Turck, A. Hainard, N. Tiberti, F. Lisacek, J.-C. Sanchez, and M. Müller. proc: An open-source package for R and S+ to analyze and compare ROC curves. *BMC Bioinformatics*, 12:77, 2011. URL https://doi.org/10.1186/1471-2105-12-77. [p525]

M. X. Rodriguez-Alvarez and J. Roca-Pardinas. *npROCRegression: Kernel-Based Nonparametric ROC Regression Modelling*, 2017. URL https://CRAN.R-project.org/package=npROCRegression. R package version 1.0-5. [p525]

M. X. Rodríguez-Álvarez, J. Roca-Pardiñas, and C. Cadarso-Suárez. ROC curve and covariates: Extending induced methodology to the non-parametric framework. *Statistics and Computing*, 21(4): 483–499, 2011a. URL https://doi.org/10.1007/s11222-010-9184-1. [p525, 526, 535, 542]

M. X. Rodríguez-Álvarez, J. Roca-Pardiñas, and C. Cadarso-Suárez. A new flexible direct ROC regression model: Application to the detection of cardiovascular risk factors by anthropometric measures. *Computational Statistics & Data Analysis*, 55(12):3257–3270, 2011b. URL https://doi.org/10.1016/j.csda.2011.06.008. [p525, 535, 542]

M. X. Rodríguez-Álvarez, P. G. Tahoces, C. Cadarso-Suárez, and M. J. Lado. Comparative study of ROC regression techniques – applications for the computer-aided diagnostic system in breast cancer detection. *Computational Statistics & Data Analysis*, 55(1):888–902, 2011c. URL https://doi.org/10.1016/j.csda.2010.07.018. [p525]

P. S. Rosenberg. Hazard function estimation using B-splines. *Biometrics*, 51(3):874–887, 1995. URL https://doi.org/10.2307/2532989. [p533]

D. Sarkar. *Lattice: Multivariate Data Visualization with R*. Springer-Verlag, New York, 2008. URL http://lmdvr.r-forge.r-project.org. ISBN 978-0-387-75968-5. [p552]

D. E. Shapiro. The interpretation of diagnostic tests. *Statistical Methods in Medical Research*, 8(2):113–134, 1999. URL https://doi.org/10.1016/s0004-9514(14)60228-2. [p528]

M. A. Tomé Martínez de Rituerto, M. A. Botana, C. Cadarso-Suárez, A. Rego-Iraeta, A. Fernández-Mariño, J. A. Mato, I. Solache, and R. Perez-Fernandez. Prevalence of metabolic syndrome in Galicia (NW Spain) on four alternative definitions and association with insulin resistance. *Journal of Endocrinological Investigation*, 32(6):505–511, Jun 2009. URL https://doi.org/10.1007/BF03346497. [p535]

W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S*. Springer, New York, fourth edition, 2002. URL http://www.stats.ox.ac.uk/pub/MASS4. ISBN 0-387-95457-0. [p552]

X.-F. Wang. *sROC: Nonparametric Smooth ROC Curves for Continuous Data*, 2012. URL https://CRAN.R-project.org/package=sROC. R package version 0.1-2. [p525]

H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. ISBN 978-3-319-24277-4. URL https://ggplot2.tidyverse.org. [p538]

W. J. Youden. Index for rating diagnostic tests. *Cancer*, 3(1):32–35, 1950. URL https://doi.org/10.1002/1097-0142(1950)3:1<32::AID-CNCR2820030106>3.0.CO;2-3. [p528]

X.-H. Zhou, D. K. McClish, and N. A. Obuchowski. *Statistical Methods in Diagnostic Medicine*. John Wiley & Sons, New York, second edition, 2011. [p525]

K. H. Zou, W. J. Hall, and D. E. Shapiro. Smooth non-parametric receiver operating characteristic (ROC) curves for continuous diagnostic tests. *Statistics in Medicine*, 16(19):2143–2156, 1997. URL https://doi.org/10.1002/(sici)1097-0258(19971015)16:19<2143::aid-sim655>3.0.co;2-3. [p526]

María Xosé Rodríguez-Álvarez
BCAM - Basque Center for Applied Mathematics and IKERBASQUE, Basque Foundation for Science
Spain
ORCID: 0000-0002-1329-9238
mxrodriguez@bcamath.org

Vanda Inácio
School of Mathematics
University of Edinburgh
Scotland, UK
ORCID: 0000-0001-8084-1616
vanda.inacio@ed.ac.uk

Automating Reproducible, Collaborative Clinical Trial Document Generation with the listdown Package

by Michael Kane, Xun Jiang and Simon Urbanek

Abstract The conveyance of clinical trial explorations and analysis results from a statistician to a clinical investigator is a critical component of the drug development and clinical research cycle. Automating the process of generating documents for data descriptions, summaries, exploration, and analysis allows the statistician to provide a more comprehensive view of the information captured by a clinical trial, and efficient generation of these documents allows the statistican to focus more on the conceptual development of a trial or trial analysis and less on the implementation of the summaries and results on which decisions are made. This paper explores the use of the **listdown** package for automating reproducible documents in clinical trials that facilitate the collaboration between statisticians and clinicians as well as defining an analysis pipeline for document generation.

Background and Introduction

The conveyance of clinical trial explorations and analysis results from a statistician to a clinical investigator is an often overlooked but critical component to the drug development and clinical research cycle. Graphs, tables, and other analysis artifacts are at the nexus of these collaborations. They facilitate identifying problems and bugs in the data preparation and processing stage, they help to build an intuitive understanding of mechanisms of disease and their treatment, they elucidate prognostic and predictive relationships, they provide insight that results in new hypotheses, and they convince researchers of analyses testing hypotheses.

Despite their importance, the process of generating these artifacts is usually done in an ad-hoc manner. This is partially because of the nuance and diversity of the hypotheses and scientific questions being interrogated and, to a lesser degree, the variation in clinical data formatting. The usual process usually has a statistician providing a standard set of artifacts, receiving feedback, and providing updates based on feedback. Work performed for one trial is rarely leveraged on others, and as a result, a large amount of work needs to be reproduced for each trial.

There are two glaring problems with this approach. First, each analysis of a trial requires a substantial amount of error-prone work. While the variation between trials means some work needs to be done for preparation, exploration, and analysis, many aspects of these trials could be better automated resulting in greater efficiency and accuracy. Second, because this work is challenging, it often occupies the majority of the statisticians' effort. Less time is spent on trial design and analysis, and then this portion is taken up by a clinician who often has less expertise with the statistical aspects of the trial. As a result, the extra effort spent on processing data undermines statisticians' role as a collaborator and relegates them to service provider. Need tools leveraging existing work to more efficiently provide holistic views on trials will result in less effort and more accurate and comprehensive trial design and analysis.

The richness of the R Core Team (2012) package ecosystem, particularly with its emphasis on analysis, visualization, reproducibility, and dissemination makes the goal of creating these tools for clinical trials feasible. Generation of tables is supported by packages including **tableone** (Yoshida and Bartel, 2020), **gt** (Iannone et al., 2020), **gtsummary** (Sjoberg et al., 2020). Visualization is achieved using package including **ggplot2** (Wickham, 2016) and **survminer** (Kassambara et al., 2020). We can even provide interactive presentations of data with **DT** (Xie et al., 2020), **plotly** (Sievert, 2020), and **trelliscopejs** (Hafen and Schloerke, 2020).

It should also be realized that work building on these tools for clinical trial data is already in process. The **greport** (Harrell Jr, 2020) package provides graphical summaries for clinical trials and has been used in conjunction with **rmarkdown** (Allaire et al., 2020) to produce specific trial report types with a specified format.

Using listdown for programmatic, collaborative clinical trial document generation

The **listdown** package (Kane et al., 2020) was recently released to automate the process of generating reproducible (RMarkdown) documents. Objects derived from a summary, exploration, or analysis are stored hierarchically in an R `list`, which defines the structure of the document. These objects are referred to as *computational components* since they are derived from computation, as opposed to prose, which makes up the *narrative components* of a document.

The computational components capture and structure the objects to be presented. Describing how the objects will be presented and how the document will be rendered is handled through the creation of a `listdown` object. The separation between how computational components are created and how they are shown to a user provides two advantages. First, it decouples the data processing and analysis from its exploration and visualization. For compute-intensive analyses, this separation is critical for avoiding redundant computations for small changes in the presentation. It also discourages putting compute-intensive code into RMarkdown documents. Second, it provides the flexibility to quickly change how a computational component is visualized or summarized or even how a document is rendered. This makes transitioning from an interactive .html document to a static .pdf document significantly easier than substituting functions and parameters in an R Markdown document.

The package has been found to be particularly useful in the reporting and research of clinical trial data. In particular, the package has been used for server collaborations focusing on either the analysis of past trial data to formulate a new trial or in trial monitoring where trial telemetry (enrollment, responses, etc.) is reported, and initial analyses are conveyed to a clinician. The associated presentations require very little context since clinicians often have as good an understanding of the data collected as that of the statistician's meaning narrative components are not needed. At the same time, a large number of hierarchical, heterogeneous artifacts (tables and multiple types of plots) can be automated where manual creation of RMarkdown documents would be inconvenient and inefficient.

The rest of this document describes concepts implemented in the **listdown** package for automated, reproducible document generation and shows its use with a simplified, synthetic clinical trial data set whose variables are typical of a non-small cell lung cancer trial. The data set comes from the **forceps** (Kane, 2020) package. As of the time this document was written, the package is under development and is not available on CRAN. However, it can be installed as follows.

```
devtools::install_github("kaneplusplus/forceps")
```

The following section uses the trial data to construct a pipeline for document generation. We note that both the data and the pipeline are simple when compared to most analyses of this type. However, it is sufficient to illustrate accompanying concepts, and both the analyses and concepts translate readily to real-world applications. A final section discusses the use of the package and its current direction.

Constructing a pipeline for document generation

The process of analyzing data can be described using the classic waterfall model of Benington (1983) where the output (the analysis presentation or service) is dependent on a sequence of tasks that come before it. This dependency structure means that if a problem is detected in a given stage of the production of the analysis, all downstream parts must be rerun to reflect the change. A graphical depiction of the waterfall model, specific to data analyses (clinical or otherwise) is shown in Figure 1. Note that data exploration and visualization are an integral part of all stages of the production and are often the means for identifying issues and refining analyses.

As explained in the previous section, we are going to implement a simple analysis pipeline. The data acquisition and preprocessing steps are handled by importing data sets from the **forceps** package and using some of the functions implemented in the package to create a single trial data set, thereby de-emphasizing these components in the pipeline. While these steps are critical, the emphasis of this paper is the incorporation of the **listdown** package into the later stages.

Data acquisision and preprocessing

Data acquisition refers to the portion of the analysis pipeline where the data is retrieved from some managed data store for integration into the pipeline. These data sets may be retrieved as tables from a database, case reports, Analysis Data Model (ADaM) data formatted according to the Clinical Data Interchange Standards Consortium (CDISC) (CDI, 2020), Electronic Health Records, or other clinical Real World Data (RWD) formats. These data are then transformed to a format appropriate for analysis.

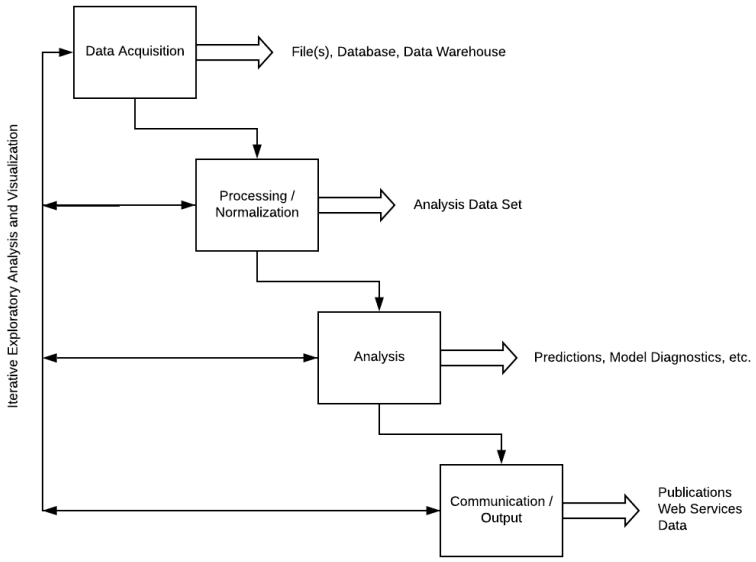


Figure 1: The data analysis waterfall.

In our simple example, this is accomplished by loading data corresponding to trial outcomes, patient adverse events, patient biomarkers, and patient demography and transforming them into a single data set with one row per patient and one variable per column using the **forceps** and **dplyr** (Wickham et al., 2020) packages. The data also includes longitudinal adverse event information, which will is stored as a nested data frame in the `ae_long` column of the resulting data set.

```
library(forceps)
library(dplyr)

data(lc_adsl, lc_adverse_events, lc_biomarkers, lc_demography)

lc_trial <- consolidate(
  list(adsl = lc_adsl,
       adverse_events = lc_adverse_events %>%
         cohort(on = "usubjid", name = "ae_long"),
       biomarkers = lc_biomarkers,
       demography = lc_demography %>%
         select(-chemo_stop)
  ),
  on = "usubjid")

lc_trial
```

```
#> # A tibble: 558 x 18
#>    usubjid best_response  pfs_days pfs_censor os_days os_censor chemo_stop arm
#>      <dbl> <chr>             <dbl>      <dbl>   <dbl>     <dbl> <chr>      <chr>
#>  1    1003 Stable Disease      101          1     233         1 adverse e~ stan~
#>  2    1005 Complete Resp~       78          0     184         1 adverse e~ stan~
#>  3    1006 Progressive D~      253          1     333         1 treatment~ stan~
#>  4    1009 Partial Respo~      130          0     643         0 <NA>       trea~
#>  5    1014 Complete Resp~       41          1     116         1 adverse e~ trea~
#>  6    1018 Partial Respo~      194          1     423         1 treatment~ stan~
#>  7    1023 Stable Disease       49          1     337         1 adverse e~ stan~
#>  8    1025 Stable Disease       95          1     589         1 adverse e~ stan~
#>  9    1030 Complete Resp~      351          1     688         1 adverse e~ stan~
#> 10    1033 Complete Resp~       33          1     125         1 treatment~ stan~
#> # ... with 548 more rows, and 10 more variables: ae_count <int>,
#> #   ae_long <list>, egfr_mutation <chr>, smoking <chr>, ecog <chr>,
#> #   prior_resp <chr>, site_id <int>, sex <chr>, refractory <lgl>, age <dbl>
```

Analysis defining and structuring the computational components

The next step is to define the computational components as a hierarchical, named list of objects that will be presented. These objects are generally derived from the exploratory, descriptive, and analysis stages and are presented as visualizations and tables. This example will focus on the exploratory and descriptive portions. We will start by defining a new function `class_and_tag()`, which takes an object and prepends a class designation to the objects along with optionally associating the object with a set of attributes. This extra information will be used later in order to dispatch to the proper functions responsible for presenting the objects in a resulting R Markdown document. If no such information is given, then the object will be presented as it usually would in the document. It should also be noted that `class_and_tag()` is provided for convenience here and will be available in subsequent package versions.

The next step is to define the computational components that will be presented. Our simple example will contain three sections: Outcome Information, Adverse Events, Biomarkers. The Adverse Events and Biomarkers sections will each show summary tables of one of the variables from those data. The Outcome Information section will be composed of two subsections, with the first (Best Response) providing a summary of the best response by the arm and the second (Overall Survival) showing a survival plot by arm of the overall survival. The call to `ld_cc_dendro()` at the end of the example provides of dendrogram of the hierarchical structure.

```
library(listdown)

trial_summary <- list(
  `Outcome Information` = list(
    `Best Response` = lc_trial %>%
      select(best_response, arm) %>%
      class_and_tag("summary_table", by = "arm"),
    `Overall Survival` = lc_trial %>%
      select(os_days, os_censor, arm) %>%
      class_and_tag("survival_plot",
                    time = "os_days",
                    censor = "os_censor",
                    x = "arm")),
  `Adverse Events` = lc_trial %>%
    select(best_response) %>%
    class_and_tag("summary_table"),
  `Biomarkers` = list(
    `EGFR` = lc_trial %>%
      select(egfr_mutation) %>%
      class_and_tag("summary_table")
  )
)

ld_cc_dendro(trial_summary)

#>
#> trial_summary
#>   |-- Outcome Information
#>    |-- Best Response
#>    |  o-- object of type(s):summary_table tbl_df tbl data.frame
#>    o-- Overall Survival
#>       o-- object of type(s):survival_plot tbl_df tbl data.frame
#>   |-- Adverse Events
#>   |  o-- object of type(s):summary_table tbl_df tbl data.frame
#>   o-- Biomarkers
#>    o-- EGFR
#>       o-- object of type(s):summary_table tbl_df tbl data.frame
```

Communicating results

The objects have been constructed and they have been structured. The next step is to define how they will be presented. This is done by creating two functions `make_summary` and `make_survival_plot`. The former takes a `data.frame` and uses the **gtsummary** package to summarize the results. If the `data.frame` includes an attribute named `by` and denoting a valid variable in the data set, then the

summary is created conditionally on that variable. The latter function also takes a `data.frame` and uses attributes to denote variables on which a Kaplan-Meier plot can be constructed using the **survival** (Terry M. Therneau and Patricia M. Grambsch, 2000) and **survminer** packages. The functions are written to a file called `decorators.r` and will be used by the R Markdown document to render the final document.

```
library(gtsummary)
library(survival)
library(survminer)
library(dplyr)

make_summary <- function(x) {
  by <- attributes(x)$by
  tbl_summary(x, by = by)
}

make_survival_plot <- function(.x) {
  att <- attributes(.x)
  x <- ifelse(is.null(att$x), "1", att$x)
  form <- sprintf("Surv(%s, %s) ~ %s", att$time, att$censor, x) %>%
    as.formula()
  fit <- surv_fit(form, data = as.data.frame(.x))
  ggsurvplot(fit, data = as.data.frame(.x))
}
```

The next step is to connect the computational components to the functions that will present them using the `listdown()` function. First, the computational components are stored in the `trial_summary` object are written to the disk. The resulting R Markdown document will read it in based on the `load_cc_expr` argument. Along with reading in the data, initialization in the R Markdown document will include sourcing the `decorators.r` file previously written to disk. This is handled with the `init_expr` argument. The `decorators` argument takes a list where the name specifies the class and the list element corresponds to a function that will present objects of the specified class. For example, `summary_table` objects are sent to the `make_summary()` function for presentation. This allows us to connect objects to their appropriate function to process and present them. Finally, `echo` and `message` chunk options are set to `FALSE` so that the code and associated messages will not appear in the final rendered document. The last line of code below displays the first 12 lines of R Markdown code chunks as they will appear in the corresponding document.

```
saveRDS(trial_summary, "cc.rds")

ld <- listdown(load_cc_expr = readRDS("cc.rds"),
               init_expr = source("decorators.r"),
               decorator = list(summary_table = make_summary,
                                survival_plot = make_survival_plot),
               echo = FALSE,
               message = FALSE)

ld_make_chunks(ld)[1:12]

#>  [1] ""
#>  [2] "```{r echo = FALSE, message = FALSE}"
#>  [3] "source(\"decorators.r\")"
#>  [4] ""
#>  [5] "cc_list <- readRDS(\"cc.rds\")"
#>  [6] "```"
#>  [7] ""
#>  [8] "# Outcome Information"
#>  [9] ""
#> [10] "## Best Response"
#> [11] ""
#> [12] "```{r echo = FALSE, message = FALSE}"
```

The last step creates the R Markdown header, writes it along with the R code chunks to a file named `"simple-data-trial-summary.rmd"` and knits the file with the **knitr** package (Xie, 2020) to a pdf document, per the `output` argument of the `ld_rmarkdown_header()` function. `md_header` is a `yml`

object making it easy to control how the document is rendered. The code to generate the document along with the resulting R Markdown and output file constitute a reproducible workflow that can be quickly iterated upon and adapted for different types of explorations, summaries, and analyses.

```
library(knitr)

md_header <- ld_rmarkdown_header("Fake Data Trial Summary",
                                 output = "pdf_document")

ld_write_file(
  rmd_header = as.character(md_header),
  ld = ld_make_chunks(ld),
  file_name = "simple-data-trial-summary.rmd")

knit("simple-data-trial-summary.rmd")
```

Direction

The **listdown** package has been successfully used for collaborations on several clinical trial analyses. The package works particularly well when creating large, navigable sets of summaries and visualizations. Current work focuses on two areas. First is the construction of standard decorators. By packaging decorators and associated functionality, presentations can be made rich as they are developed over time. This standardization also makes it easier to leverage work in configuring chunks so that they can be made more aesthetic by default. In addition, we have been working on abstract and formalize the notion of document composition. In the example presented, the aggregation of the header and R code chunks into a file was sufficient for generating a document. However, the composition of other outputs is also supported. For example, the top-level names of the trial summary could just as easily designate tabs on a web page or other target. The notion of a composer would allow a user to target a greater variety of output types, better suiting the application under consideration and the target audience for the presentation.

Bibliography

Clinical data interchange standards consortium. https://www.cdisc.org/, 2020. Accessed: 2020-10-25. [p557]

J. Allaire, Y. Xie, J. McPherson, J. Luraschi, K. Ushey, A. Atkins, H. Wickham, J. Cheng, W. Chang, and R. Iannone. *rmarkdown: Dynamic Documents for R*, 2020. URL https://github.com/rstudio/rmarkdown. R package version 2.5. [p556]

H. D. Benington. Production of large computer programs. *Annals of the History of Computing*, 5(4): 350–361, 1983. doi: 10.1109/MAHC.1983.10102. [p557]

R. Hafen and B. Schloerke. *trelliscopejs: Create Interactive Trelliscope Displays*, 2020. URL https://CRAN.R-project.org/package=trelliscopejs. R package version 0.2.5. [p556]

F. E. Harrell Jr. *greport: Graphical Reporting for Clinical Trials*, 2020. URL https://CRAN.R-project.org/package=greport. R package version 0.7-2. [p556]

R. Iannone, J. Cheng, and B. Schloerke. *gt: Easily Create Presentation-Ready Display Tables*, 2020. URL https://CRAN.R-project.org/package=gt. R package version 0.2.2. [p556]

M. J. Kane. *forceps: A grammar for manipulating clinical trial data*, 2020. URL https://github.com/kaneplusplus/forceps. R package version 0.0.2. [p557]

M. J. Kane, X. T. Jiang, and S. Urbanek. On the programmatic generation of reproducible documents, 2020. [p557]

A. Kassambara, M. Kosinski, and P. Biecek. *survminer: Drawing Survival Curves using 'ggplot2'*, 2020. URL https://CRAN.R-project.org/package=survminer. R package version 0.4.8. [p556]

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2012. URL http://www.R-project.org/. ISBN 3-900051-07-0. [p556]

C. Sievert. *Interactive Web-Based Data Visualization with R, plotly, and shiny*. Chapman and Hall/CRC, 2020. ISBN 9781138331457. URL https://plotly-r.com. [p556]

D. D. Sjoberg, M. Curry, M. Hannum, K. Whiting, and E. C. Zabor. *gtsummary: Presentation-Ready Data Summary and Analytic Result Tables*, 2020. URL https://CRAN.R-project.org/package=gtsummary. R package version 1.3.5. [p556]

Terry M. Therneau and Patricia M. Grambsch. *Modeling Survival Data: Extending the Cox Model*. Springer, New York, 2000. ISBN 0-387-98784-3. [p560]

H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. ISBN 978-3-319-24277-4. URL https://ggplot2.tidyverse.org. [p556]

H. Wickham, R. François, L. Henry, and K. Müller. *dplyr: A Grammar of Data Manipulation*, 2020. URL https://CRAN.R-project.org/package=dplyr. R package version 1.0.2. [p558]

Y. Xie. *knitr: A General-Purpose Package for Dynamic Report Generation in R*, 2020. URL https://yihui.org/knitr/. R package version 1.30. [p560]

Y. Xie, J. Cheng, and X. Tan. *DT: A Wrapper of the JavaScript Library 'DataTables'*, 2020. URL https://CRAN.R-project.org/package=DT. R package version 0.16. [p556]

K. Yoshida and A. Bartel. *tableone: Create 'Table 1' to Describe Baseline Characteristics with or without Propensity Score Weights*, 2020. URL https://CRAN.R-project.org/package=tableone. R package version 0.12.0. [p556]

Michael Kane
Yale University
60 College Street
New Haven, CT 06510, USA

michael.kane@yale.edu

Xun Jiang
Amgen Inc.
One Amgen Center Drive
Thousand Oaks, CA 91320-1799, USA

xunj@amgen.com

Simon Urbanek
The University of Aukland
38 Princes Street
Auckland Central, Aukland 1010, NZ

s.urbanek@auckland.ac.nz

Towards a Grammar for Processing Clinical Trial Data

by Michael J. Kane

Abstract The goal of this paper is to help define a path toward a grammar for processing clinical trials by a) defining a format in which we would like to represent data from standardized clinical trial data b) describing a standard set of operations to transform clinical trial data into this format, and c) to identify a set of verbs and other functionality to facilitate data processing and encourage reproducibility in the processing of these data. It provides a background on standard clinical trial data and goes through a simple preprocessing example illustrating the value of the proposed approach through the use of the **forceps** package, which is currently being used for data of this kind.

Introduction: On the use of historical clinical data

There are few areas of data science research that provide more promise to improve human quality-of-life and treat a disease than the development of methods and analysis in clinical trials. While adjacent, data-focused areas of biomedicine and health related research have recently seen increased attention, especially the analysis of real-world evidence (RWE) and electronic health records (EHR) in particular, clinical trial data maintains several distinct quality advantages, enumerated here.

1. Features and measurements are selected for their relevance. Unlike EHR's or other similar data, variables collected for a clinical trial are included because they are potentially relevant to the disease under consideration or the treatment whose efficacy is being analyzed. This makes the variable selection process considerably easier than that where data collection has not been designed for a targeted analysis of this type.
2. Data collection procedures are carefully prescribed. Clinical trial data is uniform in both which variables are collected and how they are collected. This ensures data quality across trial sites ensuring that variables are relatively complete as well as consistent.
3. Inclusion/Exclusion criteria define the population. Since RWE studies are observational, the populations they consider are not always well-understood due to bias in the collection process. On the other hand, clinical trial data sets are generally controlled and randomized, with well-documented inclusion and exclusion criteria.

Along with maintaining higher quality clinical trial data is more available and more easily accessible when compared to real-world data sources, which often require affiliations with appropriate research institutions as well as infrastructure and appropriate staff, including data managers, to extract data. By contrast, modern clinical trial data organizations allow users to quickly search and download thousands of trials, including anonymized patient-level information. These data sets tend to include control-arm data, which can be used to understand prognostic disease populations construct historical controls for existing trials. However, some also include treatment data, which can be used to characterize predictive patient subtypes for a given treatment, understand safety profiles for classes of drugs, and aid in the design of new trials. We note that, for oncology, Project Data Sphere (PDS, 2020) and outside of oncology, Immport (Imm, 2020) have been invaluable in our own experience by facilitating these types of analyses.

Clinical trial analysis data sets

During a clinical trial, patient-level data is collected in case report forms (CRFs). The format and data collected in these forms are prescribed in the trial design. These forms are the basis for the construction of analysis data sets and other documents that will be submitted to governing bodies, including the Food and Drug Association (FDA) and European Medicines Agency (EMA), for approval if the sponsor (party funding the trial) decides it is appropriate. The Clinical Data Interchange Standards Consortium (CDISC) (CDI, 2020) develops standards dealing with medical research data, including the submission of trial results. Adhering to these standards is necessary for a successful trial submission.

There are several data sets included with a submission that tend to be useful for analysis. This paper focuses on the Analysis Data Model (ADaM) data, which provides patient-level data, which has been validated and used for data derivation and analysis. An ADaM data set is itself composed of several data sets, including a Subject-Level Analysis Data Set (ADSL) holding analysis and treatment information. Other information, including baseline characteristics, demographic data, visit informa-

tion, etc., are held in the and Basic Data Structure (BDS) formatted data sets. Finally, adverse events are held in the Analysis Data Sets for Adverse Events (ADAE).

Challenges to analyzing these data sets

ADaM data for a clinical trial is generally made available as a set of SAS7BDAT (Shotwell et al., 2013) files. While neither the FDA nor the EMA require this format for submission nor do they requires the use of SAS (SAS Institute, 2020) for analysis, there is a heavy bias toward the data format and computing platform. This is partially because they are validated and approved by governing bodies and because a large effort has gone into their use in submissions. Packages like **sas7bdat** (Shotwell, 2014) and, more recently, **haven** (Wickham and Miller, 2020) have gone a long way to make these data sets easily accessible to R (R Core Team, 2012) users working with clinical trial data.

Despite the effort that has gone into defining a structure for the data as well as the tools implemented to aid in their analysis, the data sets themselves are not particularly easy to analyze for two reasons. First, the standard is not "tidy" as defined by Wickham et al. (2014). In particular, it is not required that each variable forms a column. In fact, multiple variables may be stored in one column, with another column acting as a key as to which variable's value is given. This case is often seen in the ADSL data set where a single column may primary and secondary endpoints. For data sets like these, the value variable is held in the Analysis Value (AVAL) if the corresponding variable is numeric, Analysis Value Character (AVALC) if the variable is a string, the Parameter Character Description (PARAMCD) column giving a shorted variable name, and the Parameter column providing a text description of the variable. As an example, consider the `adakiep.xpt` data set, which is provided as an example on the CDISC website and whose data is included in the supplementary material.

```
library(readr)

adakiep <- read_csv("adakiep.csv")
adakiep
```

```
#> # A tibble: 24 x 8
#>    USUBJID     PARAM                PARAMCD AVALC   ADY ADT        SRCDOM SRCSEQ
#>    <chr>       <chr>                <chr>   <chr> <dbl> <date>     <chr>   <dbl>
#>  1 XYZ-001-001 Death                DEATH   Y        85 2013-11-02 DS          1
#>  2 XYZ-001-001 Dialysis             DIALYS~ Y        80 2013-10-29 PR          2
#>  3 XYZ-001-001 eGFR 25 Percent Dec~ EGFRDEC N        85 2013-11-02 <NA>       NA
#>  4 XYZ-001-001 Composite AKI Endpo~ AKIEP   Y        80 2013-10-29 <NA>       NA
#>  5 XYZ-001-002 Death                DEATH   Y        82 2015-03-20 DS          1
#>  6 XYZ-001-002 Dialysis             DIALYS~ Y        73 2015-03-11 PR          2
#>  7 XYZ-001-002 eGFR 25 Percent Dec~ EGFRDEC N        82 2015-03-20 <NA>       NA
#>  8 XYZ-001-002 Composite AKI Endpo~ AKIEP   Y        73 2015-03-11 <NA>       NA
#>  9 XYZ-001-003 Death                DEATH   N        94 2010-10-12 DS          1
#> 10 XYZ-001-003 Dialysis             DIALYS~ Y        64 2010-09-12 PR          2
#> # ... with 14 more rows
```

The data set includes minimal information about the trial. However, we can infer that it is from a study focusing on kidney disease. There are four distinct endpoints, death, whether dialysis was needed, whether a 25% decrease in estimated glomerular filtration rate, indicating a decrease in kidney function. For analysis, these data will need to be re-arranged so that each endpoint has it's own column along with another column per endpoint indicating the trial day where the measurement was taken (from the ADY column).

The task of transforming these types of data into into appropriate analysis is complicated by the fact that there may be other files with relevant information with similar layout or layouts slightly more complicated if they include longitudinal information, for example. The rest of this paper focuses on shaping these types of data so that they can be quickly understood; they are amenable to many different types of analyses at the individual patient level; and they can be reformatted for an even larger class of analyses through a minimal set of verbs, including cohorting, which is introduced in this paper and is implemented in the **forceps** package (Kane, 2020). The package is currently in development and has not been released to CRAN. However, it has been tagged for prerelease on Github and can be installed with the following code.

```
devtools::install_github("kaneplusplus/forceps@v0.0.5")
```

The next section specifies the target data shape, which can be thought of as a restriction on the tidy format. The following section specifies the steps needed to prepare clinical trial data so that it conforms

to this restriction and includes an anonymized trial example. The final section provides a roadmap if near-term development as well as directions for enhancements and integration of the larger R ecosystem.

A tidy representation for a consolidated analysis data set

Clinical trial data is collected to be used in an analysis that determines whether or not a treatment for a disease provides a benefit when compared to those receiving either a placebo or the standard of care. "Benefit" is quantified by one or more endpoints, defined before the trial starts (in the *design*), which are compared across arms (treatment and placebo) at the conclusion of the trial using a statistical test. These data provide a wealth of information, and their usefulness extends well beyond the scope of the trial. For example, they can also be used to understand prognostic characteristics of the disease-population; they can be used to create a "historical control' ' for another trial; they can be used to identify patients' characteristics associated with better outcomes, etc.

As shown in the previous section, while ADaM-formatted data is structured, the structure does not lend itself to analysis without first performing some data transformations. We propose that the result of these transformations is a single data set with the following characteristics.

1. Each row corresponds to a single patient.
2. A variable with one value per patient should be included as a column variable.
3. Longitudinal, time series, or repeated measures data should be stored as an embedded `data.frame` per subject.

Data conforming to these characteristics provide several advantages to ADaM data sets. First, they are oriented towards trial analysis. Essentially, trials compare response rates between treatment and control arms. Having those values coded as their own variables in a single data set minimizes the complexity and effort that would otherwise go into extracting data from multiple files, cleaning them and joining them. Second, it minimizes the reshaping effort for other types of analyses. For example, response rates are often analyzed by the site at which patient measurements were taken in order to check for certain types of enrollment heterogeneity. The described patient-centric format can be transformed into a site-centric format by nesting or grouping on a site variable followed by the extraction of site-specific features and analyses, which can then be compared across sites. Transforming between these formats requires a single operation. Likewise, the patient-centric format can be transformed to a patient-longitudinal format by unnesting on the embedded variable holding the relevant longitudinal information. Third and finally, creating a single patient-centric data set minimizes the chance of inconsistent analyses. Primary and secondary analyses often use similar variables and may require similar preprocessing. If these preprocessing steps are performed separately for parallel analyses, then the probability that at least one of them contains an error in these steps is greater than when a validated patient-centric data set is created. It also makes it easier to provide provenance for an analysis if they are dependent on the same preprocessed data.

Processing ADaM data to reach the tidy representation

This section provides an example of how to use the functionality provided in the **forceps** package, in the order that the operations take place. The data set is provided with the package, and the variable names are taken from several example lung cancer studies. The data set has been significantly reduced in size, and some values and variable names have been preprocessed. This allows the example to remain easy to follow. It also allows us to illustrate the formation of a patient-centric data set in a single pass. In practice, this is often an iterative process, requiring several revisions as bugs are found, and hypotheses change.

The data sets used are as follows, and the task will be to create a patient-centered data set as described above.

1. `lc_adverse_events` - adverse events longitudinal data.
2. `lc_biomarkers` - patient biomarkers.
3. `lc_demography` - patient demographic information.
4. `lc_adsl` - response data.

Creating the data dictionary

SAS ADaM formatted data sets generally include extra information about variables, including a short description of each of the variables and possibly formatting information. The **haven** package

keeps this as attributes of each of the columns of a `tibble` that is read from these files. The **forceps** package is capable of extracting this meta-information to create a `tibble` that can be used as a data dictionary using the `consolidated_describe_data()` function, shown below. In practice, we have found it helpful as a starting for a fuller description of the data and often add columns to further categorize individual variables for analyses.

```
library(forceps)

data(lc_adverse_events)
data(lc_biomarkers)
data(lc_demography)
data(lc_adsl)

consolidated_describe_data(lc_adverse_events,
                           lc_biomarkers,
                           lc_demography,
                           lc_adsl)
```

```
#> # A tibble: 27 x 5
#>    var_name      type     label                     format_sas data_source
#>    <chr>         <chr>    <chr>                     <chr>      <chr>
#>  1 usubjid       double   Randomization Code        <NA>       lc_adverse_even~
#>  2 ae            charact~ AE Preferred Term         character  lc_adverse_even~
#>  3 ae_type       charact~ System Organ Class 1      character  lc_adverse_even~
#>  4 grade         integer  Adverse Event Grade?      numeric    lc_adverse_even~
#>  5 ae_day        double   Days From First Dose (num~ <NA>       lc_adverse_even~
#>  6 ae_duration   double   Adverse Event Duration    numeric    lc_adverse_even~
#>  7 ae_treat      logical  Was the Adverse Event Tre~ logical    lc_adverse_even~
#>  8 ae_count      integer  Total Patient Adverse Eve~ integer    lc_adverse_even~
#>  9 usubjid       double   Randomization Code        <NA>       lc_biomarkers
#> 10 egfr_mutation charact~ EGFR Mutation +ve/-ve Res~ character  lc_biomarkers
#> # ... with 17 more rows
```

Cohorting

The data dictionary (or data description) provides a summary of the variable types and information held by variables in each of the data sets. Some data sets will include repeated, longitudinal, or time series information about individual patients, like `lc_adverese_events` in our example. Consolidating data sets like these into a single, patient-centric data set generally involves three distinct operations. The first can be thought of as `pivot_wider()` operations that take columns composed of multiple variables and spread them across new columns in the data set. The second takes the data set and `nest()`'s the data so that the the resulting data set contains time-varying data embedded in a `data.frame` variable and those variables that are repeated appear once per patient in the new variables. This verb, which is referred to as `cohort()` in the package, takes the variable to cohort on (the `usubjid` in the example below), checks for values that are repeated by the subject identifier (`ae_count` in the example below) and those that are not, and handles the nesting appropriately. A final operation may be applied to the patient-level embedded `data.frame` objects to extract other features that will be used in subsequent analyses.

```
library(dplyr)

data(lc_adverse_events)

lc_adverse_events %>% head()
```

```
#> # A tibble: 6 x 8
#>   usubjid ae        ae_type           grade ae_day ae_duration ae_treat ae_count
#>     <dbl> <chr>     <chr>             <int>  <dbl>       <dbl> <lgl>       <int>
#> 1    1003 BURNING ~ NERVOUS SYSTEM D~     1     27           4 FALSE          15
#> 2    1003 CONSTIPA~ GASTROINTESTINAL~     2      4           4 TRUE           15
#> 3    1003 DEPRESSI~ PSYCHIATRIC DISO~     2     66          NA FALSE          15
#> 4    1003 BACK PAIN MUSCULOSKELETAL ~     2     27          NA TRUE           15
#> 5    1003 DYSURIA   RENAL AND URINAR~     2      1           3 TRUE           15
#> 6    1003 SKIN EXF~ SKIN AND SUBCUTA~     1      5          26 FALSE          15
```

```
lc_adverse_events <- lc_adverse_events %>%
  cohort(on = "usubjid", name = "ae_long")

lc_adverse_events %>% head()
```

```
#> # A tibble: 6 x 3
#>   usubjid ae_count ae_long
#>     <dbl>    <int> <list>
#> 1    1003       15 <tibble [15 x 6]>
#> 2    1005       19 <tibble [19 x 6]>
#> 3    1006       11 <tibble [11 x 6]>
#> 4    1009       12 <tibble [12 x 6]>
#> 5    1014        5 <tibble [5 x 6]>
#> 6    1018       10 <tibble [10 x 6]>
```

Identifying conflicts and redundancies

After cohorting, each of the data sets is in the specified format, and we are almost ready to combine them. It is important to first check to see if there are variables that are repeated across the individual data sets and detect conflicts. While ADaM data sets *should* be free of conflicts and redundancies, we have observed multiple cases where this is not true. In order to identify these issues, the `duplicate_vars()` function is provided. The function checks the column names of each of the data sets with those of other column names. The object returned is a named list where the name corresponds to the variable that is repeated. Each list element returns a `tibble`, joined by the `on` parameter with columns corresponding to the `on` variable, the duplicated variable, the data sets where the duplicated variable appears. The example below shows that the `chemo_stop` variable appears in the `demography` and `adsl` data sets. Furthermore, we can see that the values in each of the data sets are different by looking at the correspondence between the `demography` and `adsl` columns. To fix this and move on, we will remove the variable from the `demography` data set.

```
data(lc_adsl)
data(lc_biomarkers)
data(lc_demography)
data_list <- list(demography = lc_demography,
                  biomarkers = lc_biomarkers,
                  adverse_events = lc_adverse_events,
                  adsl = lc_adsl)
duplicated_vars(data_list, on = "usubjid")
```

```
#> $chemo_stop
#> # A tibble: 558 x 4
#>    usubjid var        demography            adsl
#>      <dbl> <chr>      <chr>                 <chr>
#>  1    1003 chemo_stop patient discontinued  adverse events
#>  2    1005 chemo_stop treatment ineffective adverse events
#>  3    1006 chemo_stop <NA>                  treatment ineffective
#>  4    1009 chemo_stop treatment ineffective <NA>
#>  5    1014 chemo_stop <NA>                  adverse events
#>  6    1018 chemo_stop treatment ineffective treatment ineffective
#>  7    1023 chemo_stop <NA>                  adverse events
#>  8    1025 chemo_stop adverse events        adverse events
#>  9    1030 chemo_stop adverse events        adverse events
#> 10    1033 chemo_stop adverse events        treatment ineffective
#> # ... with 548 more rows
```

```
data_list$demography <- data_list$demography %>%
  select(-chemo_stop)
```

Consolidating

The last step is to consolidate the data sets into a single one. This is accomplished by reducing the `data_list` using full joins, along with some extra checking. The `consolidate()` function wraps this functionality. The result, conforming to the provided format, which can easily be used in the exploration and analysis stage.

```
consolidate(data_list, on = "usubjid")
```

```
#> # A tibble: 558 x 18
#>    usubjid site_id sex   refractory   age egfr_mutation smoking ecog  prior_resp
#>      <dbl>   <int> <chr> <lgl>      <dbl> <chr>         <chr>   <chr> <chr>
#>  1    1003       1 male  FALSE         51 negative      former~ ambu~ complete ~
#>  2    1005       4 fema~ TRUE          44 negative      former~ ambu~ partial r~
#>  3    1006       2 male  TRUE          22 negative      former~ ambu~ complete ~
#>  4    1009       8 male  FALSE         44 <NA>          unknown ambu~ complete ~
#>  5    1014       6 male  TRUE          76 <NA>          former~ ambu~ partial r~
#>  6    1018      10 fema~ TRUE          35 positive      former~ ambu~ complete ~
#>  7    1023       6 fema~ TRUE          73 <NA>          former~ ambu~ complete ~
#>  8    1025       7 male  FALSE         71 <NA>          never ~ ambu~ partial r~
#>  9    1030       5 fema~ TRUE          20 <NA>          unknown ambu~ partial r~
#> 10    1033       6 fema~ TRUE          55 <NA>          unknown ambu~ stable di~
#> # ... with 548 more rows, and 9 more variables: ae_count <int>, ae_long <list>,
#> #   best_response <chr>, pfs_days <dbl>, pfs_censor <dbl>, os_days <dbl>,
#> #   os_censor <dbl>, chemo_stop <chr>, arm <chr>
```

Direction: An integrated approach to processing clinical data

As stated before, the goal of this paper is to help define a path toward a grammar for processing clinical trials by a) defining a format in which we would like to represent data from standardized clinical trial data b) describing a standard set of operations to transform clinical trial data into this format, and c) to identify a set of verbs and other functionality to facilitate data processing of this kind and encourage reproducibility of these steps. Admittedly, this only serves to mitigate the process of preparing these types of data for exploration and analysis. Clinical trial data generally contains many more variables than what was presented, and each of these data sets comes with its own set of "quirks" and other challenges. However, it does serve to make the data preparation better defined and propose a path toward standardization of both the processed data set format as well as the operations to achieve that goal.

Along with further development towards those ends, there is a plethora of development that can be done to provide an integrated data processing experience. For example, the define.xml file, which appears alongside ADaM data sets, gives better descriptions of the variables as well as the variable values. Tools to integrate these data into the construction of the data dictionary would go a long way towards orienting researchers with the data contained and help them more quickly formulate analyses. Packages like **lumberjack** (van der Loo, 2020) could enhance and augment data preprocessing steps by keeping better track of when data are being removed and how they are being manipulated. The artifacts accumulated could then be used by packages such as **ggconsort** (Higgins, 2020) to provide consort diagrams of how patients progress through the trial and how data progresses through preprocessing. In the longer term, these advancements can provide better data provenance, more reproducible processing, quicker debugging of problems in the processing stage, and give rise to more effective and convenient tools for summarizing trial data.

Bibliography

Clinical data interchange standards consortium. https://www.cdisc.org/, 2020. Accessed: 2020-10-25. [p563]

Immport: Bioinformatics for the future of immunology. https://www.immport.org/home, 2020. Accessed: 2020-10-25. [p563]

Project data sphere: Convener, collaborator, catalyst in the fight against cancer. https://www.projectdatasphere.org/, 2020. Accessed: 2020-10-25. [p563]

P. Higgins. *ggconsort: Creates CONSORT Diagrams for RCTs*, 2020. URL https://github.com/higgi13425/ggconsort. R package version 0.0.0.9000. [p568]

M. J. Kane. *forceps: A grammar for manipulating clinical trial data*, 2020. URL https://github.com/kaneplusplus/forceps. R package version 0.0.1. [p564]

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2012. URL http://www.R-project.org/. ISBN 3-900051-07-0. [p564]

SAS Institute. *Base SAS 9.4 procedures guide*. SAS Institute, 2020. [p564]

M. Shotwell. *sas7bdat: SAS Database Reader (experimental)*, 2014. URL https://CRAN.R-project.org/package=sas7bdat. R package version 0.5. [p564]

M. S. Shotwell, C. Cummins, S. Header, S. Pages, S. Subheaders, and S. P. B. Data. Sas7bdat database binary format, 2013. [p564]

M. van der Loo. Monitoring data in r with the lumberjack package. *Journal of Statistical Software*, page Accepted for publication, 2020. URL https://CRAN.R-project.org/package=lumberjack. [p568]

H. Wickham and E. Miller. *haven: Import and Export 'SPSS', 'Stata' and 'SAS' Files*, 2020. URL https://CRAN.R-project.org/package=haven. R package version 2.3.1. [p564]

H. Wickham et al. Tidy data. *Journal of Statistical Software*, 59(10):1–23, 2014. [p564]

Michael J. Kane
Yale University
60 College Street
New Haven, CT 06510, USA

michael.kane@yale.edu

Reproducible Summary Tables with the gtsummary Package

by Daniel D. Sjoberg, Karissa Whiting, Michael Curry, Jessica A. Lavery, Joseph Larmarange

Abstract The **gtsummary** package provides an elegant and flexible way to create publication-ready summary tables in R. A critical part of the work of statisticians, data scientists, and analysts is summarizing data sets and regression models in R and publishing or sharing polished summary tables. The **gtsummary** package was created to streamline these everyday analysis tasks by allowing users to easily create reproducible summaries of data sets, regression models, survey data, and survival data with a simple interface and very little code. The package follows a tidy framework, making it easy to integrate with standard data workflows, and offers many table customization features through function arguments, helper functions, and custom themes.

Introduction

Table summaries are a fundamental tool in an analyst's toolbox that help us understand and communicate patterns in our data. The ability to easily create and export polished and reproducible tables is essential. The **gtsummary** (Sjoberg et al., 2020) package provides an elegant and flexible framework to create publication-ready analytical and summary tables in R. This package works to close the gap between a reproducible RMarkdown report and the final report. Specifically, **gtsummary** allows the user to fully customize and format summary tables with code, eliminating the need to modify any tables by hand after the table has been exported. Removing the need to modify tables after the table has been created eliminates an error-prone step in our workflow and increases the reproducibility of our analyses and reports.

Using **gtsummary**, analysts can easily summarize data frames, present and compare descriptive statistics between groups, summarize regression models, and report statistics inline in RMarkdown reports. After identifying these basic structures of most tables presented in the medical literature (and other fields), we wrote **gtsummary** to ease the creation of fully-formatted, ready-to-publish tables.

Additionally, **gtsummary** leverages other analysis and tidying R packages to create a complete analysis and reporting framework. For example, we take advantage of the existing **broom** (Robinson et al., 2020) tidiers to prepare regression results for `tbl_regression()` and use **gt** (Iannone et al., 2020) to print **gtsummary** tables to various output formats (e.g., HTML, PDF, Word, or RTF). Furthermore, **gtsummary** functions are designed to work within a "tidy" framework, utilizing the **magrittr** (Bache and Wickham, 2020) pipe operator and **tidyselect** (Henry and Wickham, 2020) functions used throughout the **tidyverse** (Wickham et al., 2019).

While other R packages are available to present data and regression model summary tables, such as **skimr**, **stargazer**, **finalfit**, and **tableone**, **gtsummary** is unique in that it is a one-stop-shop for most types of statistical tables and offers diverse features to customize the content of tables to a high degree. The default **gtsummary** table is suitable to be published in a scientific journal with little or no additional formatting. For example, **gtsummary** has specific internal algorithms to identify variable data types, so there is no need for users to specify whether a variable should be displayed with categorical or continuous summaries, which yields summary tables with minimal code.

Along with descriptive summaries, **gtsummary** summarizes statistical models, survey data, survival data and builds cross-tabulations. After data are summarized in a table, **gtsummary** allows users to combine tables, either side-by-side (with `tbl_merge()`) , or on top of each other (with `tbl_stack()`). The table merging and stacking abilities allows analysts to easily synthesize and compare output from several tables and share information in a compact format. All tables in this manuscript were created using **gtsummary** v1.4.1.

Data Summaries

To showcase **gtsummary** functions, we will use a simulated clinical trial data set containing baseline characteristics of 200 patients who received Drug A or Drug B, as well as the outcomes of tumor response and death. Each variable in the data frame has been assigned an attribute label with the **labelled** package (Larmarange, 2020), e.g., `trial %>% set_variable_labels(age = "Age")`, that will be shown in the summary tables. These labels are displayed in the **gtsummary** tables by default, and had labels not been assigned, the variable name would have been shown.

colname	label	class	values
trt	Chemotherapy Treatment	character	Drug A, Drug B
age	Age	numeric	6, 9, 10, 17, ...
marker	Marker Level (ng/mL)	numeric	0.003, 0.005, 0.013, 0.015, ...
stage	T Stage	factor	T1, T2, T3, T4
grade	Grade	factor	I, II, III
response	Tumor Response	integer	0, 1
death	Patient Died	integer	0, 1
ttdeath	Months to Death/Censor	numeric	3.53, 5.33, 6.32, 7.27, ...

Table 1. Example data frame, `trial`

`tbl_summary()`

The default output from `tbl_summary()` is meant to be publication-ready. The `tbl_summary()` function can take, at minimum, a data frame as the only input, and returns descriptive statistics for each column in the data frame. This is often the first table of clinical manuscripts and describes the characteristics of the study cohort. A simple example is shown below. Notably, by specifying the by= argument, you can stratify the summary table. In the example below, we have split the table by the treatment a patient received.

```
trial %>%
  select(age, grade, response, trt) %>%
  tbl_summary(by = trt)
```

Characteristic	**Drug A**, N = 98[1]	**Drug B**, N = 102[1]
Age	46 (37, 59)	48 (39, 56)
Unknown	7	4
Grade		
I	35 (36%)	33 (32%)
II	32 (33%)	36 (35%)
III	31 (32%)	33 (32%)
Tumor Response	28 (29%)	33 (34%)
Unknown	3	4

[1] Median (IQR); n (%)

The function is highly customizable, and it is initiated with sensible default settings. Specifically, `tbl_summary()` detects variable types of input data and calculates descriptive statistics accordingly. For example, variables coded as `0/1`, `TRUE/FALSE`, and `Yes/No` are presented dichotomously. Additionally, `NA` values are recognized as missing and listed as unknown, and if a data set is labeled, the label attributes are utilized.

Default settings may be customized using the `tbl_summary()` function arguments.

Argument	Description
label=	specify the variable labels printed in table
type=	specify the variable type (e.g., continuous, categorical, etc.)
statistic=	change the summary statistics presented
digits=	number of digits the summary statistics will be rounded to
missing=	whether to display a row with the number of missing observations
missing_text=	text label for the missing number row
sort=	change the sorting of categorical levels by frequency
percent=	print column, row, or cell percentages
include=	list of variables to include in summary table

Table 2. `tbl_summary()` function arguments

For continuous variables, tables display one row of statistics per variable by default. This can be customized, and in the example below, the age variable is cast to `"continuous2"` type, meaning the

continuous summary statistics will appear on two or more rows in the table. This allows the number of non-missing observations and the mean to be displayed on separate lines.

In the example below, the "age" variable's label is updated to "Patient Age". Default summary statistics for both continuous and categorical variables are updated using the statistic= argument. **gtsummary** uses **glue** (Hester, 2020) syntax to construct the statistics displayed in the table. Function names appearing in curly brackets will be replaced by the evaluated value. The digits= argument is used to increase the number of decimal places to which the statistics are rounded, and the missing row is omitted with missing = "no".

```
trial %>%
  select(age, grade, response, trt) %>%
  tbl_summary(
    by = trt,
    type = age ~ "continuous2",
    label = age ~ "Patient Age",
    statistic = list(age ~ c("{N_nonmiss}", "{mean} ({sd})"),
                     c(grade, response) ~ "{n} / {N} ({p}%)"),
    digits = c(grade, response) ~ c(0, 0, 1),
    missing = "no"
  )
```

Characteristic	**Drug A**, N = 98	**Drug B**, N = 102
Patient Age		
N	91	98
Mean (SD)	47 (15)	47 (14)
Grade		
I	35 / 98 (35.7%)	33 / 102 (32.4%)
II	32 / 98 (32.7%)	36 / 102 (35.3%)
III	31 / 98 (31.6%)	33 / 102 (32.4%)
Tumor Response	28 / 95 (29.5%)	33 / 98 (33.7%)

A note about notation: Throughout the **gtsummary** package, you will find function arguments that accept a list of formulas (or a single formula) as the input. In the example above, the label for the age variable was updated using label = age ~ "Patient Age"—equivalently, label = list(age ~ "Patient Age"). To select groups of variables, utilize the select helpers from the **tidyselect** and **gtsummary** packages. The all_continuous() selector is a convenient way to select all continuous variables. In the example above, it could have been used to change the summary statistics for all continuous variables—all_continuous() ~ c("{N_nonmiss}","{mean} ({sd})"). Similarly, users may utilize all_categorical() (from **gtsummary**) or any of the **tidyselect** helpers used throughout the **tidyverse** packages, such as starts_with(), contains(), etc.

In addition to summary statistics, the **gtsummary** package has several functions to add additional information or statistics to tbl_summary() tables.

Function	Description
add_p()	add *p*-values to the output comparing values across groups
add_overall()	add a column with overall summary statistics
add_n()	add a column with N (or N missing) for each variable
add_difference()	add column for difference between two group, confidence interval, and *p*-value
add_stat_label()	add label for the summary statistics shown in each row
add_stat()	generic function to add a column with user-defined values
add_q()	add a column of *q*-values to control for multiple comparisons

Table 3. tbl_summary() functions to add information

In the example below, descriptive statistics are shown by the treatment received and overall, as well as a *p*-value comparing the values between the treatments. Default statistical tests are chosen based on data type, and the statistical test performed can be customized in the add_p() function. *p*-value formatting can be adjusted using the pvalue_fun= argument, which accepts both a proper function, as well the formula shortcut notation used throughout the **tidyverse** packages.

```
trial %>%
  select(age, grade, response, trt) %>%
  tbl_summary(by = trt) %>%
  add_overall() %>%
  add_p(test = all_continuous() ~ "t.test",
        pvalue_fun = ~style_pvalue(., digits = 2))
```

Characteristic	**Overall**, N = 200[1]	**Drug A**, N = 98[1]	**Drug B**, N = 102[1]	**p-value**[2]
Age	47 (38, 57)	46 (37, 59)	48 (39, 56)	0.83
Unknown	11	7	4	
Grade				0.87
I	68 (34%)	35 (36%)	33 (32%)	
II	68 (34%)	32 (33%)	36 (35%)	
III	64 (32%)	31 (32%)	33 (32%)	
Tumor Response	61 (32%)	28 (29%)	33 (34%)	0.53
Unknown	7	3	4	

[1] Median (IQR); n (%)
[2] Welch Two Sample t-test; Pearson's Chi-squared test

`tbl_svysummary()`

The `tbl_svysummary()` function is analogous to `tbl_summary()`, except a **survey** (Lumley, 2020) object is supplied rather than a data frame. The summary statistics presented take into account the survey weights, as do any *p*-values presented.

```
# convert trial data frame to survey object
svy_trial <- survey::svydesign(data = trial, ids = ~ 1, weights = ~ 1)

tbl_svysummary_1 <-
  svy_trial %>%
  tbl_svysummary(by = trt, include = c(trt, age, grade)) %>%
  add_p()
```

Characteristic	**Drug A**, N = 98[1]	**Drug B**, N = 102[1]	**p-value**[2]
Age	46 (37, 58)	48 (38, 56)	0.7
Unknown	7	4	
Grade			0.9
I	35 (36%)	33 (32%)	
II	32 (33%)	36 (35%)	
III	31 (32%)	33 (32%)	

[1] Median (IQR); n (%)
[2] Wilcoxon rank-sum test for complex survey samples; chi-squared test with Rao & Scott's second-order correction

`tbl_cross()`

The `tbl_cross()` function is a wrapper for `tbl_summary()` and creates a simple, publication-ready cross tabulation.

```
trial %>%
  tbl_cross(row = stage, col = trt, percent = "cell") %>%
  add_p(source_note = TRUE)
```

Characteristic	Chemotherapy Treatment		Total
	Drug A	Drug B	
T Stage			
T1	28 (14%)	25 (12%)	53 (26%)
T2	25 (12%)	29 (14%)	54 (27%)
T3	22 (11%)	21 (10%)	43 (22%)
T4	23 (12%)	27 (14%)	50 (25%)
Total	98 (49%)	102 (51%)	200 (100%)
Pearson's Chi-squared test, p=0.9			

`tbl_survfit()`

The `tbl_survfit()` function parses and tabulates `survival::survfit()` objects presenting survival percentile estimates and survival probabilities at specified times.

```
library(survival)

list(survfit(Surv(ttdeath, death) ~ trt, trial),
     survfit(Surv(ttdeath, death) ~ grade, trial)) %>%
  tbl_survfit(times = c(12, 24),
              label_header = "**{time} Month**") %>%
  add_p()
```

Characteristic	12 Month	24 Month	p-value[1]
Chemotherapy Treatment			0.2
Drug A	91% (85%, 97%)	47% (38%, 58%)	
Drug B	86% (80%, 93%)	41% (33%, 52%)	
Grade			0.072
I	97% (93%, 100%)	51% (41%, 65%)	
II	82% (74%, 92%)	47% (37%, 61%)	
III	86% (78%, 95%)	33% (23%, 47%)	
[1] Log-rank test			

Customization

The **gtsummary** package includes functions specifically made to modify and format the summary tables. These functions work with any table constructed with **gtsummary**. The most common uses are changing the column headers and footnotes or modifying the look of tables through bolding and italicization.

Function	Description
`modify_header()`	update column headers
`modify_footnote()`	update column footnote
`modify_spanning_header()`	update spanning headers
`modify_caption()`	update table caption/title
`bold_labels()`	bold variable labels
`bold_levels()`	bold variable levels
`italicize_labels()`	italicize variable labels
`italicize_levels()`	italicize variable levels
`bold_p()`	bold significant *p*-values

Table 4. Functions to style and modify gtsummary tables

The **gtsummary** package utilizes the **gt** package to print the summary tables. The **gt** package exports approximately one hundred functions to customize and style tables. When you need to add additional details or styling not available within **gtsummary**, use the `as_gt()` function to convert the **gtsummary** object to **gt** and continue customization.

The example below is a common table reported in clinical trials and observational research where two treatments are compared. The treatment differences were added with the `add_difference()` function. The table includes customization using both **gtsummary** and **gt** functions. The **gtsummary** functions are utilized to bold the variable labels, update the column headers, and add a spanning header. Additional **gt** customization was utilized to add table captions and source notes.

```
trial %>%
  select(marker, response, trt) %>%
  tbl_summary(by = trt,
              missing = "no",
              statistic = marker ~ "{mean} ({sd})") %>%
  add_difference() %>%
  add_n() %>%
  add_stat_label() %>%
  bold_labels() %>%
  modify_header(list(label ~ "**Variable**", all_stat_cols() ~ "**{level}**")) %>%
  modify_spanning_header(all_stat_cols() ~ "**Randomization Assignment**") %>%
  as_gt() %>%
  gt::tab_header(
    title = gt::md("**Table 1. Treatment Differences**"),
    subtitle = gt::md("_Highly Confidential_")
  ) %>%
  gt::tab_source_note("Data updated June 26, 2015")
```

Table 1. Treatment Differences

Highly Confidential

		Randomization Assignment				
Variable	**N**	**Drug A**	**Drug B**	**Difference[1]**	**95% CI[1,2]**	**p-value[1]**
Marker Level (ng/mL), Mean (SD)	190	1.02 (0.89)	0.82 (0.83)	0.20	-0.05, 0.44	0.12
Tumor Response, n (%)	193	28 (29%)	33 (34%)	-4.2%	-18%, 9.9%	0.6

[1] Welch Two Sample t-test; Two sample test for equality of proportions

[2] CI = Confidence Interval

Data updated June 26, 2015

Model Summaries

Regression modeling is one of the most common tools of medical research. The **gtsummary** package has two functions to help analysts prepare tabular summaries of regression models: `tbl_regression()` and `tbl_uvregression()`.

tbl_regression()

The `tbl_regression()` function takes a regression model object in R and returns a formatted table of regression model results. Like `tbl_summary()`, `tbl_regression()` creates highly customizable analytic tables with sensible defaults. Common regression models, such as logistic regression and Cox proportional hazards regression, are automatically identified, and the tables headers are pre-filled with appropriate column headers (i.e., Odds Ratio and Hazard Ratio).

In the example below, the logistic regression model is summarized with `tbl_regression()`. Note that a reference row for grade has been added, and the variable labels have been carried through into the table. Using `exponentiate = TRUE`, we exponentiate the regression coefficients, yielding the odds ratios. The helper function `add_global_p()` was used to replace the *p*-values for each term with the global *p*-value for grade.

```
glm(response ~ age + grade, trial, family = binomial) %>%
  tbl_regression(exponentiate = TRUE) %>%
  add_global_p()
```

Characteristic	OR[1]	95% CI[1]	p-value
Age	1.02	1.00, 1.04	0.092
Grade			0.9
I	—	—	
II	0.85	0.39, 1.85	
III	1.01	0.47, 2.16	

[1] OR = Odds Ratio, CI = Confidence Interval

The `tbl_regression()` function leverages the huge effort behind the **broom**, **parameters** (Lüdecke et al., 2020), and **broom.helpers** (Larmarange and Sjoberg, 2021) packages to perform the initial formatting of the regression object. Because `tbl_regression()` utilizes these packages, there are many model types that are supported out of the box, such as `lm()`, `glm()`, `lme4::lmer()`, `lme4::glmer()`, `geepack::geeglm()`, `survival::coxph()`, `survival::survreg()`, `survival::clogit()`, `nnet::multinom()`, `rstanarm::stan_glm()`, models built with the **mice** package (van Buuren and Groothuis-Oudshoorn, 2011), and many more. A custom tidier may be specified as well, which is helpful when you need to present non-standard modifications to your model results such as Wald confidence intervals or results with modified variance-covariance standard errors.

tbl_uvregression()

The `tbl_uvregression()` function is a wrapper for `tbl_regression()` that is useful when you need a series of univariate regression models. The user passes a data frame to `tbl_uvregression()`, indicates what the outcome is, what regression model to run, and the function will return a formatted table of stacked univariate regression models.

```
trial %>%
  select(response, age, grade) %>%
  tbl_uvregression(
    y = response,
    method = glm,
    method.args = list(family = binomial),
    exponentiate = TRUE,
    pvalue_fun = ~style_pvalue(., digits = 2)
  ) %>%
  add_nevent() %>%
  add_global_p()
```

Characteristic	N	Event N	OR[1]	95% CI[1]	p-value
Age	183	58	1.02	1.00, 1.04	0.091
Grade	193	61			0.93
I			—	—	
II			0.95	0.45, 2.00	
III			1.10	0.52, 2.29	

[1] OR = Odds Ratio, CI = Confidence Interval

Inline Reporting

Reproducible reports are an important part of good analytic practices. We often need to report the results from a table in the text of an R markdown report. The `inline_text()` function reports statistics from **gtsummary** tables inline in an R markdown document.

Imagine you need to report the results for age from the univariate table above. Typically, the odds ratio, confidence interval, and *p*-value would be hard-coded into a report, which can lead to reproducibility issues if the data is updated and the hard-coded statistics are not amended. A simple call to the `inline_text()` function will dynamically add the model results to an RMarkdown report.

> The odds ratio for age was `` `r inline_text(uvreg, variable = age)` ``.

Here is how the line will appear in your report.

> The odds ratio for age was 1.02 (95% CI 1.00, 1.04; p=0.091).

The default pattern to display for a regression table is `"{estimate} ({conf.level*100}% CI {conf.low},{conf.high}; {p.value})"` (again using glue syntax), and can be modified with the `inline_text(pattern=)` argument.

Merging and Stacking

The **gtsummary** tables shown above are often ready for publication as they are; however, it is common that more complex tables need to be constructed. This can be achieved by merging or stacking **gtsummary** tables using the `tbl_merge()` and `tbl_stack()` functions. For example, in cancer research we often report models predicting a tumor's response to treatment and risk of death side-by-side in publications. This type of table is simple to construct using `tbl_merge()`. First, build a table for each regression model using `tbl_regression()`, then merge the two tables with `tbl_merge()`. Any number of **gtsummary** tables can be merged with this function.

```
tbl1 <-
  glm(response ~ age + grade, trial, family = binomial) %>%
  tbl_regression(exponentiate = TRUE)

tbl2 <-
  coxph(Surv(ttdeath, death) ~ age + grade, trial) %>%
  tbl_regression(exponentiate = TRUE)

tbl_merge_1 <-
  tbl_merge(
    tbls = list(tbl1, tbl2),
    tab_spanner = c("**Tumor Response**", "**Time to Death**")
  )
```

	Tumor Response			Time to Death		
Characteristic	**OR[1]**	**95% CI[1]**	**p-value**	**HR[1]**	**95% CI[1]**	**p-value**
Age	1.02	1.00, 1.04	0.10	1.01	0.99, 1.02	0.3
Grade						
I	—	—		—	—	
II	0.85	0.39, 1.85	0.7	1.20	0.73, 1.97	0.5
III	1.01	0.47, 2.16	>0.9	1.80	1.13, 2.87	0.014

[1] OR = Odds Ratio, CI = Confidence Interval, HR = Hazard Ratio

Similarly, any number of **gtsummary** tables may be stacked using the `tbl_stack()` function.

Themes

We love themes. The default styling (e.g., statistics displayed in `tbl_summary()`, how *p*-values are rounded, decimal separator, and more) follow the reporting guidelines from European Urology, The Journal of Urology, Urology, and the British Journal of Urology International (Assel et al., 2019). However, you will likely submit to another journal, or your personal preferences differ from the defaults. The **gtsummary** package is unique from other table building packages with the ability to set fine-grained customization defaults with themes. Themes were created to make these customizations easy to navigate and reuse across documents or projects. With themes, users can control default settings for existing functions (e.g., always present means instead of medians in `tbl_summary()`), as well as other changes that are not modifiable with function arguments. Several themes are available to follow various journals' reporting guidelines, reduce cell padding and font size, and language themes to translate **gtsummary** tables to more than 14 languages.

For example, using the theme for *The Journal of the American Medical Association* (*JAMA*), large *p*-values are rounded to two decimal places, confidence intervals are shown as `"lb to ub"` instead of `"lb,ub"`, and the confidence interval is displayed in the same column as the model coefficients.

```
theme_gtsummary_journal("jama")

glm(response ~ age + grade, trial, family = binomial) %>%
  tbl_regression(exponentiate = TRUE)
```

Characteristic	OR (95% CI)[1]	p-value
Age	1.02 (1.00 to 1.04)	0.10
Grade		
I	—	
II	0.85 (0.39 to 1.85)	0.69
III	1.01 (0.47 to 2.16)	0.97

[1] OR = Odds Ratio, CI = Confidence Interval

The language theme can be used to translate the table to another language and allows users to specify the decimal and big mark symbols. For example, `theme_gtsummary_language(language = "es",decimal.mark = ",",big.mark = ".")` will translate the output to Spanish and format numeric results as `1.000,00` instead of `1,000.00` (the default formatting).

A custom theme was used to construct the **gtsummary** tables shown in this manuscript to match the *R Journal* font and reduce the default cell padding. Themes are an evolving feature, and we welcome additions of new journals' reporting guidelines or other themes useful to users. A full glossary of customizable theme elements is available in the package's themes vignette (http://www.danieldsjoberg.com/gtsummary/articles/themes.html).

Print Engines

Tables printed with **gtsummary** can be seamlessly integrated into RMarkdown documents and knitted into various output types using a number of print engines. The package was written to be a companion to the **gt** package from RStudio and is optimized to leverage the advanced customization features of this print engine, but offers compatibility with a variety of popular printing methods, including `knitr::kable()` (Xie, 2020), **flextable** (Gohel, 2020), **huxtable** (Hugh-Jones, 2020), and **kableExtra** (Zhu, 2020). While **gt** is used as the default for most outputs, you can easily use your print engine of choice with the conversion helper functions provided in the package (e.g., `as_flex_table()`). It is possible to get results in HTML, PDF (via LaTeX), RTF, Microsoft Word, PowerPoint, Excel, and others, utilizing the various print engines. The package is designed to interact with these print engines behind the scenes to reduce the burden on users, and you generally only need to be aware of them if you want to add advanced customizations.

Summary

The functions in the **gtsummary** package were designed to reduce the burden of reporting and to work together to easily construct both simple and complex tables. It is our hope that the user-friendly syntax and publication-ready tables will aid analysts in preparing reproducible and high-quality findings.

Bibliography

M. Assel, D. Sjoberg, A. Elders, X. Wang, D. Huo, A. Botchway, K. Delfino, Y. Fan, Z. Zhao, T. Koyama, et al. Guidelines for reporting of statistics for clinical research in urology. *European urology*, 75(3): 358, 2019. [p578]

S. M. Bache and H. Wickham. *magrittr: A Forward-Pipe Operator for R*, 2020. URL https://CRAN.R-project.org/package=magrittr. R package version 2.0.1. [p570]

D. Gohel. *flextable: Functions for Tabular Reporting*, 2020. URL https://CRAN.R-project.org/package=flextable. R package version 0.6.1. [p579]

L. Henry and H. Wickham. *tidyselect: Select from a Set of Strings*, 2020. URL https://CRAN.R-project.org/package=tidyselect. R package version 1.1.0. [p570]

J. Hester. *glue: Interpreted String Literals*, 2020. URL https://CRAN.R-project.org/package=glue. R package version 1.4.2. [p572]

D. Hugh-Jones. *huxtable: Easily Create and Style Tables for LaTeX, HTML and Other Formats*, 2020. URL https://CRAN.R-project.org/package=huxtable. R package version 5.1.1. [p579]

R. Iannone, J. Cheng, and B. Schloerke. *gt: Easily Create Presentation-Ready Display Tables*, 2020. https://gt.rstudio.com/, https://github.com/rstudio/gt. [p570]

J. Larmarange. *labelled: Manipulating Labelled Data*, 2020. URL https://CRAN.R-project.org/package=labelled. R package version 2.7.0. [p570]

J. Larmarange and D. D. Sjoberg. *broom.helpers: Helpers for Model Coefficients Tibbles*, 2021. URL https://CRAN.R-project.org/package=broom.helpers. R package version 1.3.0. [p576]

T. Lumley. *survey: Analysis of Complex Survey Samples*, 2020. URL https://CRAN.R-project.org/package=survey. R package version 4.0. [p573]

D. Lüdecke, M. S. Ben-Shachar, I. Patil, and D. Makowski. Extracting, computing and exploring the parameters of statistical models using R. *Journal of Open Source Software*, 5(53):2445, 2020. doi: 10.21105/joss.02445. [p576]

D. Robinson, A. Hayes, and S. Couch. *broom: Convert Statistical Objects into Tidy Tibbles*, 2020. URL https://CRAN.R-project.org/package=broom. R package version 0.7.2. [p570]

D. D. Sjoberg, M. Curry, M. Hannum, K. Whiting, and E. C. Zabor. *gtsummary: Presentation-Ready Data Summary and Analytic Result Tables*, 2020. URL https://CRAN.R-project.org/package=gtsummary. R package version 1.3.6, https://github.com/ddsjoberg/gtsummary, http://www.danieldsjoberg.com/gtsummary/. [p570]

S. van Buuren and K. Groothuis-Oudshoorn. mice: Multivariate imputation by chained equations in r. *Journal of Statistical Software*, 45(3):1–67, 2011. URL https://www.jstatsoft.org/v45/i03/. [p576]

H. Wickham, M. Averick, J. Bryan, W. Chang, L. D. McGowan, R. François, G. Grolemund, A. Hayes, L. Henry, J. Hester, M. Kuhn, T. L. Pedersen, E. Miller, S. M. Bache, K. Müller, J. Ooms, D. Robinson, D. P. Seidel, V. Spinu, K. Takahashi, D. Vaughan, C. Wilke, K. Woo, and H. Yutani. Welcome to the tidyverse. *Journal of Open Source Software*, 4(43):1686, 2019. doi: 10.21105/joss.01686. [p570]

Y. Xie. *knitr: A General-Purpose Package for Dynamic Report Generation in R*, 2020. URL https://CRAN.R-project.org/package=knitr. R package version 1.30. [p579]

H. Zhu. *kableExtra: Construct Complex Table with 'kable' and Pipe Syntax*, 2020. URL https://CRAN.R-project.org/package=kableExtra. R package version 1.3.1. [p579]

Daniel D. Sjoberg
Memorial Sloan Kettering Cancer Center
1275 York Ave., New York, New York 10022
USA
ORCiD: 0000-0003-0862-2018
sjobergd@mskcc.org

Karissa Whiting
Memorial Sloan Kettering Cancer Center
1275 York Ave., New York, New York 10022
USA
ORCiD: 0000-0002-4683-1868
whitingk@mskcc.org

Michael Curry
Memorial Sloan Kettering Cancer Center
1275 York Ave., New York, New York 10022
USA
ORCiD: 0000-0002-0261-4044
currym1@mskcc.org

Jessica A. Lavery
Memorial Sloan Kettering Cancer Center
1275 York Ave., New York, New York 10022
USA
ORCiD: 0000-0002-2746-5647
laveryj@mskcc.org

Joseph Larmarange
Centre Population & Développement, IRD, Université de Paris, Inserm
45 rue des Saints-Pères 75006 PARIS
France
ORCID: 0000-0001-7097-700X
joseph.larmarange@ceped.org

Regularized Transformation Models: The tramnet Package

by Lucas Kook and Torsten Hothorn

Abstract The **tramnet** package implements regularized linear transformation models by combining the flexible class of transformation models from **tram** with constrained convex optimization implemented in **CVXR**. Regularized transformation models unify many existing and novel regularized regression models under one theoretical and computational framework. Regularization strategies implemented for transformation models in **tramnet** include the Lasso, ridge regression, and the elastic net and follow the parameterization in **glmnet**. Several functionalities for optimizing the hyperparameters, including model-based optimization based on the **mlrMBO** package, are implemented. A multitude of S3 methods is deployed for visualization, handling, and simulation purposes. This work aims at illustrating all facets of **tramnet** in realistic settings and comparing regularized transformation models with existing implementations of similar models.

Introduction

A plethora of R packages exists to estimate generalized linear regression models via penalized maximum likelihood, such as **penalized** (Goeman, 2010) and **glmnet** (Friedman et al., 2010). Both packages come with an extension to fit a penalized form of the Cox proportional hazard model. The **tramnet** package aims at unifying the above-mentioned and several novel models using the theoretical and computational framework of transformation models. Novel models in this class include Continuous Outcome Logistic Regression (COLR), as introduced by Lohse et al. (2017) and Box-Cox type regression models with a transformed conditionally normal response (Box and Cox, 1964; Hothorn, 2020c).

The disciplined convex optimization package **CVXR** (Fu et al., 2020) is applied to solve the constrained convex optimization problems that arise when fitting regularized transformation models. Transformation models are introduced in Section Transformation models. For a more theoretical treatise, we refer to Hothorn et al. (2014, 2018); Hothorn (2020b). Convex optimization and domain-specific languages are briefly discussed in Section Constrained convex optimization, followed by a treatment of model-based optimization for hyperparameter tuning (Section Model-based optimization).

Transformation models

In stark contrast to penalized generalized linear models, regularized transformation models aim at estimating the response's whole conditional distribution instead of focusing on a single moment, *e.g.*, the conditional mean. This conditional distribution function of a response Y is decomposed into an *a priori* chosen absolute continuous and log-concave error distribution F and a conditional transformation function $h(y|\boldsymbol{x}, \boldsymbol{s})$ that depends on the measured covariates $\boldsymbol{x}$ and stratum variables $\boldsymbol{s}$ and is monotone increasing in y. Although the model class is more flexible, packages **tram** and **tramnet** focus on stratified linear transformation models of the form

$$\mathbb{P}\left(Y \leq y | \boldsymbol{X} = \boldsymbol{x}, \boldsymbol{S} = \boldsymbol{s}\right) = F\left(h(y|\boldsymbol{s}, \boldsymbol{x})\right) = F\left(h(y|\boldsymbol{s}) - \boldsymbol{x}^\top \boldsymbol{\beta}\right). \tag{1}$$

Here, the baseline transformation is allowed to vary with stratum variables $\boldsymbol{s}$, while covariate effects $\boldsymbol{\beta}$ are restricted to be shifts common to all baseline transformations $h(y|\boldsymbol{s})$.

In order for the model to represent a valid cumulative distribution function, $F\left(h(y|\boldsymbol{s}, \boldsymbol{x})\right)$ has to be monotone increasing in y, and thus, in h for all possible strata $\boldsymbol{s}$ and all possible configurations of the covariates $\boldsymbol{x}$. To ensure monotonicity, h is parameterized in terms of a basis expansion using Bernstein polynomials as implemented in the **basefun** package (Hothorn, 2020b). Hence, h is of the form

$$h(y) = \boldsymbol{a}_{\text{Bs},p}(y)^\top \boldsymbol{\vartheta},$$

where $\boldsymbol{a}_{\text{Bs},p}(y)$ denotes the vector of basis functions in y of order p and $\boldsymbol{\vartheta}$ are the coefficients for each basis function. Conveniently, $\boldsymbol{a}_{\text{Bs},p}(y)^\top \boldsymbol{\vartheta}$ is monotone increasing in y as long as

$$\vartheta_i \leq \vartheta_{i+1}, \quad i = 0, \ldots, p-1 \tag{2}$$

holds. For the concrete parameterization of stratified linear transformation models, the reader is referred to Hothorn (2020c).

Many contemporary models can be understood as linear transformation models, such as the normal linear regression model, logistic regression for binary, ordered, and continuous responses, as well as exponential, Weibull and Rayleigh regression, and the Cox model in survival analysis. Thus, by appropriately choosing and parameterizing F and h, one can understand all those models in the same maximum likelihood-based framework. One can formulate the corresponding likelihood contributions not only for exact observations but under any form of random censoring and truncation for continuous and count or ordered categorical responses.

Given a univariate response Y and a set of covariates $\boldsymbol{X} = \boldsymbol{x}$ and strata $\boldsymbol{S} = \boldsymbol{s}$, one can specify the following cumulative distribution function and density valid for any linear transformation model:

$$
\begin{aligned}
F_{Y|\boldsymbol{X}=\boldsymbol{x}}(y|\boldsymbol{s},\boldsymbol{x}) &= F\left(h(y \mid \boldsymbol{s}) - \boldsymbol{x}^\top \boldsymbol{\beta}\right), \\
f_{Y|\boldsymbol{X}=\boldsymbol{x}}(y|\boldsymbol{s},\boldsymbol{x}) &= F'\left(h(y \mid \boldsymbol{s}) - \boldsymbol{x}^\top \boldsymbol{\beta}\right) \cdot h'(y \mid \boldsymbol{s}).
\end{aligned}
$$

From here, the log-likelihood contributions for exact, right, left, and interval-censored responses can be derived as

$$
\ell(\boldsymbol{\vartheta}, \boldsymbol{\beta}; y_i, \boldsymbol{s}_i, \boldsymbol{x}_i) = \begin{cases}
\log\left(F'\left(h\left(y_i \mid \boldsymbol{s}_i\right) - \boldsymbol{x}_i^\top \boldsymbol{\beta}\right)\right) + \log\left(h'(y_i \mid \boldsymbol{s}_i)\right) & y_i & \text{exact} \\
\log\left(F\left(h(\bar{y} \mid \boldsymbol{s}_i) - \boldsymbol{x}_i^\top \boldsymbol{\beta}\right)\right) & y_i \in (-\infty, \bar{y}] & \text{left} \\
\log\left(1 - F\left(h(\underline{y} \mid \boldsymbol{s}_i) - \boldsymbol{x}_i^\top \boldsymbol{\beta}\right)\right) & y_i \in (\underline{y}, \infty) & \text{right} \\
\log\left(F\left(h(\bar{y} \mid \boldsymbol{s}_i) - \boldsymbol{x}_i^\top \boldsymbol{\beta}\right) - F\left(h(\underline{y} \mid \boldsymbol{s}_i) - \boldsymbol{x}_i^\top \boldsymbol{\beta}\right)\right) & y_i \in (\underline{y}, \bar{y}] & \text{interval.}
\end{cases}
$$

The joint log-likelihood of several observations $\{(y_i, \boldsymbol{x}_i, \boldsymbol{s}_i)\}_{i=1}^n$ is obtained by summing over the individual log-likelihood contributions ℓ_i under the assumption that the individual samples are independent and identically distributed, the case exclusively dealt with by **tramnet**.

Regularization

The aim of **tramnet** is to enable the estimation of regularized stratified linear transformation models. This is achieved by optimizing a penalized form of the log-likelihood introduced in the last section. The penalized log-likelihood,

$$
\tilde{\ell}(\boldsymbol{\vartheta}, \boldsymbol{\beta}, \lambda, \alpha; y, \boldsymbol{s}, \boldsymbol{x}) = \ell(\boldsymbol{\vartheta}, \boldsymbol{\beta}; y, \boldsymbol{s}, \boldsymbol{x}) - \lambda\left(\alpha \left\|\boldsymbol{\beta}\right\|_1 + \frac{1}{2}(1-\alpha)\left\|\boldsymbol{\beta}\right\|_2^2\right),
$$

consists of the unpenalized log-likelihood and an additional penalty term. Note that only the shift parameters $\boldsymbol{\beta}$ are penalized, whereas the coefficients for the baseline transformation $\boldsymbol{\vartheta}$ remain unpenalized. The parameterization of the penalty is chosen to be the same as in **glmnet**, consisting of a global penalization parameter λ, and a mixing parameter α controlling the amount of L_1 compared to L_2 penalization.

The two penalties and any combination thereof have unique properties and may be useful under different circumstances. A pure L_1 penalty was first introduced by Tibshirani (1996) in an OLS framework and was dubbed the Lasso (Least Absolute Shrinkage and Selection Operator) due to its property of shrinking regression coefficients exactly to 0 for large enough λ. A pure Lasso penalty can be obtained in a regularized transformation model by specifying $\alpha = 1$. Applying an L_2 penalty in an OLS problem was introduced more than five decades earlier by Tikhonov (1943) and later termed ridge regression (Hoerl and Kennard, 1970). In contrast to Lasso, ridge regression leads to shrunken regression coefficients but does not perform automatic variable selection. Zou and Hastie (2005) picked up on both approaches, discussed their advantages, disadvantages, and overall characteristics and combined them into the elastic net penalty, a convex combination of an L_1 and L_2 penalty controlled by the mixing parameter α. Some of these properties will be illustrated for different models and a real-world data set in Sections Censoring and likelihood forms and Hyperparameter tuning.

Constrained convex optimization

Special algorithms were developed to optimize regularized objective functions, most prominently the LARS and LARS-EN algorithm (Efron et al., 2004) and variants thereof for the penalized Cox model (Goeman, 2010). However, the aim of **tramnet** is to solve the objective functions arising in regularized transformation models in a single computational framework. Due to the log-concavity of all choices for F in this package and $h(y|\boldsymbol{x}, \boldsymbol{s})$ being monotone increasing in y, the resulting log-likelihood contributions for any form of censoring and truncation are concave and can thus be solved by constrained convex optimization.

The fairly recent development of **CVXR** allows the specification of constrained convex optimization problems in terms of a domain-specific language, yielding an intuitive and highly flexible framework for constrained optimization. Because checking the convexity of an arbitrarily complex expression is extremely hard, **CVXR** makes use of a library of smaller expressions, called atoms, with known monotonicity and curvature and tries to decompose the objective at hand using a set of rules from disciplined convex optimization (DCP, Grant et al., 2006). Thus, a complex expression's curvature can be more easily determined.

More formally, convex optimization aims at solving a problem of the form

$$\begin{array}{ll} \underset{\boldsymbol{\vartheta}}{\text{minimize}} & g(\boldsymbol{\vartheta}) \\ \text{subject to} & g_i(\boldsymbol{\vartheta}) \leq 0, \ i = 1, \ldots, K, \\ & \boldsymbol{A}\boldsymbol{\vartheta} = \boldsymbol{b}, \end{array}$$

where $\boldsymbol{\vartheta} \in \mathbb{R}^p$ is the parameter vector, $g(\boldsymbol{\vartheta})$ is the objective function to be optimized, $g_i(\boldsymbol{\vartheta})$ specify the inequality constraints, and $\boldsymbol{A} \in \mathbb{R}^{n \times p}$ and $\boldsymbol{b} \in \mathbb{R}^p$ parameterize any equality constraints on $\boldsymbol{\vartheta}$. Importantly, the objective function and all inequality constraint functions are convex (Boyd and Vandenberghe, 2004).

The likelihood $\sum_i \ell(\boldsymbol{\vartheta}, \boldsymbol{\beta}; y_i, \boldsymbol{s}_i, \boldsymbol{x}_i)$ for transformation models of the form (1) are convex for error distributions with log-concave density because log-convexity of $-F'$ ensures the existence and uniqueness of the most likely transformation $\hat{h}$ and the convexity of $-\ell(h; y, \boldsymbol{x})$. Because the penalty term is convex in $\boldsymbol{\beta}$, it can be added to the negative log-likelihood while conserving convexity. However, monotonicity of h imposes inequality constraints on the parameters of the baseline transformation, as illustrated in equation (2). The elegance of domain-specific language-based optimizers comes to play when adding these and potential other inequality or equality constraints to the objective function, which will be showcased in Section Additional constraints. Thus, the optimization routines implemented in package **CVXR** can be applied for computing maximum likelihood estimates of the parameters of model (1).

Model-based optimization

The predictive capabilities of regularized regression models heavily depend on the hyperparameters α and λ. Hyperparameter tuning can be addressed by a multitude of methods with varying computational complexity, advantages, and disadvantages. Naive or random grid search for more than one tuning parameter are computationally demanding, especially if the objective function is expensive to evaluate. Model-based optimization circumvents this issue by fitting a surrogate model, usually a Gaussian process, to the objective function. The objective function is evaluated at an initial, *e.g.*, , a random latin hypercube, design, to which the Gaussian process is subsequently fit. The surrogate model then proposes the next set of hyperparameters at which to evaluate the objective function by some infill criterion (Horn and Bischl, 2016). Bischl et al. (2017) implement model-based optimization for multi-objective blackbox functions in the **mlrMBO** package. The objective function can, in theory, be vector-valued and the tuning parameter spaces may be categorical. In **tramnet**, the objective function is the cross-validated log-likelihood optimized using a Kriging surrogate model with expected improvement as the infill criterion. Model-based optimization for hyperparameter tuning is illustrated in Section Prostate cancer data analysis.

Basic usage

The initial step is fitting a potentially stratified transformation model of the form

```
R> m1 <- tram(y | s ~ 1, ...)
```

omitting all explanatory variables. This sets up the basis expansion for the transformation function, whose regression coefficients will not be penalized, as mentioned in Section 2.1.2. Additionally,

Model Class	Censoring Type			
	Exact	Left	Right	Interval
BoxCox	✓	✗	✗	✗
Colr	✓	✓	✓	✗
Coxph	✓	✗	✓	✗
Lehmann	✓	✓	✗	✗

Table 1: Combinations of possible model classes and censoring types in the **tramnet** package. Due to missing optimization rules in **CVXR**, not every combination of error distribution and censoring type yield solvable objective functions. This will change with coming updates in the **CVXR** package.

`tramnet()` needs a model matrix including the predictors, whose regression coefficients ought to be penalized. For numerical reasons, it is useful to provide a scaled model matrix instead of the original data, such that every parameter is equally affected by the regularization. Lastly, `tramnet()` will need the tuning parameters $\alpha \in [0,1]$ and $\lambda \in \mathbb{R}^+$, with α representing a mixing parameter and λ controlling the extent of regularization. Setting $\lambda = 0$ will result in an unpenalized model, regardless of the value of α.

```
R> x <- model.matrix(~ 0 + x, ...)
R> x_scaled <- scale(x)
R> mt <- tramnet(model = m1, x = x_scaled, lambda, alpha, ...)
```

`S3` methods accompanying the `"tramnet"` class will be discussed in Section S3 Methods.

Censoring and likelihood forms

Specific combinations of F and the form of censoring yield log-log-concave log-likelihoods. Under these circumstances, **tramnet** is not yet able to solve the resulting optimization problem. Table 1 indicates which model class can be fitted under what type of censoring in the current version of **tramnet**.

Prostate cancer data analysis

The regularized normal linear and extensions to transformed normal regression models will be illustrated using the `Prostate` data set (Stamey et al., 1989), which was used by Zou and Hastie (2005) to highlight properties of the elastic net.

```
R> data("Prostate", package = "lasso2")
R> Prostate$psa <- exp(Prostate$lpsa)
R> Prostate[, 1:8] <- scale(Prostate[, 1:8])
```

The data set contains 97 observations and 9 covariates. In the original paper, the authors chose the log-transformed prostate specific antigen concentration (`lpsa`) as the response and used the eight remaining predictors log cancer volume (`lcavol`), log prostate weight (`lweight`), age of the patient (`age`), log benign prostatic hyperplasia amount (`lbph`), seminal vesicle invasion (`svi` coded as 1 for yes, 0 for no), log capsular penetration (`lcp`), Gleason score (`gleason`), and percentage Gleason score 4 or 5 (`pgg45`) as covariates.

Linear and Box-Cox type regression models

Zou and Hastie (2005) imposed an assumption on the conditional distribution of the response by log-transforming and fitting a linear model. In the following, it is shown that the impact of this assumption may be assessed by estimating the baseline transformation from the data, followed by a comparison with the log-transformation applied by Zou and Hastie (2005). The linear models in `lpsa` and log(`psa`) are compared to transformation models with basis expansions in both log(`psa`) and `psa`, while specifying conditional normality of the transformed response. Additionally, the models are compared to an alternative implementation of regularized normal linear regression in **penalized**. Five different models will be used to illustrate important facets of transformation models, including parameterization and interpretation. The models are summarized in Table 2 and will be elaborated on throughout this section. The comparison is based on unpenalized models first. Later, the section highlights the penalized models together with hyperparameter tuning.

Name	Code	Model for $F_{Y\|\boldsymbol{X}=\boldsymbol{x}}(y\|\boldsymbol{x})$
mp	`penalized(response = lpsa, penalized = x)`	$\Phi\left(\vartheta_1 + \vartheta_2 \log(y) - \boldsymbol{x}^\top\boldsymbol{\beta}\right)$
mt	`Lm(lpsa ~ .)`	$\Phi\left(\vartheta_1 + \vartheta_2 \log(y) - \boldsymbol{x}^\top\boldsymbol{\beta}\right)$
mtp	`BoxCox(psa ~ ., log_first = TRUE, order = 1)`	$\Phi\left(\boldsymbol{a}_{\text{Bs},1}(\log(y))^\top\boldsymbol{\vartheta} - \boldsymbol{x}^\top\boldsymbol{\beta}\right)$
mt1	`BoxCox(psa ~ ., log_first = TRUE, order = 7)`	$\Phi\left(\boldsymbol{a}_{\text{Bs},7}(\log(y))^\top\boldsymbol{\vartheta} - \boldsymbol{x}^\top\boldsymbol{\beta}\right)$
mt2	`BoxCox(psa ~ ., log_first = FALSE, order = 11)`	$\Phi\left(\boldsymbol{a}_{\text{Bs},11}(y)^\top\boldsymbol{\vartheta} - \boldsymbol{x}^\top\boldsymbol{\beta}\right)$

Table 2: Summary of the five models illustrated in Section Prostate cancer data analysis, including their name throughout the manuscript, the R code to fit them, and the mathematical formulation of their conditional cumulative distribution function. For comparison, `mp` is included as an ordinary linear model, which is equivalent to model `mt` in terms of log-likelihood, but differs in the parameterization of the transformation function h and thus yields scaled coefficient estimates (*cf.* Table 3). Model `mtp` is a linear model parameterized in terms of a Bernstein basis of maximum order 1. This will yield the same coefficient estimates as `mt`, but a log-likelihood that is comparable to models `mt1` and `mt2`, whose transformation functions are parameterized in terms of higher-order Bernstein bases. The `log_first` argument specifies whether the basis expansion is calculated on the log-transformed or untransformed response.

```
R> fm_Pr <- psa ~ lcavol + lweight + age + lbph + svi + lcp + gleason + pgg45
R> fm_Pr1 <- update(fm_Pr, ~ 0 + .)
R> x <- model.matrix(fm_Pr1, data = Prostate)
```

The normal linear regression model is implemented in **tram**'s `Lm()` function. `Lm()`'s parameterization differs from the usual linear model, hence caution has to be taken when interpreting the resulting regression coefficients $\boldsymbol{\beta}$. In order to compare the results to an equivalent, already existing implementation, the same model is fitted using **penalized**.

```
R> m0 <- Lm(lpsa ~ 1, data = Prostate)
R> mt <- tramnet(m0, x = x, alpha = 0, lambda = 0)
R> mp <- penalized(response = Prostate$lpsa, penalized = x,
+                  lambda1 = 0, lambda2 = 0)
```

A linear model of the form

$$Y = \tilde{\alpha} + \boldsymbol{x}^\top\tilde{\boldsymbol{\beta}} + \varepsilon, \quad \varepsilon \sim \mathrm{N}(0, \sigma^2)$$

can be understood as a transformation model through reparameterization as

$$\mathbb{P}(Y \leq y|\boldsymbol{X} = \boldsymbol{x}) = \Phi\left(\vartheta_1 + \vartheta_2 y - \boldsymbol{x}^\top\boldsymbol{\beta}\right).$$

Here, $\vartheta_1 = -\tilde{\alpha}/\sigma$ is a reparameterized intercept term, $\vartheta_2 = 1/\sigma$ is the slope of the baseline transformation, and the regression coefficients $\boldsymbol{\beta} = \tilde{\boldsymbol{\beta}}/\sigma$ represent scaled shift terms, influencing only the intercept. To recover the usual parameterization, `tramnet::coef.Lm()` offers the `as.lm = TRUE` argument.

```
R> cfx_tramnet <- coef(mt, as.lm = TRUE)
```

The transformation function for the linear model is depicted in Figure 1 (pink line). Because a linear baseline transformation imposes restrictive assumptions on the response's conditional distribution, it is advantageous to replace the linear baseline transformation by a more flexible one. In the case of the Box-Cox type regression model, the linear baseline transformation $h(y) = \vartheta_1 + \vartheta_2 \log y$ is replaced by the basis expansion $h(y) = \boldsymbol{a}_{\text{Bs},7}(\log y)^\top\boldsymbol{\vartheta}$.

```
R> ord <- 7 # flexible baseline transformation
R> m01 <- BoxCox(psa ~ 1, data = Prostate, order = ord,
+                extrapolate = TRUE, log_first = TRUE)
R> mt1 <- tramnet(m01, x = x, alpha = 0, lambda = 0)
```

The Box-Cox type regression model is then estimated with the `BoxCox()` function while specifying the appropriate maximum order of the Bernstein polynomial. Because the more flexible transformation slightly deviates from being linear, the normal linear model yields a smaller log-likelihood (*cf.* Table 3).

To make sure that this improvement is not due to the increased number of parameters and hence overfitting, the models' predictive capacities could be compared via cross-validation.

These results hold for the pre-specified log transformation of the response and a basis expansion thereof. Instead of prespecifying the log-transformation, its "logarithmic nature" can be estimated from the data. Afterward, one can compare the deviation from a log-linear baseline transformation graphically and by inspecting the predictive performance of the model in terms of out-of-sample log-likelihood.

```
R> m02 <- BoxCox(psa ~ 1, order = 11, data = Prostate, extrapolate = TRUE)
R> mt2 <- tramnet(m02, x = x, lambda = 0, alpha = 0)
```

Indeed, the baseline transformation in Figure 1 is similar to the basis expansion in the log-transformed response upon visual inspection. Because `mt` is estimated using the log-transformed response and `mt1` and `mt2` are based on the original scale of the response, the resulting model log-likelihoods are not comparable. To overcome this issue, one can fit a Box-Cox type model with maximum order 1, as this results in a linear but alternatively parameterized baseline transformation.

```
R> m0p <- BoxCox(psa ~ 1, order = 1, data = Prostate, log_first = TRUE)
R> mtp <- tramnet(m0p, x = x, lambda = 0, alpha = 0)
```

Figure 1 plots the three distinct baseline transformations resulting from models `mt`, `mt1`, and `mt2`. The initial assumption to model the prostate-specific antigen concentration linearly on the log-scale seems to be valid when comparing the three transformation functions. The linear transformation in `lpsa` used in `mt`, and the basis expansion in log(`psa`) (`mt1`) are almost indistinguishable and yield very similar coefficient estimates, as well as log-likelihoods (*cf.* Table 3, `mtp` *vs.* `mt1`). The basis expansion in `psa` (`mt2`) is expected to be less stable due to the highly skewed untransformed response. This is reflected in Figure 1, where the baseline transformation deviates from being linear towards the bounds of the response's support. However, the log-linear behavior of h was clearly captured by this model and further supported the initial assumption of conditional log-normality of the response. For the same reasons, the resulting log-likelihood of `mt2` is smaller than for `mt1` (Table 3). Taken together, this exemplary analysis highlights the flexibility and usefulness of transformation models to judge crucial modeling assumptions.

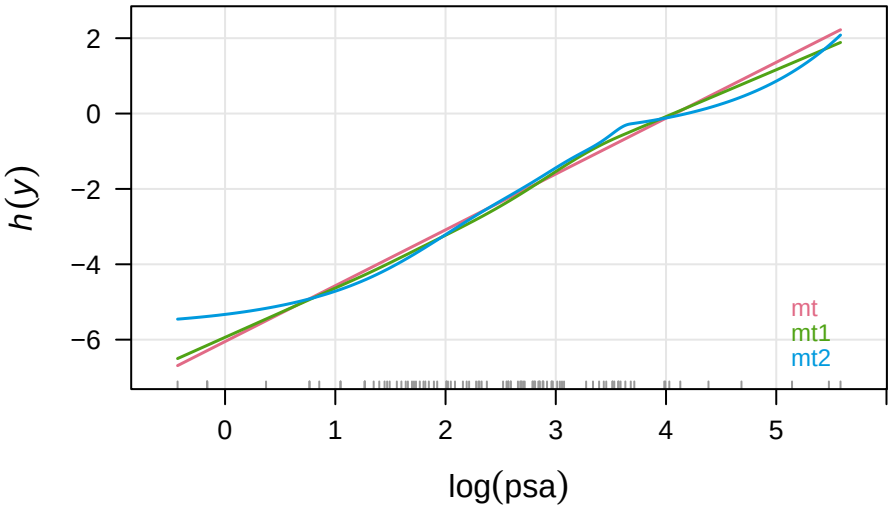


Figure 1: Comparison of different choices for the baseline transformation of the response (prostate-specific antigen concentration) in the `Prostate` data. The original analysis prespecified a log-transformation of the response and then assumed conditional normality on this scale. Hence the baseline transformation of `mt` is of the form: $h(\text{lpsa}) = \vartheta_1 + \vartheta_2 \cdot \text{lpsa}$. Now, one can allow a more flexible transformation function in log(`psa`) to judge any deviations of $h(\log(\text{psa}))$ from linearity, leading to a baseline transformation in terms of basis functions: $\boldsymbol{a}_{\text{Bs},7}(\log(\text{psa}))^\top \boldsymbol{\vartheta}$ in `mt1`. Lastly, instead of presuming a log-transformation, one could estimate the baseline transformation from the raw response (`psa`), *i.e.*, $h(\text{psa}) = \boldsymbol{a}_{\text{Bs},11}(\text{psa})^\top \boldsymbol{\vartheta}$ in `mt2`. In this case, a higher-order basis expansion was chosen to account for the skewness of the raw response. Notably, all three baseline transformations are fairly comparable. The basis expansion in `psa` deviates from being log-linear towards the boundaries of the response's support, as there are only a few observations.

Model	Coefficient estimates								logLik
	lcavol	lweight	age	lbph	svi	lcp	gleason	pgg45	
mp	0.69	0.23	-0.15	0.16	0.32	-0.15	0.03	0.13	-99.5
mt	1.03	0.33	-0.22	0.23	0.47	-0.22	0.05	0.19	-99.5
mtp	1.03	0.33	-0.22	0.23	0.47	-0.22	0.05	0.19	-339.9
mt1	1.03	0.34	-0.21	0.22	0.48	-0.23	0.04	0.22	-338.0
mt2	0.97	0.32	-0.19	0.22	0.48	-0.21	0.07	0.21	-343.5

Table 3: Comparison of the three transformation models on the `Prostate` data. Coefficient estimates are shown for each model, together with the in-sample log-likelihood in the last column. The first three models, `mp`, `mt`, and `mtp` use a linear baseline transformation in `lpsa` and log(`psa`), respectively. The `mp` model was fit using **penalized** and gave the scaled version of the regression coefficients in `mt`, but the same log-likelihood. At the same time, `mt` and `mtp` differ only in their response variable and its subsequent log-transformation in `mtp`, yielding the same coefficient estimates but a different log-likelihood. Models `mt1` and `mt2` allow a flexible basis expansion in log(`psa`) and `psa`, respectively. Model `mt1`, allowing for a flexible basis expansion in `lpsa`, fits the data the best. However, the resulting coefficient estimates are similar for all models.

Hyperparameter tuning

This section features cross-validation, model-based optimization, and profiling functions for hyperparameter tuning, whose appropriate values are highly problem-dependent and hard to know in advance. **tramnet** implements naive grid search and model-based optimization in the functions `cvl_tramnet()` and `tramnet_mbo()`, respectively. In the framework of regularized transformation models, it is very natural to choose the out-of-sample log-likelihood as the objective function because the notion of a mean square loss does not make sense for survival, let alone censored outcomes. The out-of-sample log-likelihood is, in fact, the log score, which is a proper scoring rule (Gneiting and Raftery, 2007).

```
R> m0 <- BoxCox(lpsa ~ 1, data = Prostate, order = 7, extrapolate = TRUE)
R> mt <- tramnet(m01, x = x, alpha = 1, lambda = 0)
```

tramnet offers cross-validation in `cvl_tramnet()`, comparable to the `optL1()`, and `optL2()` functions in **penalized**, which takes a sequence of values for λ and α and performs a simple – and arguably slow – grid search. Per default, it computes 2-fold cross-validation. The user is encouraged, however, to judge the resulting bias-variance trade-off accordingly.

```
R> lambdas <- c(0, 10^seq(-4, log10(15), length.out = 4))
R> cvlt <- cvl_tramnet(object = mt, fold = 2, lambda = lambdas, alpha = 1)
```

In order to compare cross-validation across multiple packages and functions, it is also possible to supply the folds for each row in the design matrix as a numeric vector, as for example returned by `penalized::optL1()`.

```
R> pen_cvl <- optL1(response = lpsa, penalized = x, lambda2 = 0, data = Prostate,
+                   fold = 2)
R> cvlt <- cvl_tramnet(object = mt, lambda = lambdas, alpha = 1,
+                      folds = pen_cvl$fold)
```

The resulting object is of class `"cvl_tramnet"` and contains a table for the cross-validated log-likelihoods for each fold and the sum thereof, the 'optimal' tuning parameter constellation, which resulted in the largest cross-validated log-likelihood, tables for the cross-validated regularization paths, the folds and lastly the full fit based on the 'optimal' tuning parameters. Additionally, the resulting object can be used to visualize the log-likelihood and coefficient trajectories. These trajectories highly depend on the chosen folds, and the user is referred to the full profiling functions discussed in Section Regularization paths.

Model-based optimization

In contrast to naive grid search, model-based optimization comprises more elegant methods for hyperparameter tuning. **tramnet** offers the `mbo_tramnet()` and `mbo_recommended()` functions. The former implements Kriging-based hyperparameter tuning for the elastic net, the Lasso, and ridge regression. `mbo_tramnet()` takes a `"tramnet"` object as input and computes the cross-validated log-likelihood based on the provided `fold` or `folds` argument. The initial design is a random latin

hypercube design with `n_design` rows per parameter. The number of sequential fits of the surrogate models is specified through `n_iter`, and the range of the tuning parameters can be specified by `max/min` arguments. The default infill criterion is expected improvement. However, Bischl et al. (2017) encourage the use of the lower confidence bound over expected improvement, which can be achieved in `mbo_tramnet()` by specifying `opt_crit = makeMBOInfillCritCB()`. 10-fold cross-validation is used to compute the objective function for the initial design and each iteration. The recommended model is then extracted using `mbo_recommended()`.

```
R> tmbo <- mbo_tramnet(mt, obj_type = "elnet", fold = 10)
R> mtmbo <- mbo_recommended(tmbo, m0, x)
```

Unlike in the previous section, one can directly optimize the tuning parameters in an elastic net problem instead of optimizing over one hyperparameter at a time or having to specify Lasso or ridge regression *a priori*. The output of `mbo_tramnet()` is quite verbose and can be shortened by using the helper function `print_mbo()`.

```
R> print_mbo(tmbo)

## Recommended parameters:
## lmb=1.04e-05; alp=0.751
## Objective: y = 710
```

Interpreting the output, model-based optimization suggests an unpenalized model with $\alpha = 0.75$ and $\lambda = 0$. This result stresses the advantages of model-based optimization over naive or random grid search in terms of complexity and computational efficiency. In the end, the proposed model is unpenalized and thus does not introduce sparsity in the regression coefficients.

```
R> coef(mtmbo)

## lcavol lweight     age    lbph     svi     lcp gleason   pgg45
## 1.0312  0.3380 -0.2068  0.2198  0.4801 -0.2329  0.0437  0.2157

R> summary(mtmbo)$sparsity
## [1] "8 regression coefficients, 8 of which are non-zero"
```

Regularization paths

As discussed before, it may be useful to inspect the full regularization paths over the tuning parameters λ and α. Akin to the functions `profL1()` and `profL2()` in package **penalized**, **tramnet** offers `prof_lambda()` and `prof_alpha()`. Since these functions take a fitted model of class `"tramnet"` as input, which is internally updated, it is crucial to correctly specify the other tuning parameter in the model fitting step. In the example to come, `mt` was fit using $\alpha = 1$ and $\lambda = 0$, resulting in a Lasso penalty only when profiling over λ. The resulting profile is depicted in Figure 2.

```
R> pfl <- prof_lambda(mt)
```

`prof_lambda()` takes `min_lambda`, `max_lambda`, and `nprof` as arguments and internally generates an equi-spaced sequence from `min_lambda` to `max_lambda` on the log scale of length `nprof`. By default, this sequence ranges from 0 to 15 and is of length 5.

```
R> plot_path(pfl, plot_logLik = FALSE, las = 1, col = col1)
```

Additional constraints

In some applications, the specification of additional constraints on the shift parameters $\boldsymbol{\beta}$ are of interest. Most commonly, either positivity or negativity for some or all regression coefficients is aimed at. In **tramnet**, additional inequality constraints can be specified via the `constraints` argument, which are internally handled as `constraints[[1]] %*% beta > constraints[[2]]`. Hence, to specify the constraint of strictly positive regression coefficients, one would supply an identity matrix of dimension p for the left-hand side and the zero p-vector for the right-hand side, as done in the following example.

```
R> m0 <- BoxCox(lpsa ~ 1, data = Prostate, extrapolate = TRUE)
R> mt <- tramnet(m0, x, alpha = 0, lambda = 0, constraints = list(diag(8),
+                                                                 rep(0, 8)))
R> coef(mt)
```

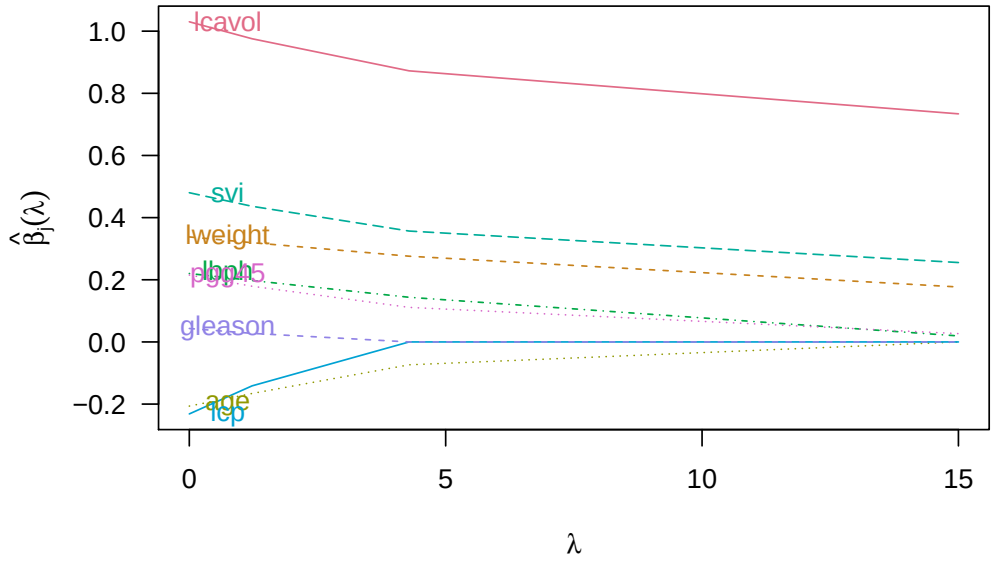


Figure 2: Full regularization paths for the tuning parameter λ using the default values of `plot_path()`.

```
## lcavol lweight    lbph     svi gleason   pgg45
## 0.9111  0.2996  0.1684  0.3969  0.0133  0.1125
```

The coefficients with a negative sign in the model without additional positivity constraints now shrink to zero, with the other coefficient estimates changing as well.

```
R> summary(mt)$sparsity
```

```
## [1] "8 regression coefficients, 6 of which are non-zero"
```

One can compare this model to the implementation in **tram**, where it is also possible to specify linear inequality constraints on the regression coefficients $\boldsymbol{\beta}$. Here, it is sufficient to specify `constraints = c("age >= 0","lcp >= 0")` for the two non-positive coefficient estimates.

```
R> m <- BoxCox(lpsa ~ . - psa, data = Prostate, extrapolate = TRUE,
+              constraints = c("age >= 0", "lcp >= 0"))
R> max(abs(coef(m) - coef(mt, tol = 0)))
```

```
## [1] 1.28e-05
```

Indeed, both optimizers arrive at virtually the same coefficient estimates.

S3 Methods

Building on the S3 infrastructure of the packages **mlt** and **tram**, this package provides corresponding methods for the following generics: `coef()`, `logLik()`, `plot()`, `predict()`, `simulate()`, and `residuals()`. The methods' additional `"tramnet"`-specific arguments will be briefly discussed in this section.

`coef.tramnet()` suppresses the baseline transformation's coefficient estimates $\hat{\boldsymbol{\vartheta}}$ by default and considers shift parameter estimates $\hat{\boldsymbol{\beta}}$ below 10^{-6} as 0 to stress the selected variables only. This threshold can be controlled by the `tol` argument. Hence, `coef(mt,with_baseline = TRUE,tol = 0)` returns all coefficients.

```
R> coef(mtmbo, with_baseline = TRUE, tol = 0)
```

```
## Bs1(lpsa) Bs2(lpsa) Bs3(lpsa) Bs4(lpsa) Bs5(lpsa) Bs6(lpsa) Bs7(lpsa)
##   -1.9775   -1.5055   -1.0335   -0.2778   -0.2778    1.0723    1.5150
## Bs8(lpsa)    lcavol   lweight       age      lbph       svi       lcp
##    1.9576    1.0312    0.3380   -0.2068    0.2198    0.4801   -0.2329
```

```
## gleason     pgg45
##    0.0437    0.2157
```

The `logLik.tramnet()` method allows the log-likelihoods re-computation under new data (*i.e.*, out-of-sample) and different coefficients (`parm`) and weights (`w`), as illustrated below.

```
R> logLik(mtmbo)

## 'log Lik.' -97.7 (df=NA)

R> cfx <- coef(mtmbo, with_baseline = TRUE, tol = 0)
R> cfx[5:8] <- 0.5
R> logLik(mtmbo, parm = cfx)

## 'log Lik.' -561 (df=NA)

R> logLik(mtmbo, newdata = Prostate[1:10,])

## 'log Lik.' -14.3 (df=NA)

R> logLik(mtmbo, w = runif(n = nrow(mtmbo$x)))

## 'log Lik.' -41.8 (df=NA)
```

In the spirit of **mlt**'s plotting methods for classes `"mlt"` and `"ctm"`, `plot.tramnet()` offers diverse plotting options for objects of class `"tramnet"`. The specification of new data and the type of plot is illustrated in the following code chunk and Figure 3.

```
R> par(mfrow = c(3, 2)); K <- 1e3
R> plot(mtmbo, type = "distribution", K = K, main = "A") # A, default
R> plot(mtmbo, type = "survivor", K = K, main = "B") # B
R> plot(mtmbo, type = "trafo", K = K, main = "C") # C
R> plot(mtmbo, type = "density", K = K, main = "D") # D
R> plot(mtmbo, type = "hazard", K = K, main = "E") # E
R> plot(mtmbo, type = "trafo", newdata = Prostate[1, ], col = 1, K = K, main = "F") # F
```

The `predict.tramnet()` method works in the same way as `predict.mlt()` and as such supports the types `trafo`, `distribution`, `survivor`, `density`, `logdensity`, `hazard`, `loghazard`, `cumhazard`, and `quantile`. For `type = "quantile"`, the corresponding probabilities (`prob`) have to be supplied as an argument to evaluate the quantile function.

```
R> predict(mtmbo, type = "quantile", prob = 0.2, newdata = Prostate[1:5,])

## prob    [,1] [,2] [,3] [,4] [,5]
##     0.2  3.4 3.55 3.74 3.72 2.68
```

Another method offered by this package implements parametric bootstrap-based sampling. In particular, `simulate.tramnet()` calls the `simulate.ctm()` function after converting the `"tramnet"` object to a `"ctm"` object.

```
R> simulate(mtmbo, nsim = 1, newdata = Prostate[1:5,], seed = 1)

## [1] 3.56 3.97 4.57 5.48 2.69
```

Lastly, `residuals.tramnet()` computes the generalized residual r defined as the score contribution for sample i with respect to a newly introduced intercept parameter γ, which is restricted to be zero. In particular,

$$r = \left. \frac{\partial \ell \left(\boldsymbol{\vartheta}, \boldsymbol{\beta}, \gamma; y, \boldsymbol{s}, \boldsymbol{x} \right)}{\partial \gamma} \right|_{\gamma = 0}$$

yields the generalized residual with respect to γ for the model

$$F_Y(y|\boldsymbol{s}, \boldsymbol{x}) = F\left(h(y \mid \boldsymbol{s}) - \boldsymbol{x}^\top \boldsymbol{\beta} - \gamma \right).$$

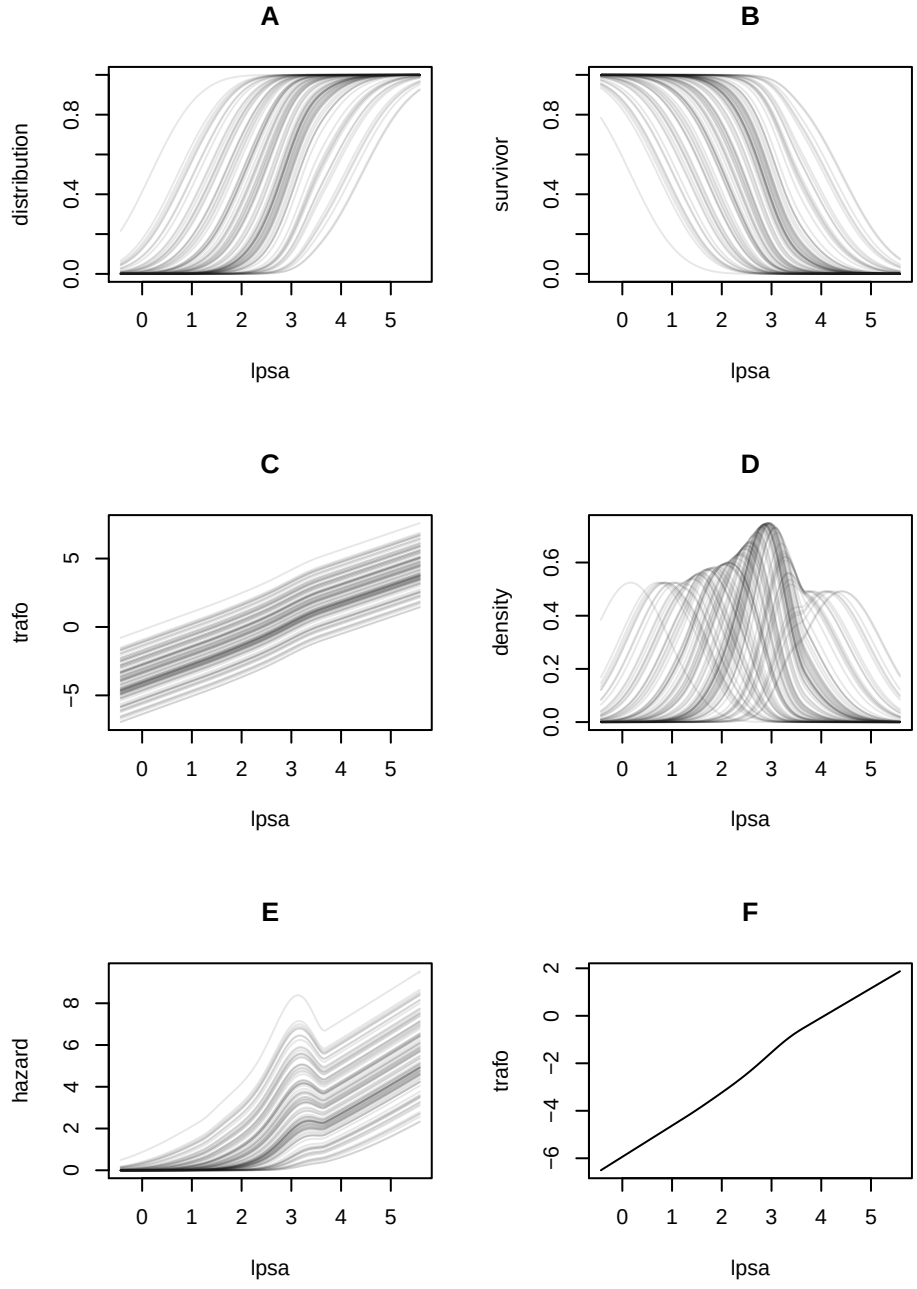


Figure 3: Illustration of `plot.tramnet()`'s versatility in visualizing the response's estimated conditional distribution on various scales, including cdf, survivor, transformation scale and pdf. Note that, by default, the plot is produced for each row in the design matrix. In unstratified linear transformation models, this leads to shifted versions of the same curve on the transformation function's scale. A: Estimated conditional distribution function for every observation. B: Estimated conditional survivor function for every observation. The conditional survivor function is defined as $S(y|\boldsymbol{x}) = 1 - F_Y(y|\boldsymbol{x})$. C: Conditional most likely transformation for every observation. Note that every conditional transformation function is a shifted version of the same curve. D: The conditional density for every observation can be calculated using $f_Y(y|\boldsymbol{x}) = F'(\boldsymbol{a}(y)^\top \boldsymbol{\vartheta} - \boldsymbol{x}^\top \boldsymbol{\beta})\boldsymbol{a}'(y)^\top \boldsymbol{\vartheta}$. E: A distribution function is fully characterized by its hazard function $\lambda(y|\boldsymbol{x}) = f_Y(y|\boldsymbol{x})/S(y|\boldsymbol{x})$, which is depicted in this panel. F: The `newdata` argument can be used to plot the predicted most likely transformation for the provided data, in this case, the first row of the `Prostate` data.

```
R> residuals(mtmbo)[1:5]

## [1] -6.50 -6.36 -6.60 -6.57 -4.17
```

In residual analysis and boosting, it is common practice to check for associations between residuals and covariates that are not included in the model. In the prostate cancer example, one could investigate whether the variables age and lcp should be included in the model. To illustrate this particular case, a nonparametric independence_test() from package **coin** can be used (Hothorn et al., 2008). First, the uncoditional transformation model m0 is fitted. Afterward, the tramnet models excluding age and lcp are estimated, and their residuals extracted using the residuals.tramnet() method. Lastly, an independence test using a maximum statistic (teststat = "max") and a Monte Carlo-based approximation of the null distribution based on resampling 10^6 times (distribution = approximate(1e6)) yields the results printed below.

```
R> library("coin")
R> m0 <- BoxCox(lpsa ~ 1, data = Prostate, extrapolate = TRUE)
R> x_no_age_lcp <- x[, !colnames(x) %in% c("age", "lcp")]
R> mt_no_age_lcp <- tramnet(m0, x_no_age_lcp, alpha = 0, lambda = 0)
R> r <- residuals(mt_no_age_lcp)
R> it <- independence_test(r ~ age + lcp, data = Prostate,
+                          teststat = "max", distribution = approximate(1e6))
R> pvalue(it, "single-step")

## age  0.023748
## lcp <0.000001
```

Because there is substantial evidence against the independence of the models' residuals to either lcp or age, we can conclude that it is worthwhile to include age and lcp in the model. Packages **trtf** (Hothorn, 2019b) and **tbm** (Hothorn, 2020a, 2019a) make use of this definition of a residual for estimating and boosting transformation models, trees, and random forests. For more theoretical insight, the reader is referred to the above mentioned publications.

Bibliography

B. Bischl, J. Richter, J. Bossek, D. Horn, J. Thomas, and M. Lang. *mlrMBO: A Modular Framework for Model-Based Optimization of Expensive Black-Box Functions*, 2017. URL http://arxiv.org/abs/1703.03373. [p583, 588]

G. E. Box and D. R. Cox. An analysis of transformations. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 26(2):211–243, 1964. doi: 10.1111/j.2517-6161.1964.tb00553.x. [p581]

S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004. doi: 10.1017/CBO9780511804441. [p583]

B. Efron, T. Hastie, I. Johnstone, R. Tibshirani, et al. Least angle regression. *The Annals of Statistics*, 32 (2):407–499, 2004. doi: 10.1214/009053604000000067. [p583]

J. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):1–22, 2010. doi: 10.18637/jss.v033.i01. [p581]

A. Fu, B. Narasimhan, and S. Boyd. CVXR: An R package for disciplined convex optimization. *Journal of Statistical Software*, 2020. URL http://arxiv.org/abs/1711.07582. Accepted for publication. [p581]

T. Gneiting and A. E. Raftery. Strictly proper scoring rules, prediction, and estimation. *Journal of the American Statistical Association*, 102(477):359–378, 2007. doi: 10.1198/016214506000001437. [p587]

J. J. Goeman. L1 penalized estimation in the Cox proportional hazards model. *Biometrical Journal*, 52 (1):–14, 2010. doi: 10.1002/bimj.200900028. [p581, 583]

M. Grant, S. Boyd, and Y. Ye. Disciplined convex programming. In *Global Optimization*, pages 155–210. Springer, 2006. doi: 10.1007/s11590-019-01422-z. [p583]

A. E. Hoerl and R. W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970. doi: 10.1080/00401706.1970.10488634. [p582]

D. Horn and B. Bischl. Multi-objective parameter configuration of machine learning algorithms using model-based optimization. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–8. IEEE, 2016. doi: 10.1109/SSCI.2016.7850221. [p583]

T. Hothorn. *tbm: Transformation Boosting Machines*, 2019a. URL https://CRAN.R-project.org/package=tbm. R package version 0.3-0. [p592]

T. Hothorn. *trtf: Transformation Trees and Forests*, 2019b. URL https://CRAN.R-project.org/package=trtf. R package version 0.3-5. [p592]

T. Hothorn. Transformation boosting machines. *Statistics and Computing*, 30:141–152, 2020a. doi: 10.1007/s11222-019-09870-4. [p592]

T. Hothorn. Most likely transformations: The mlt package. *Journal of Statistical Software*, 92(1):1–68, 2020b. doi: 10.18637/jss.v092.i01. [p581]

T. Hothorn. *tram: Transformation Models*, 2020c. URL http://ctm.R-forge.R-project.org. R package version 0.3-2. [p581, 582]

T. Hothorn, K. Hornik, M. A. van de Wiel, and A. Zeileis. Implementing a class of permutation tests: The coin package. *Journal of Statistical Software*, 28(8):1–23, 2008. doi: 10.18637/jss.v028.i08. [p592]

T. Hothorn, T. Kneib, and P. Bühlmann. Conditional transformation models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 76(1):3–27, 2014. doi: 10.1111/rssb.12017. [p581]

T. Hothorn, L. Möst, and P. Bühlmann. Most likely transformations. *Scandinavian Journal of Statistics*, 45(1):110–134, 2018. doi: 10.1111/sjos.12291. [p581]

T. Lohse, S. Rohrmann, D. Faeh, and T. Hothorn. Continuous outcome logistic regression for analyzing body mass index distributions. *F1000Research*, 6(1933), 2017. doi: 10.12688/f1000research.12934.1. [p581]

T. A. Stamey, J. N. Kabalin, J. E. McNeal, I. M. Johnstone, F. Freiha, E. A. Redwine, and N. Yang. Prostate specific antigen in the diagnosis and treatment of adenocarcinoma of the prostate. II. Radical prostatectomy treated patients. *The Journal of Urology*, 141(5):1076–1083, 1989. doi: 10.1016/S0022-5347(17)41176-1. [p584]

R. Tibshirani. Regression shrinkage and selection via the LASSO. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 58(1):267–288, 1996. doi: 10.1111/j.2517-6161.1996.tb02080.x. [p582]

A. N. Tikhonov. On the stability of inverse problems. In *Dokl. Akad. Nauk SSSR*, volume 39, pages 195–198, 1943. doi: 10.1155/2011/450269. [p582]

H. Zou and T. Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(2):301–320, 2005. doi: 10.1111/j.1467-9868.2005.00503.x. [p582, 584]

Lucas Kook, Torsten Hothorn
Institut für Epidemiologie, Biostatistik und Prävention
Universität Zürich
Hirschengraben 84, CH-8001 Zürich
Switzerland
lucasheinrich.kook@uzh.ch, Torsten.Hothorn@R-project.org

BayesSPsurv: An R Package to Estimate Bayesian (Spatial) Split-Population Survival Models

by Brandon Bolte, Nicolás Schmidt, Sergio Béjar, Nguyen Huynh, Bumba Mukherjee

Abstract Survival data often include a fraction of units that are susceptible to an event of interest as well as a fraction of "immune" units. In many applications, spatial clustering in unobserved risk factors across nearby units can also affect their survival rates and odds of becoming immune. To address these methodological challenges, this article introduces our **BayesSPsurv** R-package, which fits parametric Bayesian Spatial split-population survival (cure) models that can account for spatial autocorrelation in both subpopulations of the user's time-to-event data. Spatial autocorrelation is modeled with spatially weighted frailties, which are estimated using a conditionally autoregressive prior. The user can also fit parametric cure models with or without nonspatial i.i.d. frailties, and each model can incorporate time-varying covariates. **BayesSPsurv** also includes various functions to conduct pre-estimation spatial autocorrelation tests, visualize results, and assess model performance, all of which are illustrated using data on post-civil war peace survival.

Introduction

Conventional survival models have been applied to analyze time-to-event data across several academic disciplines, but these models rely on two core assumptions that are not always tenable. The first is that all units, including right-censored observations, will eventually experience the event of interest. In many applications, however, some fraction of subjects that are "immune" or "cured" may never experience the event (Maller and Zhou, 1996; Peng and Taylor, 2014; Beger et al., 2017). A clinical study of obesity on human death rates can employ a standard survival model because all subjects will eventually die, but a study on vaccine effectiveness will likely assume that some fraction of the treated population will become immune while others may respond differently and remain uncured. To account for both subpopulations, scholars have developed a class of split-population (SP) survival models that probabilistically separate the immune fraction from the units that are susceptible to the event of interest and then estimate the conditional hazard of survival among the latter units (Cai et al., 2012; Beger et al., 2018; Box-Steffensmeier and Zorn, 1999; Box-Steffensmeier and Jones, 2004). The second assumption of conventional survival models is that each observation is conditionally independent after controlling for covariates. This assumption is violated if spatially clustered units share common unobserved features that influence their baseline risk of experiencing an event (Darmofal, 2009; Taylor and Rowlingson, 2017). If spatial autocorrelation matters for process survival and/or the probability of being immune to an event, ignoring it can lead to biased parameter estimates.

Although there are numerous methods for dealing with spatially autocorrelated time-to-event data (e.g., Li and Ryan, 2002; Banerjee et al., 2003; Henderson et al., 2002; Zhou et al., 2020), these spatial survival models do not differentiate between at-risk and immune observations. Conversely, existing SP survival models either ignore spatial autocorrelation altogether or only account for it in the survival stage (Banerjee and Carlin, 2004). We present the **BayesSPsurv** (Bolte et al., 2021b) R-package (available at `https://CRAN.R-project.org/package=BayesSPsurv`), which has the power to overcome these limitations as well as the flexibility for researchers to model spatial clustering in their survival data however they choose to define it. In particular, **BayesSPsurv** allows the user to estimate parametric Bayesian Spatial split-population survival (cure) models with spatial frailties in both the model's split and survival stages. These models account for spatial clustering in the immune and at-risk fractions of the data. The package also includes functions and code for pre-estimation spatial autocorrelation diagnostics, visualizing results, simulating multiple Markov chains, and implementing Markov Chain Monte Carlo (MCMC) estimation routines for SP survival models with independent (exchangeable) or no random effects. The user can also include time-varying covariates in either stage. In the next section, we outline previous work on SP and spatial survival models, including existing packages and their limitations. We then formally develop the Pooled, Exchangeable, and Spatial Bayesian SP survival models before describing the various functions available in the **BayesSPsurv** package. We demonstrate these functions using replication data from a published study on the survival of post-civil war peace.

Background and other R-packages

Scholars have identified at least two sources of conditional heterogeneity in survival data that hasve been separately addressed. The first occurs when a subset of the units in the data are immune from experiencing a "failure" event, which violates the assumption that even right-censored observations experience the event of interest (Box-Steffensmeier and Zorn, 1999; Box-Steffensmeier and Jones, 2004; Beger et al., 2017). Cure rate or split-population (SP) survival models account for this immune fraction by first estimating the probability that right-censored units are immune from an event in a "split-stage" and then the time until *at-risk* censored and non-censored units experience the event. Recent work has extended these models to include i.i.d. frailties (Peng and Taylor, 2011), time-varying covariates (Beger et al., 2017), and account for random right-censoring (Patilea and Van Keilegom, 2020). Split-population survival models have been used to study phenomena, ranging from the survival of breast cancer patients (Wang et al., 2005), criminal recidivism (Schmidt and Witte, 1989), the risk of coups (Beger et al., 2017), and parasite-induced mortality in river salmon (Ray et al., 2014).

Separately, conventional survival models have been extended to account for spatial autocorrelation among nearby units (Banerjee et al., 2003; Taylor and Rowlingson, 2017; Zhou et al., 2020). These models relax the assumption of spatial independence by incorporating spatially weighted frailties into the survival model's baseline hazard function. This allows for the possibility that adjacent units share unmodeled risk factors that influence their underlying propensities for experiencing a failure event. Apart from recent advances in modeling spatial frailties in different survival frameworks (e.g. Diva et al., 2008; Zhou et al., 2020), spatial survival models have been applied to analyze, for example, geographically referenced data on leukemia survival (Henderson et al., 2002), position announcements by U.S. House members (Darmofal, 2009), prostate cancer (Zhou et al., 2020), and fire service response times (Taylor and Rowlingson, 2017). Despite these considerable methodological developments, far less attention has been dedicated to accounting for spatial autocorrelation in SP survival settings. This is surprising because spatial autocorrelation between units may influence their probability of being immune and the survival rate among the units at risk of experiencing the failure event simultaneously. Banerjee and Carlin (2004) develop Bayesian spatial cure models but focus on modeling spatial autocorrelation in the survival stage using a conditionally autoregressive (CAR) prior. In fact, these authors emphasize that future research must "include covariates and spatial random effects as regressors in the cure rate portion of the model, instead of just the log-relative risk portion" (274).

In line with these trends in the literature, some R packages fit standard and split-population survival models but do not allow for the incorporation of spatial information. The **survival** package (Therneau, 2020) fits parametric and semiparametric Cox survival models via MLE, whereas **dynsurv** (Wang et al., 2020) fits the Cox Proportional Hazards (PH) model with dynamic coefficients using MCMC methods. Conventional semi-parametric cure models can be estimated via MLE with the **smcure** (Cai et al., 2012) or **nltm** (Garibotti et al., 2019) package. The **flexsurvcure** (Amdahl, 2020) package fits parametric mixture and non-mixture cure models for time-to-event data, and the **spduration** package fits parametric SP survival models with time-varying covariates (Beger et al., 2018).

A small handful of packages allow the user to incorporate spatial information into their survival models, but *never* in split-population settings. The **BayesX** package (Umlauf et al., 2019) and its associated interface to R **R2BayesX** (Belitz et al., 2017) fit spatial survival models and structured additive regression models with spatial frailties. The **spBayesSurv** package (Zhou and Hanson, 2020) fits several Bayesian survival models with spatial frailties that can be formulated on a PH, proportional odds, or AFT scale, all of which can include time-varying covariates. The **spatsurv** package also fits Bayesian spatial survival models, including a PH model that permits users to incorporate Gaussian process frailties (Taylor and Rowlingson, 2020). Beyond R, WinBUGS and GeoBUGs code has been developed to fit, for instance, survival and non-mixture cure models with spatial frailties (Banerjee et al., 2003, 2004; Thomas et al., 2004).

To our knowledge, our **BayesSPsurv** package is the first to allow users to fit parametric split-population survival models with not just time-varying covariates but also spatial frailties in both stages. The frailties in the parametric Bayesian spatial SP model are estimated using the CAR prior approach (Besag et al., 1991; Bernardinelli et al., 1995; Banerjee et al., 2003; Banerjee and Carlin, 2004). **BayesSPsurv** also includes functions and routines coded in C++ to fit nonspatial parametric SP survival models with exchangeable frailties in the model's split and survival-stage equation and without any frailties. Statistical inference of the models in **BayesSPsurv** is conducted via combined MCMC techniques that require little input from users. Our package and supplemental code also provide functions to implement spatial autocorrelation tests, produce country-level adjacency matrices, generate and compare multiple Markov chains, assess convergence, and conduct model comparison. Before outlining the functionality of the package in greater detail, we turn to briefly develop each of the three included Bayesian split-population survival models.

The Bayesian (*Spatial*) split-population survival model

Model development

Define $i = \{1, 2, ...N\}$ for the units that may fail or experience an event of interest in a continuous-time survival dataset. Let $f(t)$ and $F(t)$ represent the probability density function and cumulative distribution function. The survival distribution is $S(t) = 1 - F(t)$, and the hazard rate is $h(t) = \frac{f(t)}{S(t)}$. Some units will fail during the time period under observation ($\widetilde{C}_i = 1$), while others do not and are "censored" ($\widetilde{C}_i = 0$). The general likelihood of the conventional survival model in which all units eventually experience the event of interest is

$$\prod_{i=1}^{N} \left[f\left(t_i\right)\right]^{\widetilde{C}_i} \left[S\left(t\right)\right]^{1-\widetilde{C}_i}. \tag{1}$$

Suppose that the survival data includes two subpopulations: an "at-risk " fraction that can fail and an "immune" fraction that will *not* experience the (failure) event of interest (Maller and Zhou, 1996; Yin and Ibrahim, 2005; Beger et al., 2017). When presented with this data generation process, researchers typically employ split-population survival (cure) models with or without unit-specific frailties to simultaneously estimate the probability of observations being in the immune fraction and the effect of covariates on the hazard of survival among the at-risk fraction (Maller and Zhou, 1996; Lu, 2010; Peng and Taylor, 2014).

To understand these models in more detail, consider the split-population survival model for the duration t that splits the sample into an at-risk and an immune fraction. Let $\alpha_i = Pr(Y_i = 1)$ be the probability with which units enter the immune fraction. α_i can be estimated via a binary response function and is defined for the logit case as:

$$\alpha_i = \frac{\exp\left(\mathbf{Z}_i\gamma + V_i\right)}{1 + \exp\left(\mathbf{Z}_i\gamma + V_i\right)}, \tag{2}$$

where $\mathbf{Z}_i$ are p2-dimensional covariates, γ the parameter vector in $\mathbb{R}^{p_2}$, and $V_i \sim N(0, \sigma^2)$ are the nonspatial i.i.d unit-specific frailties (random effects). Equation 2 is the split-population model's split-stage equation, where the unit-specific frailties V_i, which are each independent of other individual random effects, account for unobserved heterogeneity that influences probability α_i. Let $W_i \sim N(0, \sigma^2)$ denote the nonspatial i.i.d unit-specific frailties that capture the possibility that some units are at different risks of experiencing the event of interest due to unobserved factors. The proportional hazards function of the SP survival model with nonspatial, unit-specific frailties is

$$h\left(t_i|\mathbf{X}_i\boldsymbol{\beta}, W_i\right) = h_0\left(t_i\right)\omega_i \exp\left(\mathbf{X}_i\boldsymbol{\beta}\right) = h_0\left(t_i\right)\exp\left(\mathbf{X}_i\boldsymbol{\beta} + W_i\right), \tag{3}$$

where $h_0(t_i)$ is the baseline hazard (e.g., Weibull, log-logistic), $\log \omega_i = W_i$, $\mathbf{X}_i$ is the p1-dimensional covariates, and $\boldsymbol{\beta}$ the parameter vector in $\mathbb{R}^{p_1}$. We focus on incorporating unit-specific frailty terms generally because they are most commonly used in the social sciences. However, our approach could plausibly be extended to a shared frailty framework if the researcher believes that the frailties occur in clusters such that within-cluster frailties are correlated while frailties between clusters are independent.

Suppose, however, that the survival data with the two aforementioned subpopulations should be fit with time-varying covariates. We can re-define this data with unique "entry time" duration as $t0$ and "exit time" as duration t for each period at which an observation is observed. Let $t0_{ij}$ denote unit i's elapsed time since inception until the beginning of time period j, t_{ij} the elapsed time since that unit's inception until the end of period j, and $\tilde{C}_{ij} = 1$ if that unit fails or is censored ($\tilde{C}_{ij} = 0$) at t_{ij}. The probability of survival up until period j is now $S(t0) = 1 - F(t0)$ where $F\left(t0\right) = \int_0^{t0} f\left(t0\right)$. In this case, both subpopulations contribute to the log-likelihood of the split-population survival model with nonspatial i.i.d frailties as:

$$\ln L = \sum_{i=1}^{N} \left\{ \widetilde{C}_{ij} \ln \left[\left(1 - \alpha_{ij}\right) \frac{f\left(t_{ij}|\mathbf{X}_{ij}\boldsymbol{\beta}, W_i\right)}{S\left(t0_{ij}|\mathbf{X}_{ij}\boldsymbol{\beta}, W_i\right)} \right] + \left(1 - \widetilde{C}_{ij}\right) \ln \left[\alpha_i + \left(1 - \alpha_i\right) \frac{S\left(t_{ij}|\mathbf{X}_{ij}\boldsymbol{\beta}, W_i\right)}{S\left(t0_{ij}|\mathbf{X}_{ij}\boldsymbol{\beta}, W_i\right)} \right] \right\}, \tag{4}$$

where the "split-stage" equation is $\alpha_{ij} = \frac{\exp\left(\mathbf{Z}_{ij}\gamma + V_i\right)}{1+\exp\left(\mathbf{Z}_{ij}\gamma + V_i\right)}$. V_i and W_i are the nonspatial frailties. The model's survival stage estimates the probability of survival prior to the event of interest conditional upon being at-risk for that event given covariates $\mathbf{X}_{ij}$ and the baseline hazard function. If $V_i = W_i = 0$, then (4) defines the log-likelihood of the "Pooled" SP survival model (*without* unit-specific frailties)

with time-varying covariates (Ibrahim et al., 2001; Lu, 2010). However, if unobserved unit-specific heterogeneity influences the probability of immunity or survival time, it can be accounted for with the split and survival-stage frailty terms (V_i and W_i). In a Bayesian split-population survival framework, these frailties are incorporated into each stage of the model using the exchangeable normal prior,

$$W_i \sim N\left(0, 1/\tau\right) \text{ and } V_i \sim N\left(0, 1/\tau\right) , \tag{5}$$

with τ as the precision parameter (Banerjee et al., 2003; Banerjee and Carlin, 2004). The prior in (5) is induced by treating each specified unit as exchangeable rather than assigning weights corresponding to each unit's spatial relationship to one another (Bernardinelli and Montomoli, 1992; Darmofal, 2009). This Exchangeable split-population survival model is appropriate if each unit's frailty is presumed to be independent of other individual random effects. Geographically, for instance, this means that the influence of each unit-specific frailty on that unit's probability of being immune or its risk propensity is completely unrelated to the neighboring units' frailties unobserved effects.

Suppose, however, that independence among the frailties *cannot* be assumed—that is, that the frailties exhibit spatial autocorrelation or clustering that influences each units' propensity for being immune to an event of interest and their survival time if they are not immune. In a Bayesian split-population survival model, the assumption of spatial independence is relaxed by assigning spatial weights to the unit-specific frailties in the model's split and survival stage and then statistically incorporating these spatially weighted frailties via the conditionally autoregressive (CAR) prior approach (Besag et al., 1991; Banerjee et al., 2003). The CAR prior accounts for spatially autocorrelated frailties by allowing the frailties to be spatially autocorrelated across geographically adjacent units.

To understand how the CAR prior is applied, first note that spatial data often take the form of a lattice in which a continuous spatial surface is divided into a grid of units such as counties, districts, or countries. The spatially weighted frailties are constructed by defining the relevant spatial relationship among adjacent units (this could, for example, be geographic distance or contiguity) in an adjacency matrix $\mathbf{A}$ with elements $a_{ii'}$. Each element $a_{ii'}$ in $\mathbf{A}$ is given a weight of 1 if units i and i' are "neighbors," and 0 if they are not. Once these spatial weights are defined via the matrix $\mathbf{A}$, this information is then incorporated into the CAR prior, which permits us to model spatially dependent frailties between these contiguous units. To employ the CAR prior approach in a Bayesian SP survival framework, the frailties V_i are collected into the vector $\mathbf{V} = \{V_1, ..., V_N\}$, and W_i into $\mathbf{W} = \{W_1,....,W_N\}$. Separate CAR priors are then employed for $\mathbf{V}$ and $\mathbf{W}$, which implies the following model structure:

$$\mathbf{V}|\lambda \sim \text{CAR}\left(\lambda\right) \text{ and } \mathbf{W}|\lambda \sim \text{CAR}\left(\lambda\right). \tag{6}$$

λ is the precision parameter (Besag et al., 1991; Banerjee and Carlin, 2004). The CAR(λ) prior for $\mathbf{V}$ and $\mathbf{W}$ has a joint distribution in each case that has been formally characterized by scholars (Banerjee et al., 2003, 126).

The resulting conditional distributions of the spatial frailties for $\mathbf{V}$ and $\mathbf{W}$ are

$$V_i|V_{i'\neq i} \sim N\left(\overline{V_i}, 1/\left(\lambda m_i\right)\right), \quad W_i|W_{i'\neq i} \sim N\left(\overline{W_i}, 1/\left(\lambda m_i\right)\right). \tag{7}$$

$\overline{W_i} = m_i^{-1}\sum_{i'\ adj\ i} W_{i'}$ and $\overline{V_i} = m_i^{-1}\sum_{i'\ adj\ i} V_{i'}$. $\overline{W}_i$ and $\overline{V_i}$ are the averages of the neighboring $W_{i'\neq i}$ and $V_{i'\neq i}$, respectively, where $i'\ adj\ i$ denotes that i' is adjacent to i given $\mathbf{A}$, and m_i is the number of these adjacencies (Bernardinelli and Montomoli, 1992, 989; Thomas et al., 2004; Banerjee et al., 2003). Incorporating the spatial information in $\mathbf{A}$ in this way accounts for the possibility that spatially proximate units share common unmodeled factors that influence their probability of being immune or their survival time before experiencing the event of interest. Using this CAR prior approach to address spatial autocorrelation, the Spatial split-population survival model's log-likelihood is defined by substituting $\mathbf{V} = \{V_i\}$ and $\mathbf{W} = \{W_i\}$ in equation 4 (where $\alpha_{ij} = \frac{\exp(\mathbf{Z}_{ij}\gamma, \mathbf{V})}{1+\exp(\mathbf{Z}_{ij}\gamma, \mathbf{V})}$ is the split-stage equation).

The log-likelihood of the Pooled (non-frailty), Exchangeable (nonspatial frailty), and Spatial split-population (SP) survival models are compatible with any survival distribution. The **BayesSPsurv** package, however, supports MCMC estimation of these models for the Weibull and log-logistic distributions. Our empirical application below focuses on the Weibull survival distribution. The density, survival, and hazard rate in the Weibull case are

$$\begin{aligned} f\left(t_{ij}|\rho,\theta\right) &= \rho\theta\left(\theta t_{ij}\right)^{\rho-1}\exp\left(-\left(\theta t_{ij}\right)^{\rho}\right) \\ S\left(t_{ij}|\rho,\theta\right) &= \exp\left(-\left(\theta t_{ij}\right)^{\rho}\right) \text{ and } h\left(t_{ij}|\rho,\theta\right) = \rho\theta\left(\theta t_{ij}\right)^{\rho-1}, \end{aligned} \tag{8}$$

where $\theta = \exp\left(\mathbf{X}_{ij}\boldsymbol{\beta}, \mathbf{W}\right)$ for the Spatial, $\theta = \exp\left(\mathbf{X}_{ij}\boldsymbol{\beta} + W_i\right)$ for the Exchangeable, and $\theta = \exp\left(\mathbf{X}_{ij}\boldsymbol{\beta}\right)$ for the Pooled split-population Weibull model. The density, survival function and the hazard rate for the log-logistic case is defined in Bolte et al. (2021a).

Bayesian inference

Following standard practice for Bayesian inference (Carlin and Louis, 2000), we assign the multivariate normal (MVN) prior to $\boldsymbol{\beta} = \{\beta_1, ..., \beta_{p_1}\}$ and $\boldsymbol{\gamma} = \{\gamma_1, ..., \gamma_{p_2}\}$, and the Gamma prior for ρ with shape and scale parameters a_ρ and b_ρ for *each* of the three Bayesian split-population survival models in the **BayesSPsurv** package.

$$\begin{aligned} \rho \sim \text{Gamma}\left(a_\rho, b_\rho\right), \quad \boldsymbol{\beta} \sim \text{MVN}_{p_1}\left(\mu_\beta, \Sigma_\beta\right), \quad \boldsymbol{\gamma} \sim \text{MVN}_{p_2}\left(\mu_\gamma, \Sigma_\gamma\right) \\ \Sigma_\beta \sim \text{IW}\left(S_\beta, \nu_\beta\right); \ \Sigma_\gamma \sim \text{IW}\left(S_\gamma, \nu_\gamma\right), \end{aligned} \tag{9}$$

where a_ρ, b_ρ, S_β, ν_β, S_γ, ν_γ are the hyperparameters. We use Bayesian hierarchical modeling to estimate Σ_β and Σ_γ employing the Inverse-Wishart (IW) distribution when using the MVN (a weakly informative) prior. For Bayesian MCMC estimation of the Spatial SP survival (Weibull) model, we assign the hyperprior $p(\lambda)$ to λ given the CAR prior approach. Specifically, we assign the Gamma hyperprior $\lambda \sim \text{Gamma}(a_\lambda, b_\lambda)$ for λ (Banerjee and Carlin, 2004; Darmofal, 2009).[1] To estimate the Exchangeable SP Weibull model, we assign the (multivariate) normal prior for the model's split and survival-stage frailties (V_i, W_i), and the priors defined in (9) for $\boldsymbol{\beta}$, $\boldsymbol{\gamma}$,and ρ. To identify the Exchangeable and Spatial SP models' intercepts, we impose the constraint that the frailties sum to zero ($\sum_i V_i = 0$ and $\sum_i W_i = 0$).

The joint posterior distribution of the Bayesian Spatial SP survival model with time-varying covariates—our main model of interest—is

$$\begin{aligned} \pi\left(\boldsymbol{\beta}, \boldsymbol{\gamma}, \rho, \mathbf{W}, \mathbf{V}, \lambda, \Sigma_\beta, \Sigma_\gamma | \mathbf{C}, \mathbf{X}, \mathbf{Z}, \mathbf{t}, \mathbf{t0}, \boldsymbol{\gamma}\right) \quad \propto \quad & L\left(\boldsymbol{\beta}, \boldsymbol{\gamma}, \rho, \mathbf{W}, \mathbf{V} | \mathbf{C}, \mathbf{X}, \mathbf{Z}, \mathbf{t}, \mathbf{t0}\right) \\ & \pi\left(\mathbf{W}|\lambda\right) \pi\left(\mathbf{V}|\lambda\right) \pi\left(\beta|\mu_\beta, \Sigma_\beta\right) \pi\left(\gamma|\mu_\gamma, \Sigma_\gamma\right) \pi\left(\rho\right) \pi\left(\lambda\right) \pi\left(\Sigma_\beta\right) \pi\left(\Sigma_\gamma\right), \end{aligned} \tag{10}$$

where $L\left(\boldsymbol{\beta}, \boldsymbol{\gamma}, \rho, \mathbf{W}, \mathbf{V} | \mathbf{C}, \mathbf{X}, \mathbf{Z}, \mathbf{t}, \mathbf{t0}\right)$ is defined in (4) with $\mathbf{V} = \{V_1, ..., V_N\}$ and $\mathbf{W} = \{W_1, ..., W_N\}$. The density, survival function, and hazard rate for this likelihood is given by the Weibull (or log-logistic) distribution in our R-package. $\pi\left(\mathbf{W}|\lambda\right)$ and $\pi\left(\mathbf{V}|\lambda\right)$ are defined via their respective conditional distributions in (7). $\pi\left(\beta|\mu_\beta, \Sigma_\beta\right)$, $\pi\left(\gamma|\mu_\gamma, \Sigma_\gamma\right)$, and $\pi\left(\rho\right)$ are from (9), $\pi\left(\Sigma_\beta\right)$ and $\pi\left(\Sigma_\gamma\right)$ are from (10). $\pi\left(\lambda\right)$ is the Gamma hyperprior for the Spatial SP survival model. From (10), we can define the joint posterior distribution of the time-varying (i) Exchangeable SP survival model by incorporating the frailties V_i and W_i (instead of $\mathbf{W}$, $\mathbf{V}$, and their CAR priors) given by (5) and (ii) Pooled SP survival model by excluding frailty terms.

The three split-population survival models in **BayesSPsurv** are each estimated with an MCMC algorithm for Bayesian inference. Because closed-forms for the posterior distributions of $\boldsymbol{\beta}$, $\boldsymbol{\gamma}$ and ρ are not available, these parameters in each model are updated in the MCMC algorithm via slice-sampling (with stepout and shrinkage) from their respective full conditional distribution. The closed form of the full conditional distributions (e.g., $\pi\left(\Sigma_\beta|\beta\right)$ and $\pi\left(\Sigma_\gamma|\gamma\right)$) and details about slice-sampling is provided in Bolte et al. (2021a).[2] Further, because the closed-forms for the posterior distributions of λ, $\mathbf{W}$, and $\mathbf{V}$ are not available for the Spatial split-population survival model, our MCMC algorithm incorporates Gibbs Sampling for estimating λ. Our MCMC update scheme then employs the Metropolis-Hastings algorithm for estimating $\mathbf{V}$ given λ and then uses the Metropolis-Hastings algorithm to estimate $\mathbf{W}$ given λ.[3] In the Exchangeable SP survival model, the nonspatial i.i.d frailties V_i and W_i are updated via Metropolis-Hastings, while the Pooled SP model excludes frailty terms. The MCMC update scheme to fit each model in **BayesSPsurv** are described in detail in Bolte et al. (2021a).

[1] We specify the vague prior $(a_\lambda, b_\lambda) = (0.001, 1/0.001) = (0.001, 1000)$ as done for ρ.

[2] For more information on slice-sampling, see Neal (2003).

[3] The closed-form of the full conditional distribution of $\pi(\lambda|\mathbf{W}, \mathbf{V})$ used for the MCMC update scheme is formally characterized in Bolte et al. (2021a).

Using the BayesSPsurv R package

Each of the three Bayesian SP survival models in **BayesSPsurv** incorporates a cure rate fraction and assumes that the time-to-event baseline hazard follows a Weibull or log-logistic distribution (which the user specifies). Users can also incorporate time-varying covariates in either stage. **BayesSPsurv** contains compiled C++ code using the package **Rcpp** (Eddelbuettel et al., 2020) to maximize computational efficiency when estimating the included Bayesian SP survival models. In addition to the pre-estimation spatial autocorrelation (Join Count and Global Moran's I) tests described below, the **BayesSPsurv** package also permits users to calculate the deviance information criterion (DIC) and log-likelihood statistics from the Spatial, Exchangeable, and Pooled SP survival models' MCMC output. The DIC is a measure of model fit that also penalizes the effective number of parameters. Like the Akaike Information Criterion, models with smaller values are preferable to those with larger values. Users can also conduct MCMC diagnostics with various extant packages designed to handle `mcmc` objects such as **coda** (Plummer et al., 2020).

Function	Description
`spatialSPsurv()`	Fits Bayesian Spatial SP survival model
`exchangeSPsurv()`	Fits Bayesian Exchangeable SP survival model
`pooledSPsurv()`	Fits Bayesian Pooled SP survival model
`plot_JoinCount()`	Implement and plot Join Count statistics
`plot_Moran.I()`	Implement and plot global Moran's I statistics
`spatial_SA()`	Generate spatial weights (adjacency) matrix
`SPstats()`	Calculate DIC and log-likelihood from fitted models

Table 1: Functions in the **BayesSPsurv** package.

Loading the package, dataset, and assessing spatial autocorrelation

To demonstrate the utility and various functions in **BayesSPsurv**, we use replication data from Walter's (2015) global study on post-civil war peace duration (denoted in the package as `Walter_2015_JCR`). Walter's (2015) panel data consist of 1,237 observations from 46 countries observed between 1962 and 2009. As discussed in Bolte et al. (2021a), her data are well-suited for SP survival analysis because they include two underlying populations: an "at-risk" fraction of countries in which civil wars can potentially recur, and an "immune" fraction of countries in which civil conflict recurrence is structurally improbable. The `Walter_2015_JCR` data are a subset of Walter's (2015) most important variables, including `lgdpl` for log per capita income, `unpko` for the presence of UN peacekeeping operations, the binary variable `victory`, coded as 1 when one side in the civil war wins the conflict militarily, and the dummy variable `comprehensive`, coded as 1 when the combatants sign a comprehensive peace agreement. The dataset also includes the binary indicator `renewed_war`, coded as 1 for the year in which a civil war recurs and 0 otherwise. This variable will serve as our failure event.

First, however, we load the **BayesSPsurv** package along with the dataset and then use the `add_duration()` function from the **spduration** package to add several variables that allow us to capture the survival characteristics of the data (Beger et al., 2017). `unitID` indicates the unique unit identifier (in this case, a unique id for each civil conflict), and `timeID` specifies the temporal variable.

```
library(BayesSPsurv)
data(Walter_2015_JCR)
walter <- spduration::add_duration(Walter_2015_JCR,"renewed_war",
                                   unitID = "id", tID = "year",
                                   freq = "year", ongoing = FALSE)
str(walter)
'data.frame':        1237 obs. of  21 variables:
 $ year         : num  2002 2003 2004 2005 2003 ...
 $ lastyear     : num  0 0 0 1 0 0 0 0 0 0 ...
 $ renewed_war  : num  0 0 0 1 0 0 0 0 0 0 ...
 $ fhcompor1    : num  -1.17 -1.17 -1.08 -1 -1.08 ...
 $ lgdpl        : num  5.75 6.29 6.36 6.4 8 ...
 ...
 $ failure      : num  0 0 0 1 0 0 0 0 0 0 ...
 $ ongoing      : num  0 0 0 0 0 0 0 0 0 0 ...
 $ end.spell    : num  0 0 0 1 0 0 0 0 0 0 ...
```

```
$ cured        : num  0 0 0 0 1 1 1 1 1 1 ...
$ atrisk       : num  1 1 1 1 0 0 0 0 0 0 ...
$ censor       : num  0 0 0 0 0 0 0 0 0 0 ...
$ duration     : num  1 2 3 4 1 2 3 4 5 6 ...
$ t.0          : num  0 1 2 3 0 1 2 3 4 5 ...
```

These new variables include `duration`, a cumulative count of the years of post-war peace survived, and the dummy variable `atrisk`, coded as 1 for all observations that eventually experience war recurrence in the sample period. Altogether the data include 77 post-civil war peace spells and 24 instances of civil war recurrence.

If the preferred frailty unit is at the country-level, users can take advantage of the `spatial_SA()` function in the **BayesSPsurv** package to generate their binary spatial weights matrix. In most cases, analysts define spatial clustering in terms of geographic proximity. This often requires the researcher to define some maximum distance threshold below which two units are considered "neighbors." `spatial_SA()` allows users to specify their own distance threshold. For this example, we use the `spatial_SA()` function to generate an adjacency matrix of countries in the data whereby "proximity" ($a_{ii'} = 1$) is defined as having capitals that are within 800 km of each other (and $a_{ii'} = 0$ otherwise). We simply specify the unique identifier for each country (`ccode`) and our distance threshold of 800 km. The `spatial_SA()` function will produce a $1 \times N$ vector with identifying information (e.g., country ID) for each observation that matches the rows and columns of the matrix. The result is a list object called `walter` that includes both the original data frame and the associated adjacency matrix.

```
walter <- BayesSPsurv::spatial_SA(data = walter, var_ccode = "ccode",
walter[[2]][1:6,1:6]
```

```
     42  90  92  93  135  155
42   0   0   0   0    0    0
90   0   0   1   1    0    0
92   0   1   0   1    0    0
93   0   1   1   0    0    0
135  0   0   0   0    0    0
155  0   0   0   0    0    0
```

Note that spatial adjacency between units does not need to be defined geographically; users can conceptualize "space" as any form of the dyadic relationship between units, but typically spatial clustering is substantively captured with some measure of geographic distance. Users can also create their own adjacency matrix from scratch to incorporate into the Bayesian estimation routine if their units of analysis are something other than countries.

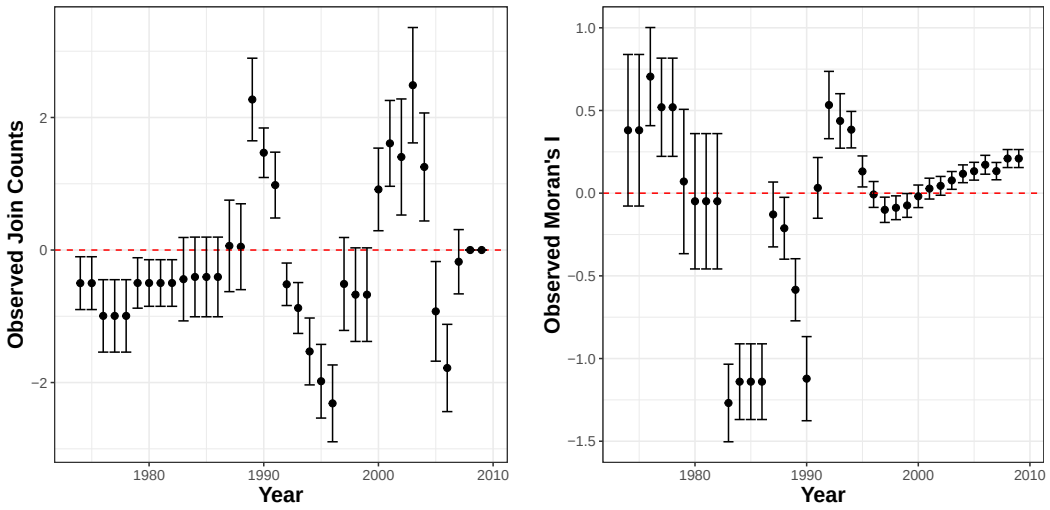
Having generated a matrix that records the spatial relationship of all pairs of units, we may now be interested in assessing the presence and degree of spatial autocorrelation in the data with respect to our outcomes of interest. The **BayesSPsurv** package provides two functions to conduct these preliminary tests on country-level data. The first is the `plot_JoinCount()` function, which generates an adjacency matrix with a user-defined distance threshold (as above), implements the join count test for each cross-section in the data, and then automatically plots the test statistics with user-specified confidence intervals across each observed year. The join count test is a widely used correlational statistic for evaluating whether the expected count of categorically distinct adjacent units is greater than what we would expect by chance alone (Cliff and Ord, 1981). More formally, if we assume two categories, Black and White, then the join count test statistic is

$$Z(BW) = \frac{BW - E(BW)}{\sqrt{\sigma^2_{BW}}}, \tag{11}$$

where BW and E(BW) are the observed and expected counts of adjacent Black and White units, respectively, and $\sigma^2_{BW} = E\left(BW^2\right) - E(BW)^2$. We use this test as a preliminary exercise to determine whether countries that *never* experience conflict recurrence in the sample exhibit spatial autocorrelation. Note that `plot_JoinCount` also has an argument specifying the minimum number of units that must be in a cross-section for the test statistic to be calculated. In our case, there were fewer than 12 countries in every year prior to 1975 in the data, and none were geographically proximate. By specifying `n=12`, we are simply dropping years prior to 1975. The following code produces the plot in Figure 1a.

```
plot_JoinCount(data = walter[[1]], var_cured = "cured", var_id = "ccode",
               var_time = "year", n = 12)
```

Negative values indicate clustering (positive spatial autocorrelation), and positive values indicate spatial dispersion. Figure 1a clearly depicts spatially correlated patterns of potential "peace consolidation,"



(a) Join Count **(b)** Moran's I

Figure 1: Pre-estimation Spatial Autocorrelation Tests

though the direction of the autocorrelation varies over time.

The second function, `plot_Moran.I()`, implements the Global Moran's I test, which assesses the direction and degree of spatial autocorrelation in continuous or ordinal data (Moran, 1950; Paradis et al., 2020). We use this test to examine whether the duration of post-civil war peace among geographically proximate countries exhibits spatial autocorrelation (or dispersion). Like the previous function, `plot_Moran.I()` plots the Moran's I values based on user-defined adjacencies and, in this case, 90% confidence intervals for each year.

```
plot_Moran.I(data = walter[[1]], var_duration = "duration", var_id = "ccode",
             var_time = "year", n = 12)
```

Figure 1b reveals that the spatial relationship of post-war peace survival oscillates between positive autocorrelation (in the 1970s, early 1990s, and late 2000s) and spatial dispersion (in much of the 1980s and late 1990s). Taken together, the Join Count and Moran's I tests suggest that unobserved heterogeneous risk factors that transcend the borders of a single state may lead to spatial autocorrelation in both the consolidation and duration of post-war peace in Walter's (2015) data. This suggests that a Spatial SP survival model is an appropriate method of analysis, particularly if proximate countries share common unobservable risk factors that affect their odds of being immune to war recurrence or the time it takes for renewed war to occur.

Applying the Bayesian Spatial SP survival model

The `spatialSPsurv()` function fits the Bayesian Spatial SP survival model, which includes spatially autocorrelated frailties in the model's split and survival stage and can incorporate time-varying covariates. We illustrate the features of the `spatialSPsurv()` function by fitting the Bayesian Spatial *Weibull* cure model on Walter's (2015) data on post-war peace.[4] The log-logistic results are reported in Bolte et al. (2021a). We include the `lgdpl` variable in the estimated model's split-stage equation, which estimates the probability of units being in the "consolidated" post-war peace group (the "immune" fraction), though it is important to note that `atrisk` is used here as the dependent variable. We estimate the probability of post-war survival among the at-risk fraction as a function of UN peacekeeping (`unpko`), outright military victory in the previous war (`victory`), the resolution of the previous war via peace agreement (`comprehensive`), and, again, GDP/capita (`lgdpl`). The spatial relationship of each country is incorporated into the model via the spatial weights matrix generated from the `spatial_SA()` function described earlier.

The Spatial SP survival model is fit via Bayesian methods for inference. The MVN prior is used for the model's split-stage (γ) and survival-stage (β) parameters (with the following hyperparameters $s_\beta = I_{p1}$, $v_\beta = p1$, $s_\gamma = I_{p2}$, $v_\gamma = p2$). The Gamma prior is used for the hazard's shape parameter ρ (with hyperparameters $a\rho = 1$, $b\rho = 1$), and separate CAR priors—with the Gamma hyperprior assigned to λ for CAR (λ)[5] —incorporate the spatially autocorrelated frailties in the model's split-stage

[4]The results for all models were calculated in **R** version 4.0.3. on a PC with a 6-core i7 processor.
[5]With vague prior $(a_\lambda, b_\lambda) = (0.001, 1/0.001) = (0.001, 1000)$ as done for ρ

(V) and survival stage (W). Estimation proceeds via the MCMC algorithm described in the previous section. To this end, we specify the MCMC sampler as follows: the argument `N = 15,000` sets the number of MCMC iterations to 15,000, `burn = 5,000` discards the first 5,000 states of the Markov chain, `thin = 15` specifies the number of steps that are employed to prevent autocorrelation, and `m = 10` limits the steps in the slice sampling to 10. The argument `w = c(1,1,1)` specifies the default values of the size of the slice sampling for γ, β, and ρ.

For simplicity, we retain the default 0 as the initial value for all parameters with the `ini.beta`, `ini.gamma`, `ini.W`, and `ini.V` arguments. We incorporate separate proposal variances, namely `prop.varV = 1e-05` and `prop.varW = 1e-05`, for the split and survival stage frailties (**V**, **W**) and then assign separate Metropolis-Hastings proposal steps for estimating these parameters to optimize the acceptance rates. To specify the covariates in the model, `duration ~` precedes the survival-stage covariates, while `immune ~` precedes the split-stage variables. `Y0` indicates the time elapsed since inception until time t - 1, while `LY` is a dummy that captures the last observation year due to censoring or failure. `A = walter[[2]]` calls the spatial weights matrix from our list object. `S= 'sp_id'` (generated by `spatial_SA()`) gives the unit identifier in the spatial weights matrix that matches the units in the data frame. The argument `form` is used to specify the parametric survival distribution (which can be either "Weibull" or "loglog").

Importantly, the model functions in the **BayesSPsurv** package generate single chains for each parameter, but we provide a routine for estimating multiple chains in parallel on the GitHub repository for the package (https://github.com/Nicolas-Schmidt/BayesSPsurv). We first discuss the results and diagnostics for the single-chain results using the above MCMC specification and then illustrate the multiple-chain results and diagnostics for the Spatial SP survival model.

```
set.seed(123456)
model <-  spatialSPsurv(duration = duration ~ victory + comprehensive + lgdpl + unpko ,
                            immune = atrisk ~ lgdpl,
                            Y0 = 't.0',
                            LY = 'lastyear',
                            S = 'sp_id' ,
                            data = walter[[1]],
                            N = 15000,
                            burn = 5000,
                            thin = 15,
                            w = c(1,1,1),
                            m = 10,
                            ini.beta =  0,
                            ini.gamma = 0,
                            ini.W = 0,
                            ini.V= 0,
                            form = "Weibull",
                            prop.varV = 1e-05,
                            prop.varW = 1e-05,
                            A = walter[[2]])
```

The function automatically provides a progress bar for users to track the percent of the computation that has been completed. The elapsed run time for the Spatial SP Weibull specification being examined here was just under 23 minutes (calculated with `system.time()`). Once completed, the generic `print()` function displays the results[6]:

```
print(model)

Call:
spatialSPsurv(duration = duration ~ victory + comprehensive +
    lgdpl + unpko, immune = atrisk ~ lgdpl, Y0 = "t.0",
    LY = "lastyear", S = "sp_id", A = walter[[2]],
    data = walter[[1]], N = 15000, burn = 5000, thin = 15, w = c(1,
        1, 1), m = 10, ini.beta = 0, ini.gamma = 0, ini.W = 0,
    ini.V = 0, form = "Weibull", prop.varV = 1e-05, prop.varW = 1e-05)


Iterations = 1:666
Thinning interval = 1
Number of chains = 1
```

[6] 95% Bayesian Credible Intervals are reported in Bolte et al. (2021a).

```
Sample size per chain = 666

Empirical mean and standard deviation for each variable,
plus standard error of the mean:


Duration equation:
                   Mean        SD    Naive SE Time-series SE
(Intercept)   0.3434823 1.0681878 0.041391437    0.078812901
victory       0.1028612 0.5321456 0.020620224    0.020620224
comprehensive 0.2036460 0.6178274 0.023940326    0.023940326
lgdpl         0.4831914 0.1424928 0.005521485    0.009230762
unpko         0.3057078 0.7695278 0.029818596    0.029818596

Immune equation:
                  Mean       SD  Naive SE Time-series SE
(Intercept) -0.4306327 4.545655 0.1761405      0.3648687
lgdpl       -2.4722920 3.113144 0.1206319      0.1646276
```

The posterior mean estimates suggest that peace agreements, military victory, and peacekeeping units may increase the survival of post-civil war peace, though none of these parameters are highly reliable given their 95% BCIs, as reported in (Bolte et al., 2021a). However, `lgdpl` appears to reliably *increase* the survival of peace among at-risk cases while also reliably *decreasing* the probability of peace "consolidation" in the split-stage. Calling the `SPstats()` function calculates the Deviance Information Criterion (DIC) and log-likelihood (LL) statistics from the estimated model, where $DIC = -2 \times (L - P)$, L is the log-likelihood of the data given the posterior means of the covariate parameters, and P is the effective number of parameters in the model.

```
SPstats(model)

$DIC
[1] -1726.128

$Loglik
[1] 5301.714
```

The `spatialSPsurv()` function produces an object of class `"mcmc"`, which is compatible with all standard summary and diagnostic methods for the single Markov chains in the **coda** package (Plummer et al., 2020). For instance, we can test for convergence with both the Geweke (1992) test using the `geweke.diag()` function and the Heidelberger and Welch (1983) stationarity test via the `heidel.diag()` function in **coda**. We report the Heidelberger and Welch (1983) stationarity test results in Bolte et al. (2021a) to save space. In this case, the Geweke tests indicate little evidence against convergence for each split-stage and survival-stage covariate, because the test statistics are reasonably close to zero.

```
geweke.diag(model$gammas)

Fraction in 1st window = 0.1
Fraction in 2nd window = 0.5

(Intercept)       lgdpl
    -1.3755      0.4429

geweke.diag(model$betas)

Fraction in 1st window = 0.1
Fraction in 2nd window = 0.5

  (Intercept)       victory comprehensive         lgdpl         unpko
      -0.8248       -0.4271        1.8237        0.3601       -0.9471

geweke.diag(model$rho)

Fraction in 1st window = 0.1
```

```
Fraction in 2nd window = 0.5

  var1
0.3544
```

coda's `plot()` function can then be used to display trace and posterior density plots for the single Markov chains of the model's split-stage γ, survival-stage β, and ρ parameters (which we do not report here to save space). Users may also wish to validate MCMC convergence using multiple chains with different initial values for each parameter rather than only examining single-chain diagnostics. A routine for generating multiple chains in parallel using the **doParallel** (Wallig et al., 2020) and **doRNG** (Gaujoux, 2020) packages is available on the **BayesSPsurv** GitHub repository, though single chains with different starting values can also be run sequentially. The user can then collect them into a single `mcmc.list` object and easily plot the trace and density plots of the multiple chains. Using the code on GitHub, the Spatial SP survival model was re-run four times (to generate four chains) with 0, 1, 10, and 50 as the initial starting values for each of the estimated parameters (β, γ, $\mathbf{W}$, and $\mathbf{V}$). The trace and density plots for the multiple chains are reported below (we excluded the β intercept for aesthetic reasons), and the total run time was approximately 28 minutes for all four chains to be generated simultaneously.

```
par(mfrow=c(2,2))
plot(gammas, density = FALSE, auto.layout = FALSE, col =c (28,26,27,29))
plot(gammas, trace = FALSE, auto.layout = FALSE)

par(mfrow=c(2,4))
plot(betas[,2:5], density= FALSE, auto.layout = FALSE, col = c(28,26,27,29))
plot(betas[,2:5], trace= FALSE, auto.layout = FALSE)
```

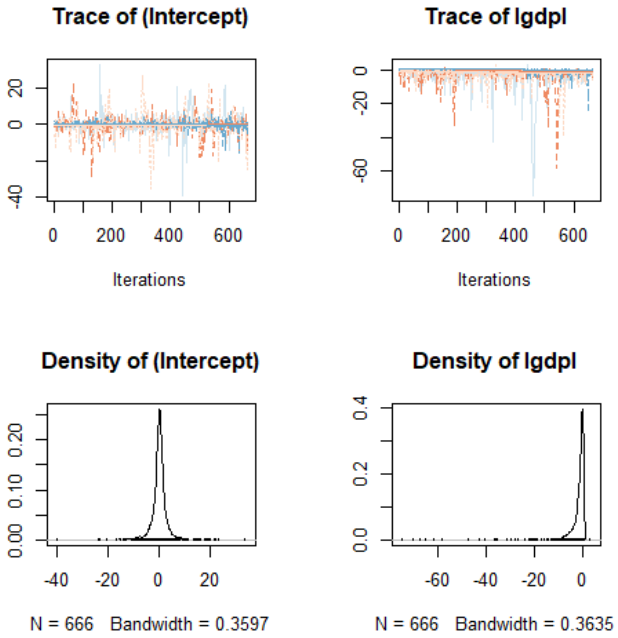


Figure 2: Posterior Densities of Bayesian Spatial Split-stage Parameters (γ)

The trace plots for the Spatial SP Weibull model's β and γ reveal decent mixing between the first three chains, though the fourth chain appears to take longer to converge. This is unsurprising given its extreme initial value. We can assess the convergence of the multiple chains using the Gelman-Rubin diagnostic, which compares the within-chain variance to the between-chain variance (Gelman and Rubin, 1992). If the difference in these variances is large, then the multiple chains likely have not converged to a proper stationary distribution. We can easily calculate the potential scale reduction factor (PSRF) with the `gelman.diag()` function in **coda**, which should be approximately 1 if all chains have converged to a common distribution.

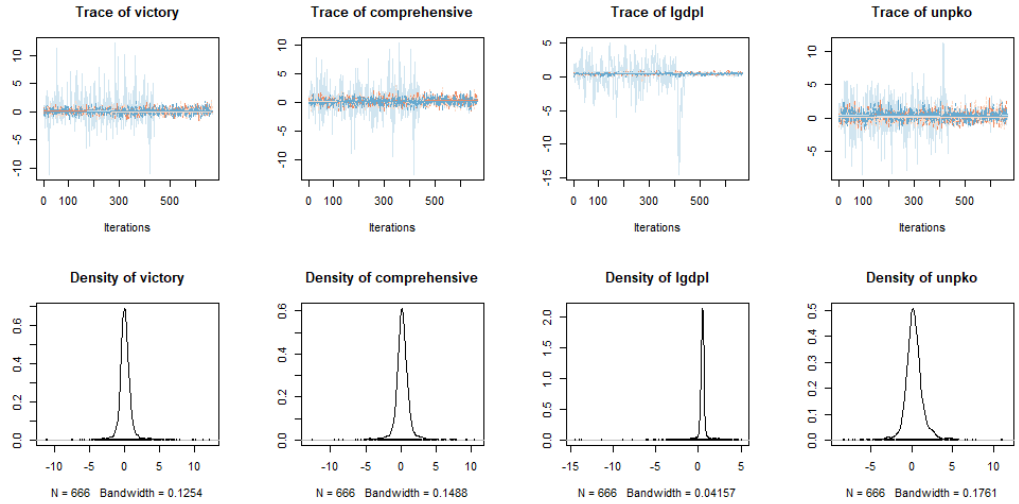


Figure 3: Posterior Densities of Bayesian Spatial Survival-stage Parameters (β)

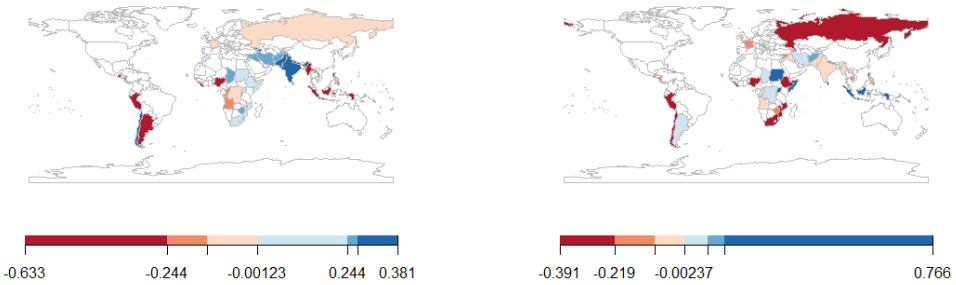
```
Potential scale reduction factors:

              Point est. Upper C.I.
(Intercept)         1.19       1.38
victory             1.17       1.20
comprehensive       1.17       1.19
lgdpl               1.32       2.40
unpko               1.11       1.14

Multivariate psrf

1.05

Potential scale reduction factors:

            Point est. Upper C.I.
(Intercept)       1.05       1.07
lgdpl             1.14       1.33

Multivariate psrf

1.05
```

The multivariate PSRF is less than 1.1 in both stages, but the point estimates for all parameters are still insufficiently far from 1, suggesting that extending the chains would likely improve the accuracy of the estimates.

One way to substantively interpret the spatial frailties is to display a map and determine whether adjacent units share similar frailty values (e.g., Darmofal, 2009). The following code from the **countrycode** (Arel-Bundock, 2020) and **rworldmap** (South, 2016) packages permits users to create choropleth maps of the spatial frailty posterior means. Figures 4a-4b display the single-chain split-stage (`V`) and survival-stage (`W`) frailties from our Spatial SP Weibull model (the code generates Figure 4a; Figure 4b simply uses `W` in place of `V`.

```
library(rworldmap)
library(countrycode)
library(classInt)

spv  <- matrix(apply(model$V, 2, mean), ncol = 1, nrow = ncol(model$V))
ISO3 <- countrycode(colnames(model$V) ,'gwn','iso3c')
spv  <- data.frame(ccode = colnames(model$V), ISO3 = ISO3, spv = spv)
map  <- joinCountryData2Map(spv, joinCode = "ISO3", nameJoinColumn = "ISO3")
```

```
classInt <- classIntervals(map[["spv"]], n = 6, style = "quantile")
mapParams <- mapCountryData(map,
                            nameColumnToPlot = "spv",
                            addLegend = FALSE,
                            catMethod = classInt[["brks"]],
                            colourPalette = palette(RColorBrewer::brewer.pal(6,"RdBu")),
                            mapTitle = "")

do.call(addMapLegend, c(mapParams, legendLabels = "all", legendWidth = 0.5,
                                   legendIntervals = "data", legendMar = 2))
```



(a) Split-stage (**V**) **(b)** Survival-stage (**W**)

Figure 4: Spatial Frailty Posterior Means

Note that in certain regions, there are distinct spatial bands in the (i) split-stage frailties, which range from -0.633 to 0.381 with a corresponding standard deviation of 0.34 and (ii) survival-stage frailties that range from -0.391 to 0.766 with a corresponding standard deviation of 0.32. These spatial bands reveal geographic clustering in both consolidation and duration of post-war peace since similar frailty values often occur near one another.

Applying the Bayesian Exchangeable and Pooled SP survival models

The **BayesSPsurv** package can also be used to estimate Bayesian Exchangeable and Pooled SP survival models. The Exchangeable model incorporates frailties that are assumed to be statistically independent in both stages, while the Pooled model excludes frailties altogether. We again use Walter's (2015) data on post-civil war peace to illustrate these models. Beginning with the Exchangeable Weibull SP survival model, we again assign the MVN prior to the model's γ and β parameters, the Gamma prior to ρ, and the same default settings for the hyperparameters ($s_\beta = I_{p1}$, $v_\beta = p1$, $s_\gamma = I_{p2}$, $v_\gamma = p2$, $a_\rho = 1$, $b_\rho = 1$). We also incorporate separate proposal variance variables for the nonspatial i.i.d frailties (V_i,W_i) and assign separate Metropolis-Hastings steps to estimate these parameters. Further, we use almost the same MCMC sampler specification as before. However, rather than incorporating spatial information via an adjacency matrix, we let `id_WV=country` to define the country-level nonspatial frailties.

```
set.seed(123456)
country  <- countrycode(unique(walter[[1]]$ccode),'gwn','iso3c')
model1 <- exchangeSPsurv(duration = duration ~ victory + comprehensive + lgdpl + unpko,
                         immune = atrisk ~ lgdpl,
                         Y0 = 't.0',
                         LY = 'lastyear',
                         S = 'sp_id' ,
                         data = walter[[1]],
                         N = 15000,
                         burn = 5000,
                         thin = 15,
                         w = c(1,1,1),
                         m = 10,
                         ini.beta =  0,
                         ini.gamma = 0,
```

```
                        ini.W = 0,
                        ini.V= 0,
                        form = "Weibull",
                        prop.varV = 1e-05,
                        prop.varW = 1e-05,
                        id_WV=country)
```

Again, users can assess either single or multiple (parallel) Markov chains with different starting parameter values when fitting the Exchangeable SP survival model. We report the single-chain results here for simplicity. The run time for the above specification was approximately 16 minutes and 30 seconds. The single-chain results and model fit statistics are obtained with the `print()` and `SPstats()` functions.

```
print(model1)

Call:
exchangeSPsurv(duration = duration ~ victory + comprehensive +
    lgdpl + unpko, immune = atrisk ~ lgdpl, Y0 = "t.0",
    LY = "lastyear", S = "sp_id", data = walter[[1]],
    N = 15000, burn = 5000, thin = 15, w = c(1, 1, 1), m = 10,
    ini.beta = 0, ini.gamma = 0, ini.W = 0, ini.V = 0, form = "Weibull",
    prop.varV = 1e-05, prop.varW = 1e-05, id_WV = country)


Iterations = 1:666
Thinning interval = 1
Number of chains = 1
Sample size per chain = 666

Empirical mean and standard deviation for each variable,
plus standard error of the mean:


Duration equation:
                    Mean        SD    Naive SE Time-series SE
(Intercept)   0.31054460 1.0938158 0.042384501     0.08335381
victory       0.14520456 0.5328378 0.020647044     0.02064704
comprehensive 0.09721372 0.6374526 0.024700786     0.02470079
lgdpl         0.48473536 0.1455731 0.005640844     0.01061988
unpko         0.44731384 0.7421925 0.028759376     0.02875938

Immune equation:
                  Mean       SD  Naive SE Time-series SE
(Intercept) -0.8993176 5.924550 0.2295716      0.4969172
lgdpl       -3.7286951 5.846367 0.2265421      0.5271203

SPstats(model1)
$DIC
[1] 3549.075

$Loglik
[1] 7506.608
```

Based on these results, `lgdpl` is still a highly reliable predictor of longer peace duration among conflicts at risk of recurrence, while the other survival stage covariates remain positive but statistically unreliable. In contrast to the spatial frailty model, however, the effect of `lgdpl` on the probability of peace consolidation is now statistically unreliable. Users interested in displaying and interpreting the exchangeable frailties can do so in a variety of ways, but if these results are being compared to those of a spatial frailty model, we recommend mapping the exchangeable frailty values and comparing the clustering (or lack thereof) of the i.i.d. frailty terms to those produced from the spatial model (Darmofal, 2009).

Finally, the `pooledSPsurv()` function fits the Pooled Bayesian SP survival model. For this application, we again assign the MVN prior to the model's γ and β parameters, the Gamma prior to ρ, and the same default settings for the hyperparameters described earlier.

```
set.seed(123456)
model2 <- pooledSPsurv(duration = duration ~ victory + comprehensive + lgdpl + unpko,
                       immune = atrisk ~ lgdpl,
                       Y0 = 't.0',
                       LY = 'lastyear',
                       data = walter[[1]],
                       N = 15000,
                       burn = 5000,
                       thin = 15,
                       w = c(1,1,1),
                       m = 10,
                       ini.beta =  0,
                       ini.gamma = 0,
                       form = "Weibull")
```

Note that the MCMC sampler is almost identical to that of the previous two models, but we now *exclude* the separate Metropolis-Hastings proposal step for the frailities in the MCMC algorithm as well as the initial values for the frailty estimates since this is a non-frailty model. The run time for the Pooled model was approximately 10 minutes. The single-chain output is again easily displayed.

```
print(model2)

Call:
pooledSPsurv(duration = duration ~ victory + comprehensive +
    lgdpl + unpko, immune = atrisk ~ lgdpl, Y0 = "t.0",
    LY = "lastyear", data = walter[[1]], N = 15000, burn = 5000,
    thin = 15, w = c(1, 1, 1), m = 10, ini.beta = 0, ini.gamma = 0,
    form = "Weibull")


Iterations = 1:666
Thinning interval = 1
Number of chains = 1
Sample size per chain = 666

Empirical mean and standard deviation for each variable,
plus standard error of the mean:


Duration equation:
                   Mean        SD    Naive SE Time-series SE
(Intercept)   0.3014335 1.0572594 0.040967971     0.07197438
victory       0.1060453 0.5485730 0.021256770     0.02125677
comprehensive 0.1900127 0.6054421 0.023460406     0.02481878
lgdpl         0.4647260 0.1552465 0.006015679     0.01166616
unpko         0.2692275 0.7740804 0.029995006     0.02999501

Immune equation:
                 Mean       SD  Naive SE Time-series SE
(Intercept) -1.213902 9.617551 0.3726725      1.3195278
lgdpl       -2.896038 4.939374 0.1913968      0.5267751


SPstats(model2)

$DIC
[1] 1804.527

$Loglik
[1] 6229.248
```

For both the Exchangeable and Pooled SP survival models, users can again employ the code and routines available in our GitHub repository to view the trace plots from the single or multiple Markov chains associated with the posterior densities for the γ, β, and ρ parameters and conduct any relevant convergence diagnostics. The plots and convergence test statistics for the models

presented here are reported in Bolte et al. (2021a) to save space. Additional information is available at https://github.com/Nicolas-Schmidt/BayesSPsurv.

Conclusion

Survival data often include two populations with different underlying risk propensities: the immune fraction of right-censored units that will never experience the event of interest and the at-risk fraction of units that have or will. Numerous R-packages such as **smcure**, **nltm**, and **spduration** have been developed to fit parametric or semi-parametric cure models for nonspatial survival data, but none of these packages allow the user to account for spatial autocorrelation in both stages. A separate set of packages allows the user to include spatially weighted frailties in conventional survival models (Taylor and Rowlingson, 2017; Zhou et al., 2020), but spatial autocorrelation may also influence the probability of immunity from an event of interest. The **BayesSPsurv** package addresses this lacuna by offering a suite of functions to fit Bayesian SP survival models. Specifically, users can estimate Bayesian Pooled, Exchangeable, and Spatial frailty SP survival models with time-varying covariates, specify their own spatial weights matrices, and easily examine diagnostic statistics. The applied potential of the Spatial SP survival model, in particular, is considerable, as researchers studying anything from the survival of cancer patients to the survival of political regimes may have a methodological need to model spatial autocorrelation in the immune fraction.

Future work can build upon the spatial and nonspatial cure models presented here to develop estimation routines for survival data with multiple stages, competing risks, or recurrent events. Moreover, although **BayesSPsurv** is the first to allow spatially autocorrelated frailties in both stages, future work can extend the frailty models in our package by employing a similar approach as Yin (2005) to incorporate shared frailties with factor loadings to account for any correlation between the frailty terms (**W** and **V**) in either stage. Subsequent iterations of the package will include alternative baseline hazards based on penalized splines or nonparametric Gaussian processes as well as the option to use a semiparametric Cox model with exchangeable or spatial frailties. Implementing these future developments with an external MCMC sampling dependency built on C++ subroutines like STAN would be useful, given the Hamiltonian Monte Carlo algorithm's efficiency in sampling from high dimensional posterior distributions. We also plan to extend **BayesSPsurv** to accommodate left and interval censoring and delayed entry.

Bibliography

J. Amdahl. *flexsurvcure: Flexible Parametric Cure Models*, 2020. URL https://CRAN.R-project.org/package=flexsurvcure. R package version 1.2.0. [p596]

V. Arel-Bundock. *countrycode: Convert Country Names and Country Codes*, 2020. URL https://CRAN.R-project.org/package=countrycode. R package version 1.2.0. [p606]

S. Banerjee and B. P. Carlin. Parametric spatial cure rate models for interval-censored time-to-relapse data. *Biometrics*, 60(1):268–275, 2004. URL https://doi.org/10.1111/j.0006-341X.2004.00032.x. [p595, 596, 598, 599]

S. Banerjee, M. M. Wall, and B. P. Carlin. Frailty modeling for spatially correlated survival data, with application to infant mortality in minnesota. *Biostatistics*, 4(1):123–142, 2003. URL https://doi.org/10.1093/biostatistics/4.1.123. [p595, 596, 598]

S. Banerjee, A. Gelfand, J. R. Knight, and C. Sirmans. Spatial modeling of house prices using normalized distance-weighted sums of stationary processes. *Journal of Business & Economic Statistics*, 22(2): 206–213, 2004. URL https://doi.org/10.1198/073500104000000091. [p596]

A. Beger, D. W. Hill, N. W. Metternich, S. Minhas, and M. D. Ward. Splitting it up: the spduration split-population duration regression package for time-varying covariates. *The R Journal*, 9(2):474–486, 2017. URL https://doi.org/10.32614/RJ-2017-056. [p595, 596, 597, 600]

A. Beger, D. Chiba, D. W. Hill, Jr., N. W. Metternich, S. Minhas, and M. D. Ward. *spduration: Split-Population Duration (Cure) Regression*, 2018. URL https://CRAN.R-project.org/package=spduration. R package version 0.17.1. [p595, 596]

C. Belitz, A. Brezger, T. Kneib, S. Lang, and N. Umlauf. *BayesX: Software for Bayesian Inference in Structured Additive Regression Models*, 2017. URL http://www.BayesX.org/. Version 1.1. [p596]

L. Bernardinelli and C. Montomoli. Empirical bayes versus fully bayesian analysis of geographical variation in disease risk. *Statistics in Medicine*, 11(8):983–1007, 1992. URL http://dx.doi.org/10.1002/sim.4780110802. [p598]

L. Bernardinelli, D. Clayton, C. Pascutto, C. Montomoli, M. Ghislandi, and M. Songini. Bayesian analysis of space—time variation in disease risk. *Statistics in medicine*, 14(21-22):2433–2443, 1995. URL http://dx.doi.org/10.1002/sim.4780142112. [p596]

J. Besag, J. York, and A. Mollié. Bayesian image restoration, with two applications in spatial statistics. *Annals of the institute of statistical mathematics*, 43(1):1–20, 1991. URL https://doi.org/10.1007/BF00116466. [p596, 598]

B. Bolte, N. Huynh, B. Mukherjee, S. Béjar, and N. Schmidt. Bayesian split-population survival models for the social sciences (version 2). Available at SSRN, 2021a. URL https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3765112. [p599, 600, 602, 603, 604, 610]

B. Bolte, N. Schmidt, S. Bejar, B. Mukherjee, M. M. Joo, and N. K. Huynh. *BayesSPsurv: Bayesian Spatial Split Population Survival Model*, 2021b. URL https://CRAN.R-project.org/package=BayesSPsurv. R package version 0.1.3. [p595]

J. M. Box-Steffensmeier and B. S. Jones. *Event History Analysis*. Cambridge University Press, New York, 2004. [p595, 596]

J. M. Box-Steffensmeier and C. Zorn. Modeling heterogeneity in duration models. In *Summer Meeting of the Political Methodology Society, July15-17*, 1999. [p595, 596]

C. Cai, Y. Zou, Y. Peng, and J. Zhang. *smcure: Fit Semiparametric Mixture Cure Models*, 2012. URL https://CRAN.R-project.org/package=smcure. R package version 2.0. [p595, 596]

B. P. Carlin and T. A. Louis. *Bayes and Empirical Bayes Methods for Data Analysis, 2nd Edition*. New York: Chapman and Hall/CRC, 2000. [p599]

A. D. Cliff and J. K. Ord. *Spatial processes: models & applications*. Taylor & Francis, 1981. [p601]

D. Darmofal. Bayesian spatial survival models for political event processes. *American Journal of Political Science*, 53(1):241–257, 2009. URL https://doi.org/10.1111/j.1540-5907.2008.00368.x. [p595, 596, 598, 599, 606, 608]

U. Diva, D. K. Dey, and S. Banerjee. Parametric models for spatially correlated survival data for individuals with multiple cancers. *Statistics in medicine*, 27(12):2127–2144, 2008. URL https://doi.org/10.1002/sim.3141. [p596]

D. Eddelbuettel, R. Francois, J. Allaire, K. Ushey, Q. Kou, N. Russell, D. Bates, and J. Chambers. *Rcpp: Seamless R and C++ Integration*, 2020. URL https://CRAN.R-project.org/package=Rcpp. R package version 1.0.5. [p600]

G. Garibotti, A. Tsodikov, and M. Clements. *nltm: Non-Linear Transformation Models*, 2019. URL https://CRAN.R-project.org/package=nltm. R package version 1.4.2. [p596]

R. Gaujoux. *doRNG: Generic Reproducible Parallel Backend for 'foreach' Loops*, 2020. URL https://cran.r-project.org/web/packages/doRNG/index.html. R package version 1.8.2. [p605]

A. Gelman and D. B. Rubin. Inference from iterative simulation using multiple sequences. *Statistical Science*, 7(4):457–511, 1992. [p605]

J. F. Geweke. Evaluating the accuracy of sampling-based approaches to the calculation of posterior moments. In J. Bernardo, J. Berger, A. Dawid, and A. Smith, editors, *Bayesian Statistics*. Oxford: Clarendon Press, 1992. URL https://doi.org/10.21034/sr.148. [p604]

P. Heidelberger and P. D. Welch. Simulation run length control in the presence of an initial transient. *Operations Research*, 3(6):1109–1144, 1983. URL https://doi.org/10.1287/opre.31.6.1109. [p604]

R. Henderson, S. Shimakura, and D. Gorst. Modeling spatial variation in leukemia survival data. *Journal of the American Statistical Association*, 97(460):965–972, 2002. URL https://doi.org/10.1198/016214502388618753. [p595, 596]

J. G. Ibrahim, M.-H. Chen, and D. Sinha. Bayesian semiparametric models for survival data with a cure fraction. *Biometrics*, 57(2):383–388, 2001. URL http://dx.doi.org/10.1111/j.0006-341X.2001.00383.x. [p598]

Y. Li and L. Ryan. Modeling spatial survival data using semiparametric frailty models. *Biometrics*, 58 (2):287–297, 2002. URL https://doi.org/10.1111/j.0006-341X.2002.00287.x. [p595]

W. Lu. Efficient estimation for an accelerated failure time model with a cure fraction. *Statistica Sinica*, 20(2):661–674, 2010. [p597, 598]

R. A. Maller and X. Zhou. *Survival analysis with long-term survivors*. John Wiley & Sons, New York, 1996. [p595, 597]

P. A. Moran. Notes on continuous stochastic phenomena. *Biometrika*, 37(1/2):17–23, 1950. URL http://dx.doi.org/10.1093/biomet/37.1-2.17. [p602]

R. M. Neal. Slice sampling. *Annals of statistics*, pages 705–741, 2003. URL http://dx.doi.org/10.1214/aos/1056562461. [p599]

E. Paradis, S. Blomberg, B. Bolker, J. Brown, S. Claramunt, J. Claude, H. S. Cuong, R. Desper, G. Didier, B. Durand, J. Dutheil, R. Ewing, O. Gascuel, T. Guillerme, C. Heibl, A. Ives, B. Jones, F. Krah, D. Lawson, V. Lefort, P. Legendre, J. Lemon, G. Louvel, E. Marcon, R. McCloskey, J. Nylander, R. Opgen-Rhein, A.-A. Popescu, M. Royer-Carenzi, K. Schliep, K. Strimmer, and D. de Vienne. *ape: Analyses of Phylogenetics and Evolution*, 2020. URL https://CRAN.R-project.org/package=ape. R package version 5.4-1. [p602]

V. Patilea and I. Van Keilegom. A general approach for cure models in survival analysis. *Annals of Statistics*, 48(4):2323–2346, 2020. URL https://doi.org/10.1214/19-AOS1889. [p596]

Y. Peng and J. M. Taylor. Mixture cure model with random effects for the analysis of a multi-center tonsil cancer study. *Statistics in medicine*, 30(3):211–223, 2011. URL https://doi.org/10.1002/sim.4098. [p596]

Y. Peng and J. M. Taylor. Cure models. *Handbook of survival analysis*, pages 113–134, 2014. URL https://doi.org/10.1201/b16248-8. [p595, 597]

M. Plummer, N. Best, K. Cowles, K. Vines, D. Sarkar, D. Bates, R. Almond, and A. Magnusson. *coda: Output Analysis and Diagnostics for MCMC*, 2020. URL https://CRAN.R-project.org/package=coda. R package version 0.19-4. [p600, 604]

R. A. Ray, R. W. Perry, N. A. Som, and J. L. Bartholomew. Using cure models for analyzing the influence of pathogens on salmon survival. *Transactions of the American Fisheries Society*, 143(2):387–398, 2014. URL https://doi.org/10.1080/00028487.2013.862183. [p596]

P. Schmidt and A. D. Witte. Predicting criminal recidivism using 'split population'survival time models. *Journal of Econometrics*, 40(1):141–159, 1989. URL https://doi.org/10.1016/0304-4076(89)90034-1. [p596]

A. South. *rworldmap: Mapping Global Data*, 2016. URL https://CRAN.R-project.org/package=rworldmap. R package version 1.3-6. [p606]

B. Taylor and B. Rowlingson. spatsurv: an r package for bayesian inference with spatial survival models. *Journal of Statistical Software*, 77(4):1–32, 2017. URL https://doi.org/10.18637/jss.v077.i04. [p595, 596, 610]

B. M. Taylor and B. S. Rowlingson. *spatsurv: Bayesian Spatial Survival Analysis with Parametric Proportional Hazards Models*, 2020. URL https://CRAN.R-project.org/package=spatsurv. R package version 1.5. [p596]

T. M. Therneau. *A Package for Survival Analysis in R*, 2020. URL https://CRAN.R-project.org/package=survival. R package version 3.1-12. [p596]

A. Thomas, N. Best, D. Lunn, D. Arnold, and D. Spiegelhalter. Geobugs version 1.2 user manual. *MRC Biostatistics Unit*, 2004. URL https://www.mrc-bsu.cam.ac.uk/wp-content/uploads/geobugs12manual.pdf. [p596, 598]

N. Umlauf, T. Kneib, and N. Klein. *BayesX: R Utilities Accompanying the Software Package BayesX*, 2019. URL https://CRAN.R-project.org/package=BayesX. R package version 0.3-1. [p596]

M. Wallig, Microsoft Corporation, S. Weston, and D. Tenenbaum. *doParallel: Foreach Parallel Adaptor for the 'parallel' Package*, 2020. URL https://cran.r-project.org/web/packages/doParallel/index.html. R package version 1.0.16. [p605]

B. F. Walter. Why bad governance leads to repeat civil war. *Journal of Conflict Resolution*, 59(7):1242–1272, 2015. URL https://doi.org/10.1177/0022002714528006. [p600, 602, 607]

W. Wang, M.-H. Chen, X. Wang, and J. Yan. *dynsurv: Dynamic Models for Survival Data*, 2020. URL https://CRAN.R-project.org/package=dynsurv. R package version 0.4-2. [p596]

Y. Wang, J. G. Klijn, Y. Zhang, A. M. Sieuwerts, M. P. Look, F. Yang, D. Talantov, M. Timmermans, M. E. Meijer-van Gelder, J. Yu, et al. Gene-expression profiles to predict distant metastasis of lymph-node-negative primary breast cancer. *The Lancet*, 365(9460):671–679, 2005. URL https://doi.org/10.1016/S0140-6736(05)17947-1. [p596]

G. Yin. Bayesian cure rate frailty models with application to a root canal therapy study. *Biometrics*, 61 (2):552–558, 2005. [p610]

G. Yin and J. G. Ibrahim. Cure rate models: a unified approach. *Canadian Journal of Statistics*, 33(4): 559–570, 2005. URL http://dx.doi.org/10.1002/cjs.5550330407. [p597]

H. Zhou and T. Hanson. *spBayesSurv: Bayesian Modeling and Analysis of Spatially Correlated Survival Data*, 2020. URL https://CRAN.R-project.org/package=spBayesSurv. R package version 1.1.4. [p596]

H. Zhou, T. Hanson, and J. Zhang. spBayesSurv: Fitting Bayesian spatial survival models using R. *Journal of Statistical Software*, 92(9):1–33, 2020. URL http://dx.doi.org/10.18637/jss.v092.i09. [p595, 596, 610]

Brandon Bolte
Department of Political Science
Penn State University
USA
blb72@psu.edu

Nicolás Schmidt
Department of Political Science
Universidad de la República
Uruguay
nschmidt@cienciassociales.edu.uy

Sergio Béjar
Department of Political Science
San José State University
USA
sergio.bejar@sjsu.edu

Nguyen Huynh
Department of Political Science
Penn State University
USA
nkh8@psu.edu

Bumba Mukherjee
Department of Political Science
Penn State University
USA
sxm73@psu.edu

stratamatch: Prognostic Score Stratification Using a Pilot Design

by Rachael C. Aikens, Joseph Rigdon, Justin Lee, Michael Baiocchi, Andrew B. Goldstone, Peter Chiu, Y. Joseph Woo, and Jonathan H. Chen

Abstract Optimal propensity score matching has emerged as one of the most ubiquitous approaches for causal inference studies on observational data. However, outstanding critiques of the statistical properties of propensity score matching have cast doubt on the statistical efficiency of this technique, and the poor scalability of optimal matching to large data sets makes this approach inconvenient if not infeasible for sample sizes that are increasingly commonplace in modern observational data. The **stratamatch** package provides implementation support and diagnostics for 'stratified matching designs,' an approach that addresses both of these issues with optimal propensity score matching for large-sample observational studies. First, stratifying the data enables more computationally efficient matching of large data sets. Second, **stratamatch** implements a 'pilot design' approach in order to stratify by a prognostic score, which may increase the precision of the effect estimate and increase power in sensitivity analyses of unmeasured confounding.

Introduction

To make causal inference from observational data, researchers must address concerns that effect estimates may be biased due to confounding factors – baseline characteristics of the individuals in the study that influence both their selection of treatment and their probable outcome. Matching methods seek to account for this self-selection by grouping treated and control individuals with similar baseline characteristics. One of the most common such methods, propensity score matching, pairs individuals who appear to have had similar probabilities of receiving the treatment according to their baseline characteristics (Rosenbaum and Rubin, 1983), with the goal of coercing the data set into a form that resembles a fully-randomized controlled trial (King and Nielsen, 2019; Rosenbaum et al., 2010; Hernán and Robins, 2016). However, propensity score matching can only address bias due to *measured* baseline covariates, necessitating sensitivity analyses to interrogate the potential of bias due to *unmeasured* confounding (Rosenbaum, 2005b; Rosenbaum et al., 2010).

In their provocative article "Why Propensity Should Not Be Used for Matching," King and Nielson argue that the fully randomized controlled trial – the design emulated by propensity score matching – is less statistically efficient than the block-randomized controlled experiment (King and Nielsen, 2019). In block-randomized designs, individuals are stratified by prognostically important covariates (e.g., for a clinical trial: sex, age group, smoking status) *prior* to randomization in order to reduce the heterogeneity between the treatment and control groups. In the experimental context, these efforts to reduce heterogeneity between compared groups help to increase the precision of the treatment effect estimate. In observational settings, reducing this type of heterogeneity not only improves precision but increases the robustness of the study's conclusions to being explained away by the possibility of unobserved confounding (Rosenbaum, 2005a; Aikens et al., 2020). The stratified matching design – in which observations are stratified prior to matching within strata – attempts to emulate the block-randomized controlled trial design in the observational context in order to secure these statistical benefits over pure propensity score matching. In addition, since the computation required for optimal matching can be quite time-consuming for studies of more than a few thousand observations, stratified matching designs could greatly improve the scalability of optimal matching. While the worst-case computational complexity of optimal matching is unfavorable, the process of matching a stratified data set with a constant stratum size scales much more effectively with sample size (for a summary of empirical run-times, see section 2.5.3).

While a variety of packages in R (R Core Team, 2019) exist for matching subjects in observational studies, limited support exists for researchers seeking to implement a stratified matching design. The popular **MatchIt** package (Ho et al., 2011) is a user-friendly option for common propensity score matching designs and related approaches, **optmatch** (Hansen and Klopfer, 2006) and **DOS2** (Rosenbaum, 2019) are a powerful combination for implementing a variety of more complicated optimal matching schemes, and **nearfar** (Rigdon et al., 2018) implements a different form of matching for the instrumental variable study. The primary goal of **stratamatch** is to make stratified matching and prognostic score designs accessible to a wider variety of applied researchers and to suggest a suite of diagnostic tools for the stratified observational study. In favorable settings, these designs could not only increase the precision and robustness of inference but could facilitate the optimal matching of sample sizes for which this technique was previously computationally impractical.

This paper discusses the methodological contributions of **stratamatch** – in particular, the implementation of a novel pilot design approach suggested by Aikens et al. (2020) (section 2.2) – and summarizes the package implementation (Section 2.3) with illustrative examples (Section 2.4). While **stratamatch** may substantially improve the scalability of optimal matching for some large data sets, the main objective of the package is not to implement a computationally complex task but to make sophisticated study design tools and concepts accessible to a wide variety of researchers.

Study design

A prognostic score stratification pilot design

Stratifying a data set based on baseline variation prior to matching reduces the heterogeneity between matched sets with respect to that baseline variation. But what baseline characteristics should be used? One option is to select prognostically important covariates by hand, based on expert knowledge. However, in practice, this "manual" stratification process often produces strata that vary wildly in size and composition. Some strata may be so small or so imbalanced in their composition of treated and control individuals that it is difficult to find high-quality matches, or many observations are thrown away. Other strata may be so large that matching within them is still computationally infeasible.

The `auto_stratify` function in **stratamatch** divides subjects into strata using a *prognostic score* (see Hansen (2008)), which summarizes the baseline variation most important to the outcome. In addition to producing strata of more regular size and composition, balancing treatment and control groups based on the prognostic score may confer several statistical benefits: increasing precision (Aikens et al., 2020; Leacy and Stuart, 2014), providing some protection against mis-specification of the propensity score (Leacy and Stuart, 2014; Antonelli et al., 2018), and decreasing the susceptibility of an observed effect to being explained away by unobserved confounding (Rosenbaum and Rubin, 1983; Aikens et al., 2020). However, fitting the prognostic score on the same data set raises concerns of overfitting and may lead to biased effect estimates (Hansen, 2008; Abadie et al., 2018). For this reason, (Aikens et al., 2020) suggest using a *pilot design* for estimating the prognostic score.

Central to the pilot design concept is maintaining separation between the design and analysis phases of a study (see table 1, or for more information Goodman et al. (2017) and Rubin (2008)). Using an observational pilot design, the researchers partition their data set into an *analysis set* and a held-aside *pilot set*. Outcome information in the pilot set can be observed (e.g. to fit a prognostic score), and the information gained can be used to inform the study design. Subsequently, in order to preserve the separation of the study design from the study analysis, the individuals from the pilot set are omitted from the main analysis (i.e., they are not reused in the analysis set). The primary insight of the pilot design is that reserving all of the observations in a study for the analysis phase (i.e., in the analysis set) is not always better. Rather, clever use of data in the design phase (i.e., in the pilot set) may facilitate the design of stronger studies.

In the **stratamatch** approach, a random subsample of controls is extracted as a pilot set to fit a prognostic model, and that model is then used to estimate prognostic scores on the mix of control and treated individuals in the analysis set. The observations in the analysis set can then be stratified based on the quantiles of the estimated prognostic score and matched by propensity score or Mahalanobis distance within strata (see section 2.3).

When to use this approach

Aikens et al. (2020) describe the scenarios in which a prognostic score matching pilot design is most useful. Briefly, the **stratamatch** approach is best for large data sets (i.e., thousands to millions of observations), especially when the number of control observations is plentiful. This technique may be particularly useful when modeling a prognostic score with the measured covariates is straightforward, and when propensity score alone is likely to exclude certain aspects of variation highly associated with outcome but unassociated with treatment assignment. While computational gains vary, stratification tends to noticeably accelerate matching for sample sizes of 5,000 or more (see section 2.5.3).

Conversely, this technique is not recommended for small data sets in which each control observation is precious, especially when prognostic scores are likely to be difficult to estimate from the measured covariates (see Aikens et al. (2020) for a lengthy discussion). Ideally, there should be ample control observations available to fit a usable prognostic model and still leave sufficient controls remaining to select high-quality matches for the treated individuals in the data set. While some **stratamatch** designs may be useful for the estimation of other causal estimands, the statistical properties of prognostic pilot designs for estimands other than the average treatment effect among the treated have not yet been characterized (Aikens et al., 2020).

Term	Description
Design phase	Phase of a study in which the researcher considers what kinds of data will provide the strongest information to address the question at hand (e.g., randomization, sampling, matching, inverse probability weighting). The goal of the design phase is to obtain data that will provide strong inference.
Analysis phase	Phase of a study in which the data that comes from the design phase are summarized into statistics. Inference and sensitivity analyses are performed.
Pilot Design	An observational study approach in which some data is spent in the design phase to improve the study design/preprocessing.
Pilot Set	A subset of data extracted to be used in the design phase.
Analysis set	The set of data reserved for inference in the analysis phase.
Propensity score	Probability of assignment to the treatment group based on measured baseline characteristics.
Prognostic score	Expectation of the outcome in the absence of treatment based on measured baseline characteristics.
Prognostic model	A model (e.g., logistic regression) used to estimate prognostic scores.
Stratum	A subset of observations in the analysis set to be matched together.

Table 1: Summary of relevant methodological terms as they apply to **stratamatch**.

Software

The **stratamatch** function, `auto_stratify`, implements the prognostic score stratification in the pilot design described above. The most basic procedure does the following:

1. Partition the data set into a pilot data set and an analysis data set
2. Fit a model for the prognostic score from the observations in the pilot set
3. Estimate prognostic scores for the analysis set using the prognostic model
4. Stratify the analysis set based on prognostic score quantiles.

A call to `auto_stratify` produces an `auto_strata` object, which contains the analysis set, the pilot set, and other information about the strata and prognostic scores. The **stratamatch** package implements a set of diagnostic plots and tables that can be used to assess the quality of a stratification. Example code, output, and diagnostics are provided in section 2.4. If the strata are satisfactory, the treatment and control individuals within each stratum can then be matched. By default, the `strata_match` function performs $1:1$ propensity score matching within each stratum. Other matching scheme possibilities are discussed in section 2.5.3).

Illustrations

Simulated example

This section demonstrates the basic functionality of **stratamatch** in simulated example. The function `make_sample_data` generates a simple simulated data set so that users can explore the design options implemented by **stratamatch**. Below, we generate a sample of 10,000 observations and print the first few rows as an illustration.

```
library("stratamatch")
library("dplyr")
dat <- make_sample_data(n = 10000)
head(dat)
```

```
           X1        X2 B1 B2 C1 treat outcome
1  0.93332697 1.0728339  1  0  a     1       0
2 -0.52503178 0.3449057  1  1  b     0       1
3  1.81443979 1.0361942  1  1  a     0       0
4  0.08304562 0.3017060  1  1  a     0       1
```

```
5  0.39571880 0.5397257  0  0  c     0       0
6 -2.19366962 1.4523274  1  1  b     0       1
```

The user should suppose that the rows of `dat` are individuals in an observational study, and the objective of the study is to estimate the effect of a binary treatment assignment (`treat`) on a binary outcome (`outcome`). Columns 1-5 represent three types of measured baseline covariates: continuous (`X1` and `X2`), binary (`B1` and `B2`), and categorical (`C1`). For this example, we assume strongly ignorable treatment assignment - that is, roughly, there are no unmeasured confounding factors (Rosenbaum and Rubin, 1983). (For sensitivity analyses for this assumption, see, for example, Rosenbaum (2005b)).

Automatic stratification

The command below uses `auto_stratify` to (1) partition 10% of the controls in `dat` into the pilot set (2) fit a prognostic score model for `outcome` based on `X1` and `X2`, (3) estimate prognostic scores on the analysis set, and (4) return to us the analysis set, divided into strata of approximately 500 individuals, based on prognostic score quantiles. All of these steps are completed automatically with this function call, and the results are returned to us as `a.strat`.

```
a.strat <- auto_stratify(dat, treat = "treat", prognosis = outcome ~ X1 + X2,
+    pilot_fraction = 0.1, size = 500)

Constructing a pilot set by subsampling 10% of controls.
Fitting prognostic model via logistic regression: outcome ~ X1 + X2
```

The result returned by `auto_stratify` is an `auto_strata` object. Running `print` on this object supplies basic information about how the stratification process has been completed.

```
print(a.strat)

auto_strata object from package stratamatch.

Function call:
auto_stratify(data = dat, treat = "treat", prognosis = outcome ~
    X1 + X2, size = 500, pilot_fraction = 0.1)

Analysis set dimensions: 9234 X 8

Pilot set dimensions: 766 X 7

Prognostic Score Formula:
outcome ~ X1 + X2
```

Here, `auto_stratify` has partitioned away a pilot set of 766 control individuals to fit our desired prognostic model, leaving 9,234 individuals in the analysis set. Using the prognostic model, prognostic scores were estimated on the individuals in the analysis set, and these individuals were divided into strata with a target size of 500. In order to record these stratification assignments, an eighth column, `stratum`, has been appended to the analysis set. The number strata and range of strata sizes can be obtained from `summary(a.strat)`.

The analysis set and pilot set are accessible via `a.strat$analysis_set` and `a.strat$pilot_set`, respectively. The `strata_table` (accessed via `a.strat$strata_table`) reports the strata sizes and the prognostic score quantile bins that define each stratum.

Diagnostics

A major focus of the **stratamatch** package is suggesting diagnostics for the quality of stratification in observational studies. The `issue_table` reports the total size and composition of each stratum.

```
a.strat$issue_table

# A tibble: 19 x 6
   Stratum Treat Control Total Control_Proportion Potential_Issues
     <int> <int>   <int> <int>              <dbl> <chr>
 1       1   167     319   486              0.656 none
 2       2   149     337   486              0.693 none
```

```
 3       3   160     326   486              0.671 none
 4       4   132     354   486              0.728 none
 5       5   123     363   486              0.747 none
 6       6   122     364   486              0.749 none
 7       7   146     340   486              0.700 none
 8       8   109     377   486              0.776 none
 9       9   131     355   486              0.730 none
10      10   132     354   486              0.728 none
11      11   111     375   486              0.772 none
12      12   108     378   486              0.778 none
13      13   112     374   486              0.770 none
14      14   122     364   486              0.749 none
15      15   100     386   486              0.794 none
16      16   109     377   486              0.776 none
17      17   114     372   486              0.765 none
18      18   107     379   486              0.780 none
19      19    85     401   486              0.825 Small treat:control ratio
```

The 'Potential_Issues' column is meant to quickly flag strata that may be problematically large, small, or imbalanced in the ratio of treated and control samples. The "small treat:control ratio" flag for stratum 19 indicates that the proportion of treated individuals is 0.2 or lower[1]. This is a relatively common issue, which is often easily addressed (see section 2.6).

The **stratamatch** package implements four diagnostic plotting options:

1. **Size-Ratio Plot:** (Figure 1) Displays each stratum in the analysis set based on its size and the percentage of control observations in order to identify potentially problematic strata.
2. **Propensity Score Histogram:** (Figure 1) Displays the distribution of estimated propensity scores across the treatment and control groups within a single stratum or the entire analysis set. These plots are used for assessing propensity score overlap.
3. **Assignment-Control Plot:** (Figure 2) Displays each individual based on estimated propensity score and estimated prognostic score, based on visualizations from Aikens et al. (2020). As above, these plots can display a single stratum or the entire analysis set. Assignment-control plots are useful for checking the overlap and correlation of prognostic and propensity scores.
4. **Residual Plots:** (Not shown) Show the diagnostic plots for the prognostic model used to estimate the prognostic scores. It is essentially a wrapper for `plot.lm` (see the documentation for `plot.lm` in the base R package, **stats**). Note that since the pilot set alone is used to fit the prognostic model, only the pilot set is used for these diagnostic plots.

The code below makes each of the plot types listed above, including two assignment-control plots: one for the entire analysis set and one for a single stratum. The results are shown in figures 1 and 2, with interpretation in the figure captions. For propensity score histograms and assignment-control plots, the 'propensity' argument is required, specifying how the propensity scores should be estimated. Below, the propensity score is fit on the analysis set based on a regression of treatment assignment on 'X1', 'X2', 'B1', and 'B2' (for other input options, run `help(plot.auto_strata)` or `help(plot.manual_strata)`).

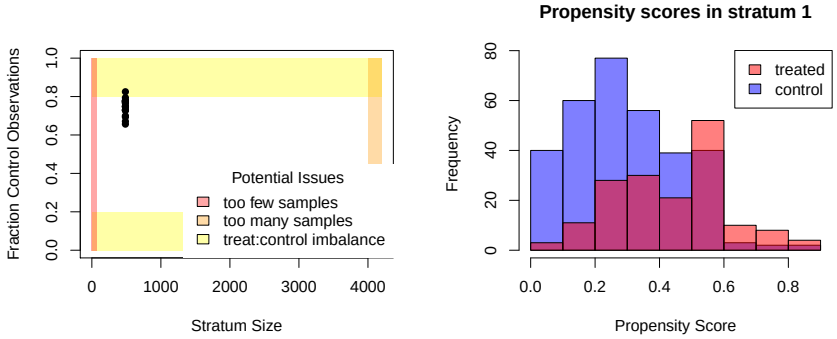
```
plot(a.strat, type = "SR")
plot(a.strat, type = "hist", propensity = treat ~ X2 + X1 + B1 + B2, stratum = 1)
plot(a.strat, type = "AC", propensity = treat ~ X2 + X1 + B1 + B2)
plot(a.strat, type = "AC", propensity = treat ~ X2 + X1 + B1 + B2, stratum = 1)
plot(a.strat, type = "residual")
```

In this example, the command `a.strat$prognostic_model` would supply the prognostic model (an `lm` or `glm` object) for further diagnostics (e.g., with `summary(a.strat$prognostic_model)`). Assessment of the prognostic model can indicate whether a sufficient number of observations has been partitioned into the pilot set (see section 2.6). However, one benefit of a stratified matching design is that even an imperfect prognostic model may yield robust inference if the resulting strata are of sufficient quality to allow for a strong propensity match (see, for example, theory on stratified sampling (Lohr, 2019) or commentary on doubly robust matching (Leacy and Stuart, 2014; Antonelli et al., 2018))

Matching

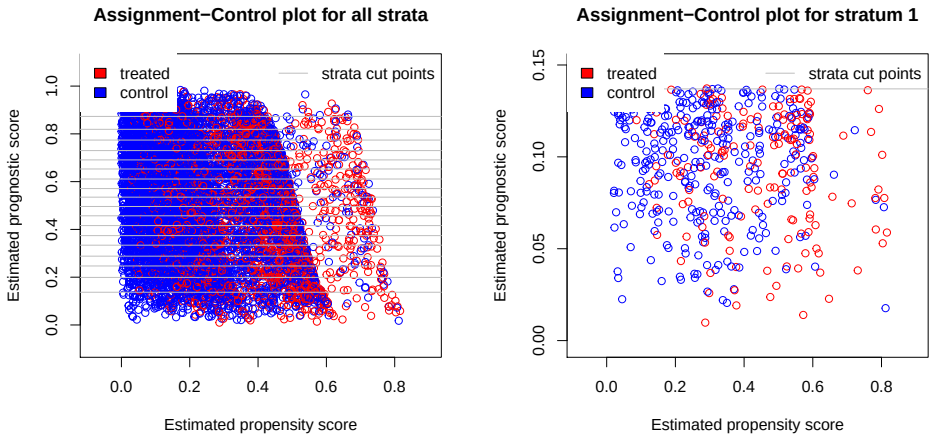
Once the data have been stratified, the user can optimally match individuals within each stratum. The `strata_match` function supports optimal 1:1, 1:k, or full matching (Rosenbaum, 1991; Hansen and

[1]Note that the specific thresholds defining the potential issue flags (e.g., 20% treated individuals or fewer) are not universal cutoffs but guidelines meant to draw researchers' attention to possible irregularities.



(a) A size ratio plot **(b)** A propensity score histogram

Figure 1: (A) A size-ratio plot, with each point representing a stratum. Yellow regions: treated to control ratio is imbalanced. Orange: strata size is large enough that matching may be computationally time-consuming. Red: strata are small enough that match quality may be poor. In a perfectly ideal stratification, all strata would fall within the white rectangle. In practice, some stratification issues are common and easily addressed; see section 2.6. (B) A histogram of estimated propensity scores for a selected stratum. In an ideal scenario, there is ample overlap between treated and control individuals within each stratum.



(a) An assignment-control plot across all strata **(b)** An assignment-control plot for stratum 1

Figure 2: Assignment-control plots (Aikens et al., 2020) showing estimated propensity score versus estimated prognostic score for each subject in the analysis set (A) or a selected stratum (B). In an ideal scenario, there is ample overlap between treated and control individuals in terms of both prognosis and propensity (for other cases, see section 2.6). Grey lines denote the prognostic score thresholds defining the strata.

Klopfer, 2006), based on a propensity score or Mahalanobis distance. The sample code below performs 1:1 propensity score matching. This function makes essential use of the **optmatch** package (Hansen and Klopfer, 2006; Bertsekas and Tseng, 1988) to perform the matching within strata.

```
mymatch <- strata_match(a.strat, model = treat ~ X1 + X2 + B1 + B2)

Fitting propensity model: treat ~ X1 + X2 + B1 + B2
```

The result is an optimal 1 to 1 matching within prognostic score strata. Above, `mymatch` is an `optmatch` class object, as described by the **optmatch** package (Hansen and Klopfer, 2006). For the most part, `mymatch` can be treated as a factor giving match assignments for each row of the data set. The command `summary(mymatch)` would display the number of pairs, the number of unmatched individuals, and the effective sample size. For suggestions regarding other matching schemes for stratified data, see 2.5.3.

A brief comment on estimation

The procedure for performing inference after matching – in particular the estimation of the standard error of the effect estimate for the purposes of hypothesis tests and confidence intervals – is a topic of some debate in the literature. We will not attempt to resolve this debate here, although interested readers may find the commentary by Stuart (Stuart, 2010) to be an accessible starting place, and note more recent work by Abadie and Speiss (Abadie and Spiess, 2021), and numerous other authors (for example Abadie and Imbens (2006, 2011); Austin and Small (2014); Austin and Cafri (2020)). Below, we describe two contrasting approaches that are most familiar to epidemiology and statistics, with references to coding resources.

First, Rosenbuam (Rosenbaum, 2005b; Rosenbaum et al., 2010) motivates the use of permutation-based tests followed by sensitivity analyses for unobserved confounding. Both of these are implemented in **sensitivitymw** (Rosenbaum, 2014, 2015) for pairmatching and **sensitivityfull** (Rosenbaum, 2017, 2007) for full matching. In keeping with a randomization inference framework, these techniques generally consider inference conditional on the matched sample and focus on uncertainty derived from the randomization process emulated by the matching. Researchers with further information on the sampling process that generated the observational data may thereafter combine this approach with a sampling variation framework to estimate parameters and standard errors for a more general target population (see the framework outlined by Tipton (2013) for the experimental setting).

A second common approach uses covariate adjustment. This framework is motivated importantly by Ho et al. (2007), who make the case for matching as a preprocessing step to reduce the dependence of parametric analyses on model selection. In keeping with the regression literature from the social sciences, these approaches often begin by supposing that the complete observational data set is an independent and identically distributed set of observations from some larger population, perhaps according to some parametric data-generating model. Within this framework, there is still considerable debate regarding correct standard error estimation. A thorough practical tutorial for the covariate adjustment approach with code examples and some suggestions for standard error estimation is featured in the recent **MatchIt** vignette, "Estimating Effects after Matching" (Greifer, 2020). Note that the pilot design implemented by **stratamatch** removes control individuals at random from the data set while retaining all treated individuals. Thus, while we recommend stratamatch for the estimation of the average treatment effect among the treated, the characteristics of stratamatch designs for estimation of other causal estimands (e.g., average treatment effect) have not yet been well characterized.

Real-data example: Life sustaining treatments for critical care patients

As an applied example, the **stratamatch** package contains a re-processed version of de-identified medical data from Chavez et al. (2018). Briefly, the authors extracted demographic information, common laboratory test results, comorbidity information, and treatment team assignments for 10,157 ICU patients from the Stanford University Hospital who met their inclusion criteria. During their stay, each patient's critical care preferences are summarized with a code status. The default – Full Code status – indicates no limitations on resuscitative measures, while other codes (e.g. 'Do not resuscitate', or 'DNR') indicate different limitations on the intensity and type of resuscitation the patient should receive if they become pulseless or apneic (i.e., their heart stops or they stop breathing). This code status is a product of complex dynamics between patient and provider. When a patient's code status does not reflect their goals of care, patients may have life-sustaining care inappropriately withheld, or they may receive aggressive treatment that does not effectively increase their quality or quantity of remaining life.

In this example, suppose a researcher wants to study whether comparable patients under the care of surgical teams vs. non-surgical teams are more likely to have their code status set to limit resuscitation (i.e., any form of 'DNR'). From this, we could infer tendencies that different treatment teams have in counseling and decision-making about life-sustaining treatments for the critically ill. However, the patient groups seen by surgical vs. non-surgical teams are necessarily different, because patients are assigned to treatment teams based on their reason for being in the hospital and their treatment history. Thus, a naive comparison of DNR order frequency between care team types would be misleading. To better account for these potential differences, we employ a stratified pilot matching design to compare "treated" (assigned to a surgical care team) individuals with "control" (assigned to a non-surgical care team) ones that are similar in terms of their prognostic and propensity scores.

Automatic stratification

Patients must be first stratified by a prognostic score (i.e., their estimated probability of receiving a DNR order if they are not assigned to a surgical care team) before being matched on a propensity score (i.e., their estimated probability of assignment to a surgical care team). In the example below, we use `auto_stratify` on the `ICU_data` to (1) partition 10% of controls into a pilot set, (2) build a prognostic score model on that pilot set based on age ('`Birth.preTimeDays`'), sex, and race/ethnicity (3) estimate prognostic scores on the analysis set and (4) return a stratified data set with approximately 500 individuals per stratum.

```
ICU_astrat <- auto_stratify(data = ICU_data, treat = "surgicalTeam",
    prognosis = DNR ~ Birth.preTimeDays + Female.pre + RaceAsian.pre +
      RaceUnknown.pre + RaceOther.pre + RacePacificIslander.pre +
      RaceBlack.pre + RaceNativeAmerican.pre + all_latinos,
    pilot_fraction = 0.1, size = 500)

Constructing a pilot set by subsampling 10% of controls.
Fitting prognostic model via logistic regression: DNR ~ Birth.preTimeDays +
    Female.pre + RaceAsian.pre + RaceUnknown.pre +  RaceOther.pre +
    RaceBlack.pre + RacePacificIslander.pre + RaceNativeAmerican.pre +
    all_latinos
```

Next, we print the results.

```
print(ICU_astrat)

auto_strata object from package stratamatch.

Function call:
auto_stratify(data = ICU_data, treat = "surgicalTeam",
    prognosis = DNR ~ Birth.preTimeDays + Female.pre + RaceAsian.pre +
        RaceUnknown.pre + RaceOther.pre + RaceBlack.pre +
        RacePacificIslander.pre + RaceNativeAmerican.pre + all_latinos,
        size = 500, pilot_fraction = 0.1)

Analysis set dimensions: 9364 X 14

Pilot set dimensions: 793 X 13

Prognostic Score Formula:
DNR ~ Birth.preTimeDays + Female.pre + RaceAsian.pre + RaceUnknown.pre +
    RaceOther.pre + RaceBlack.pre + RacePacificIslander.pre +
    RaceNativeAmerican.pre + all_latinos

summary(ICU_astrat)

Number of strata: 19

        Min size: 492         Max size: 494

Strata with Potential Issues: 2
```

We see here that `auto_stratify` partitioned the data into a pilot set of 793 "controls" (i.e., patients not assigned to a surgical treatment team) and an analysis set of the 9,364 remaining individuals. The

prognostic model was fit on the pilot set according to the formula we provided, regressing DNR code assignment on age, sex, and race. This model was used to estimate the prognostic score (probability of DNR code assignment based on demographics) for each of the 9,364 individuals in the analysis set. Finally, each individual in the analysis set was assigned to a stratum based on this score. 19 strata, each containing between 492 and 494 patients, were created. This stratum assignment information was appended to the analysis set by adding a new 14th column, `stratum`.

Manual stratification

Rather than using a pilot design to build a prognostic score, researchers may wish to stratify the data set based on discrete covariates (e.g., chosen by a domain expert). The `manual_stratify` function supports these study designs. For example, the code below bins the 10,157 patients in the data set purely based on race/ethnicity and sex. In contrast, the size-ratio plots for the automatic stratification show a much smaller range of sizes and control proportions, with fewer – and more easily addressed – potential issues.

```
ICU_mstrat <- manual_stratify(data = ICU_data,
    strata_formula = surgicalTeam ~ Female.pre + RaceAsian.pre +
      RaceUnknown.pre + RaceOther.pre + RaceBlack.pre +
      RacePacificIslander.pre + RaceNativeAmerican.pre + all_latinos)
summary(ICU_mstrat)

Number of strata: 16

        Min size: 17         Max size: 3314

Strata with Potential Issues: 9
```

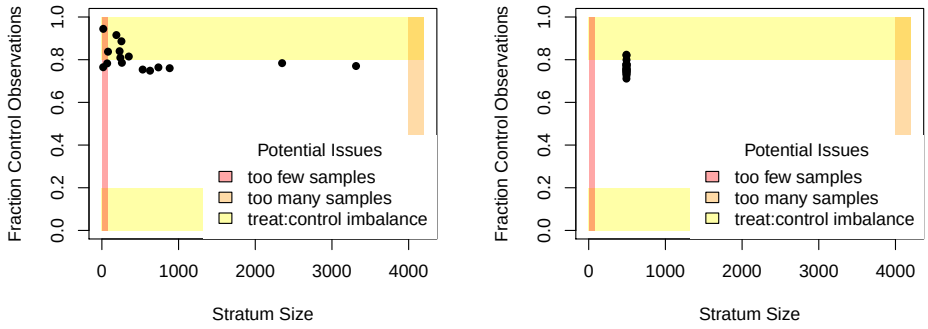
The resulting `manual_strata` object has many of the same properties as an `auto_strata` object from `auto_stratify` and can be matched in the same way with `strata_match`. However, `manual_strata` objects do not have a pilot set prognostic score information, and accordingly assignment-control and residual plots are not supported for these inputs.

This more traditional manual approach may be preferred in some cases for its simplicity and because it obviates the need to sacrifice observations to fit a prognostic model. However, selecting a binning scheme that results in favorable strata may be a time-consuming iterative process, as highlighted by the diagnostics in the following section. These issues underscore the potential usefulness of the prognostic score stratification implemented by `auto_stratify`.

Diagnostics

Size-ratio plots for the manual and automatic stratification illustrate a common issue with manual stratification: it is often difficult to select discrete covariates that result in appropriately sized and balanced strata (Figure 3). This also is reflected by the number of strata with potential issues in the manual stratification issue table below. For example, stratum 1 below (white males) contains 3,314 patients, while stratum 3 (Native American males) contains only 18 patients, only one of whom was assigned to a surgical team. In exceedingly large strata, matching becomes increasingly computationally intensive, while in exceedingly small and/or highly imbalanced strata, finding high-quality matches can be difficult or infeasible (see section 2.5.3).

```
 ICU_mstrat$issue_table

# A tibble: 16 x 6
   Stratum Treat Control Total Control_Proportion Potential_Issues
     <int> <int>   <int> <int>              <dbl> <chr>
 1       1   761    2553  3314              0.770 none
 2       2   212     672   884              0.760 none
 3       3     1      17    18              0.944 Too few samples; Small treat:con...
 4       4    13      67    80              0.838 Small treat:control ratio
 5       5    56     205   261              0.785 none
 6       6    65     286   351              0.815 Small treat:control ratio
 7       7    29     226   255              0.886 Small treat:control ratio
 8       8   174     563   737              0.764 none
 9       9   508    1842  2350              0.784 none
10      10   158     470   628              0.748 none
```



(a) Size-ratio plot for manual stratification **(b)** Size-ratio plot for automatic stratification

Figure 3: Size-ratio plots for (A) manual stratification on sex and race/ethnicity and (B) automatic stratifications of the same data set of ICU patients. Manual stratification often results in highly variable size and treat:control balance between strata, as reflected by the number of strata points in the shaded zones.

```
11      11     4      13    17            0.765 Too few samples
12      12    15      54    69            0.783 Too few samples
13      13    37     194   231            0.840 Small treat:control ratio
14      14    46     195   241            0.809 Small treat:control ratio
15      15    16     173   189            0.915 Small treat:control ratio
16      16   131     401   532            0.754 none
```

The code below displays the assignment-control plot for one of the strata in the automatically stratified data set (Figure 4).

```
plot(ICU_astrat, type = "AC",
    propensity = surgicalTeam ~ Female.pre + Birth.preTimeDays +
      RaceAsian.pre + RaceUnknown.pre + RaceOther.pre + RaceBlack.pre +
      RacePacificIslander.pre + RaceNativeAmerican.pre + all_latinos,
    stratum = 2)
```

The striae in this assignment-control plot appear when discrete characteristics (e.g. sex and race/ethnicity) are highly weighted in the propensity or prognostic score, causing observations to cluster together. Since this is relatively common, 'jitter' arguments can be used to add small amounts of random noise to the coordinates of each point in order to avoid stacking.

Matching

After a suitable stratification is selected, observations can be matched within strata using `strata_match`. Since every stratum from the automatic stratification in this example contains at least a 1:2 ratio of patients who were assigned to surgical teams and those who were not, we can match 2 "control" (i.e., non-surgical team) patients to each "treated" (i.e., surgical team) subject in each stratum. In this step, we match individuals who, based on their baseline covariates, appear equally likely to have been assigned to a surgical team vs. not. The following performs the matching.

```
ICU_match <- strata_match(ICU_astrat,
     model = surgicalTeam ~ Birth.preTimeDays + Female.pre +
      RaceAsian.pre + RaceUnknown.pre + RaceOther.pre + RaceBlack.pre +
      RacePacificIslander.pre + RaceNativeAmerican.pre + all_latinos,
    k = 2)
```

```
Fitting propensity model: surgicalTeam ~ Birth.preTimeDays + Female.pre +
RaceAsian.pre + RaceUnknown.pre + RaceOther.pre + RaceBlack.pre +
RacePacificIslander.pre + RaceNativeAmerican.pre + all_latinos
```

Below, we print a summary.

```
summary(ICU_match)
```

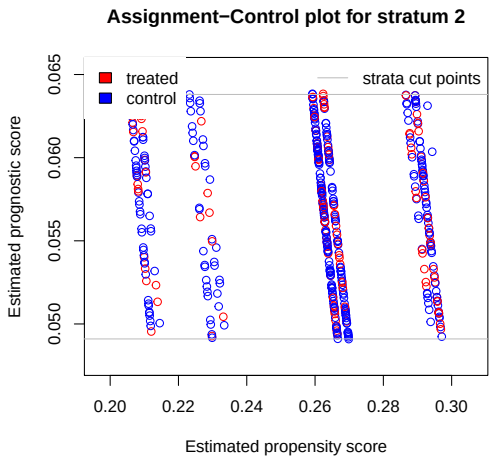


Figure 4: Assignment-control plot for automatic stratification of ICU data. The vertical striations are caused by heavily weighted discrete features in the propensity model, which cause points to align together.

```
Structure of matched sets:
 1:2  0:1
2226 2686
Effective Sample Size:  2968
(equivalent number of matched pairs).
```

At this point, the researcher can compare matched treated and control individuals to infer whether patients assigned to surgical treatment teams are more or less likely to be assigned a DNR code status, following up with sensitivity analyses (see, for example, **sensitivitymw** (Rosenbaum, 2014))

Key design choices and advanced functionality

The selection of the pilot set

The previous illustrations demonstrated the simplest method of extracting the pilot set: a random subsampling of all controls. Prior work by Aikens et al. (2020) contains a more thorough discussion of the considerations that might inform the selection of a pilot set.

A first consideration is the pilot set size. In general, the researcher should create a pilot set large enough to build a reliable prognostic model and retain enough remaining controls to select high-quality matches to the treatment group. This depends on the quality and number of available controls and the relative difficultly of fitting a prognostic model on the measured covariates. When high-quality controls (i.e., those resembling the treatment group) are scarce, the researcher should consider a smaller pilot set or a different study design altogether.

Another consideration is composition. Ideally, the individuals in the pilot set should be similar to the individuals in the treatment group, so a prognostic model built on this pilot set will not be extrapolating heavily when estimating prognostic scores on the analysis set. This approach can be especially important when there is some category of observations in the data that is relatively rare, and the researcher would like to ensure that some observations in this category end up in both the pilot and analysis sets. When discrete covariates are specified with the 'group_by_covariates' argument to auto_stratify the pilot set will be split proportionally based on these covariates so that the pilot set will be representative of the total control sample in terms of these covariates. This option can be used directly with auto_stratify. However, the split_pilot_set function is supplied as a convenience for users who prefer to split the pilot set themselves before stratification, as demonstrated below.

```
ICU_split <- split_pilot_set(ICU_data, treat = "surgicalTeam",
    pilot_fraction = 0.1, group_by_covariates = c("Female.pre", "self_pay"))

Constructing a pilot set by subsampling 10% of controls.
Subsampling while balancing on:
Female.pre self_pay
```

`ICU_split`, above, is a list containing a `pilot_set` and an `analysis_set`, partitioned while balancing sex and payment method (i.e., insurance or self-pay). Once this is done, the results can be passed to `auto_stratify` such as with the code below.

```
ICU_astrat2 <- auto_stratify(data = ICU_split$analysis_set,
    treat = "surgicalTeam",
    prognosis = DNR ~ Birth.preTimeDays + Female.pre + RaceAsian.pre +
      RaceUnknown.pre + RaceOther.pre + RacePacificIslander.pre +
      RaceBlack.pre + RaceNativeAmerican.pre + all_latinos,
    pilot_sample = ICU_split$pilot_set, size = 500)
```

Fitting the prognostic model

To fit the prognostic model, `auto_stratify` uses either linear (continuous outcome) or logistic regression (binary outcome). To accommodate a wider variety of modeling choices, `auto_stratify` can also be run using a vector of analysis set prognostic scores or prognostic model object[2].

The example below uses the **glmnet** package (Friedman et al., 2010) to fit a cross-validated lasso on the pilot set that was extracted in the previous section.

```
library("glmnet")
x_pilot <- ICU_split$pilot_set %>%
    dplyr::select(Birth.preTimeDays, Female.pre, RaceAsian.pre,
      RaceUnknown.pre, RaceOther.pre, RaceBlack.pre,
      RacePacificIslander.pre, RaceNativeAmerican.pre, all_latinos) %>%
    as.matrix()
y_pilot <- ICU_split$pilot_set %>%
    dplyr::select(DNR) %>%
    as.matrix()

cvfit <- cv.glmnet(x_pilot, y_pilot, family = "binomial")
```

The prognostic scores can then be estimated on the analysis set.

```
x_analysis <- ICU_split$analysis_set %>%
    dplyr::select(Birth.preTimeDays, Female.pre, RaceAsian.pre,
      RaceUnknown.pre, RaceOther.pre, RaceBlack.pre,
      RacePacificIslander.pre, RaceNativeAmerican.pre, all_latinos) %>%
    as.matrix()

lasso_scores <- predict(cvfit, newx = x_analysis, s = "lambda.min",
    type = "response")
```

Finally, these scores can be passed to `auto_stratify` with the 'prognosis' argument, producing a stratified data set that can be examined further with **stratamatch** diagnostic tools.

```
ICU_astrat3 <- auto_stratify(data = ICU_split$analysis_set,
    treat = "surgicalTeam", outcome = "DNR", prognosis = lasso_scores,
    pilot_sample = ICU_split$pilot_set, size = 500)
```

Other examples of prognostic score modeling options can be found in the **stratamatch** "Advanced Functionality" vignette.

Matching

Section 2.4 demonstrates how the **stratamatch** package can be used for optimal $1 : k$ matching on a propensity score. The `strata_match` function also supports full matching (Hansen and Klopfer, 2006; Rosenbaum, 1991), and the use of Mahalanobis distance instead of a propensity score. If desired, a data set stratified with **stratamatch** can instead be matched within strata using other matching software (e.g., **optmatch** (Hansen and Klopfer, 2006) or **MatchIt** (Ho et al., 2011)). For example, users proficient with **optmatch** will note that adding `+ strata(stratum)` to the matching formula supplied to `optmatch::pairmatch` and other matching functions will match within stratum assignments in the analysis set.

[2]Model objects must have a method associated with the `predict` generic function

More nuanced matching schemes may also help address imbalances in the number of treated and control units within strata. For example, the researcher could perform $1:k$ matching within each stratum, but allow k to vary between strata - matching more controls to each treated individual in strata where controls are plentiful and performing $1:1$ or $1:2$ matching where controls are less abundant. Another solution is to use a matching scheme within strata that naturally allows for variation in the ratio of treated and control individuals in matched sets, such as full matching (Rosenbaum, 1991; Hansen and Klopfer, 2006) or variable k matching (Pimentel et al., 2015).

As shown in figure 5, stratification is expected to substantially accelerate the matching process, especially for large sample sizes (several thousand or more). Hansen and Klopfer articulate a worst-case run-time for various forms of optimal matching with **optmatch** as $O(n^3 \log(nM))$, where M represents the maximum matching discrepancy between treated and control observations (Hansen and Klopfer, 2006). For context, this scales slightly less favorably than matrix inversion, which quickly becomes time-consuming for large inputs. By comparison, matching within strata of a fixed size tends to scale much more favorably for large n (figure 5). To further accelerate computation, a researcher might distribute matching the stratified data set over several computing nodes.

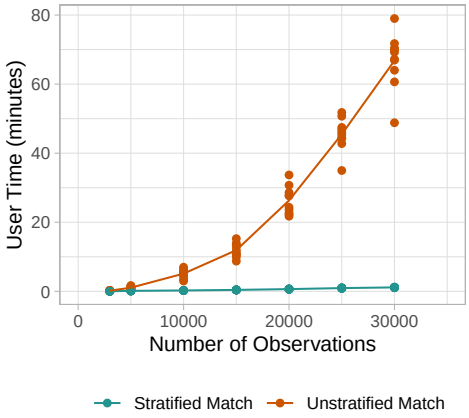


Figure 5: Measured computation times for stratified and unstratified matching on a modern laptop. Unstratified matching scales in a supra-linear manner with sample size (Hansen and Klopfer, 2006), while stratified matching with a set strata size tends to scale more favorably with n. At sample sizes of 30,000, optimally matching a whole data set may take over an hour, and much larger sample sizes may quickly become infeasible.

Trouble-shooting a stratification scheme

This section summarizes some common pitfalls and workarounds while stratifying a data set. Importantly, in order to preserve the separation of the design and analysis set, individuals partitioned into the pilot set must not be recombined with the analysis set. For instance, simply running `auto_stratify` repeatedly with different seeds to sample new pilot sets from the data and fit new prognostic score models may lead to overfitting of the prognostic model, raising concerns of bias in the study results (see Hansen (2008); Abadie et al. (2018)).

The following issues are common:

1. **Some strata are too small or too large:** This problem can often be solved simply by rerunning `auto_stratify` with a different '`size`' parameter. When this is done, the researcher should be sure to use the same pilot and analysis set as they received when they first ran `auto_stratify` (i.e., do not partition a new pilot set).
2. **The strata have a poor balance of treated and control individuals:** This situation is relatively common but often straightforward to address with matching schemes that match more controls to each treated observation or allow for variable treat:control ratios. See section 2.5.3 for some suggestions.
3. **The prognostic model is poor:** In some cases, the user may encounter an error fitting the prognostic model, or they may suspect from prognostic model diagnostics that the model does a poor job of capturing variation predictive of the outcome. There are a few reasons the prognostic model may be problematic.

(a) *The prognostic model was mis-specified.* In this case, the user should fit a revised prognostic model on the same pilot set as was previously used. However, refitting repeatedly can lead to overfitting, so this should be done in moderation.

(b) *The pilot set was too small to get a reliable fit.* In this case, the user can add more samples from the analysis set to the existing pilot set. Samples that are moved into the pilot set must stay in the pilot set and should not be re-pooled with the analysis set.

(c) *Pilot set size is sufficient, but prognostic model perfectly separates treated individuals from control individuals:* If this occurs in either the pilot set or analysis set, it may be a sign that overlap is poor. See below.

4. **The treated and control individuals within strata have poor overlap in propensity and/or prognostic scores:** This problem is best diagnosed with assignment-control plots (see Aikens et al. (2020) for a deeper description). Propensity and prognostic score based subclassification methods both depend on some form of overlap in the baseline characteristics of treated and control individuals in order to make a valid estimate of causal effect (for a summary, see Leacy and Stuart (2014)). Treatment and control groups that are clearly separated in terms of either their propensity scores or prognostic scores can indicate that these two groups should not be compared because the resulting inference on treatment effect would be misleading. A researcher facing this situation might consider trimming the score space (Glynn et al., 2019) in some cases or seeking out another data set if the overlap problems are severe. While this may seem to be a disappointing result, the ability to identify these data issues before proceeding is one of the most important strengths of design-based causal inference (see, for example, Austin (2011)).

Summary and discussion

Stratifying a data set prior to matching may make optimal and full matching designs scale more practically for modern observational sample sizes (Figure 5). However, the primary objective of **stratamatch** is not to directly implement a computationally taxing task, but to expand access to sophisticated study design tools for a wide range of researchers with varying levels of technical and statistical sophistication. Indeed, the computational steps of stratification are relatively straightforward; however, the statistical concept of the pilot design is nuanced, and the process of stratifying a data set and interrogating the quality of that stratification can be thought-intensive and isn't well-supported by other resources. The **stratamatch** package is intended to make prognostic score stratification pilot designs – and stratified matching designs in general – easily implementable, with helpful diagnostic tools and documentation. The overall goal of this effort is to push researchers toward approaches and diagnostics that emphasize stronger study design in the observational setting. In modern observational studies, designs, such as the **stratamatch** approach, that are *tailored* to large-sample studies can offer increased precision and other statistical benefits that might otherwise be left on the table by more traditional approaches.

Bibliography

A. Abadie and G. W. Imbens. Large sample properties of matching estimators for average treatment effects. *econometrica*, 74(1):235–267, 2006. [p620]

A. Abadie and G. W. Imbens. Bias-corrected matching estimators for average treatment effects. *Journal of Business & Economic Statistics*, 29(1):1–11, 2011. [p620]

A. Abadie and J. Spiess. Robust post-matching inference. *Journal of the American Statistical Association*, pages 1–13, 2021. [p620]

A. Abadie, M. M. Chingos, and M. R. West. Endogenous stratification in randomized experiments. *Review of Economics and Statistics*, 100(4):567–580, 2018. [p615, 626]

R. C. Aikens, D. Greaves, and M. Baiocchi. A pilot design for observational studies: Using abundant data thoughtfully. *Statistics in Medicine*, 39(30):4821–4840, 2020. [p614, 615, 618, 619, 624, 627]

J. Antonelli, M. Cefalu, N. Palmer, and D. Agniel. Doubly robust matching estimators for high dimensional confounding adjustment. *Biometrics*, 74(4):1171–1179, 2018. [p615, 618]

P. C. Austin. An introduction to propensity score methods for reducing the effects of confounding in observational studies. *Multivariate behavioral research*, 46(3):399–424, 2011. [p627]

P. C. Austin and G. Cafri. Variance estimation when using propensity-score matching with replacement with survival or time-to-event outcomes. *Statistics in medicine*, 39(11):1623–1640, 2020. [p620]

P. C. Austin and D. S. Small. The use of bootstrapping when using propensity-score matching without replacement: a simulation study. *Statistics in medicine*, 33(24):4306–4319, 2014. [p620]

D. P. Bertsekas and P. Tseng. Relaxation methods for minimum cost ordinary and generalized network flow problems. *Operations Research*, 36(1):93–114, 1988. [p620]

G. Chavez, I. B. Richman, R. Kaimal, J. Bentley, L. A. Yasukawa, R. B. Altman, V. S. Periyakoil, and J. H. Chen. Reversals and limitations on high-intensity, life-sustaining treatments. *PloS one*, 13(2): e0190569, 2018. [p620]

J. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):1–22, 2010. URL http://www.jstatsoft.org/v33/i01/. [p625]

R. J. Glynn, M. Lunt, K. J. Rothman, C. Poole, S. Schneeweiss, and T. Stürmer. Comparison of alternative approaches to trim subjects in the tails of the propensity score distribution. *Pharmacoepidemiology and drug safety*, 28(10):1290–1298, 2019. [p627]

S. N. Goodman, S. Schneeweiss, and M. Baiocchi. Using design thinking to differentiate useful from misleading evidence in observational research. *Jama*, 317(7):705–707, 2017. [p615]

N. Greifer. Estimating effects after matching. https://cran.r-project.org/web/packages/MatchIt/vignettes/estimating-effects.html, Dec. 2020. Accessed: 2021-5-3. [p620]

B. B. Hansen. The prognostic analogue of the propensity score. *Biometrika*, 95(2):481–488, 2008. [p615, 626]

B. B. Hansen and S. O. Klopfer. Optimal full matching and related designs via network flows. *Journal of Computational and Graphical Statistics*, 15(3):609–627, 2006. [p614, 618, 620, 625, 626]

M. A. Hernán and J. M. Robins. Using big data to emulate a target trial when a randomized trial is not available. *American journal of epidemiology*, 183(8):758–764, 2016. [p614]

D. E. Ho, K. Imai, G. King, and E. A. Stuart. Matching as nonparametric preprocessing for reducing model dependence in parametric causal inference. *Political analysis*, 15(3):199–236, 2007. [p620]

D. E. Ho, K. Imai, G. King, E. A. Stuart, et al. **MatchIt**: Nonparametric preprocessing for parametric causal inference. *Journal of Statistical Software, http://gking. harvard. edu/matchit*, 2011. [p614, 625]

G. King and R. Nielsen. Why propensity scores should not be used for matching. *Political Analysis*, 27 (4):435–454, Oct. 2019. [p614]

F. P. Leacy and E. A. Stuart. On the joint use of propensity and prognostic scores in estimation of the average treatment effect on the treated: A simulation study. *Statistics in medicine*, 33(20):3488–3508, 2014. [p615, 618, 627]

S. L. Lohr. *Sampling: Design and analysis: Design and analysis*. CRC Press, 2019. [p618]

S. D. Pimentel, F. Yoon, and L. Keele. Variable-ratio matching with fine balance in a study of the peer health exchange. *Statistics in medicine*, 34(30):4070–4082, 2015. [p626]

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2019. URL https://www.R-project.org/. [p614]

J. Rigdon, M. Baiocchi, and S. Basu. Near-far matching in R: The nearfar package. *Journal of Statistical Software, Code Snippets*, 86(5):1–21, 2018. [p614]

P. R. Rosenbaum. A characterization of optimal designs for observational studies. *Journal of the Royal Statistical Society B*, 53(3):597–610, 1991. [p618, 625, 626]

P. R. Rosenbaum. Heterogeneity and causality: Unit heterogeneity and design sensitivity in observational studies. *The American Statistician*, 59(2):147–152, 2005a. [p614]

P. R. Rosenbaum. Sensitivity analysis in observational studies. *Encyclopedia of statistics in behavioral science*, 4:1809–1814, 2005b. [p614, 617, 620]

P. R. Rosenbaum. Sensitivity analysis for m-estimates, tests, and confidence intervals in matched observational studies. *Biometrics*, 63(2):456–464, 2007. [p620]

P. R. Rosenbaum. *sensitivitymw: Sensitivity analysis using weighted M-statistics*, 2014. URL https://CRAN.R-project.org/package=sensitivitymw. R package version 1.1. [p620, 624]

P. R. Rosenbaum. Two r packages for sensitivity analysis in observational studies. *Observational Studies*, 1(1):1–17, 2015. [p620]

P. R. Rosenbaum. *sensitivityfull: Sensitivity Analysis for Full Matching in Observational Studies*, 2017. URL https://CRAN.R-project.org/package=sensitivityfull. R package version 1.5.6. [p620]

P. R. Rosenbaum. *DOS2: Design of Observational Studies, Companion to the Second Edition*, 2019. URL https://CRAN.R-project.org/package=DOS2. R package version 0.5.2. [p614]

P. R. Rosenbaum and D. B. Rubin. The central role of the propensity score in observational studies for causal effects. *Biometrika*, 70(1):41–55, 1983. [p614, 615, 617]

P. R. Rosenbaum et al. *Design of Observational Studies*, volume 10. Springer-Verlag, 2010. [p614, 620]

D. B. Rubin. For objective causal inference, design trumps analysis. *The Annals of Applied Statistics*, 2 (3):808–840, 2008. [p615]

E. A. Stuart. Matching methods for causal inference: A review and a look forward. *Statistical science: a review journal of the Institute of Mathematical Statistics*, 25(1):1, 2010. [p620]

E. Tipton. Improving generalizations from experiments using propensity score subclassification: Assumptions, properties, and contexts. *Journal of Educational and Behavioral Statistics*, 38(3):239–266, 2013. [p620]

Rachael C. Aikens
Interdepartmental Program in Biomedical Informatics
Stanford University
Stanford, CA
USA

Joseph Rigdon
Department of Biostatistics and Data Science
Wake Forest School of Medicine
Winston-Salem, North Carolina

Justin Lee
Quantitative Sciences Unit
Stanford University
Stanford, CA

Michael Baiocchi
Epidemiology and Population Health
Stanford University
Stanford, CA

Andrew B. Goldstone
Division of Cardiovascular Surgery
University of Pennsylvania
Philadelphia, PA

Peter Chiu
Department of Cardiothoracic Surgery
Stanford University School of Medicine
Stanford, CA

Y. Joseph Woo
Department of Cardiothoracic Surgery
Stanford University School of Medicine
Stanford, CA

Jonathan H. Chen
Biomedical Informatics Research Institute

Stanford University Medical Center
Stanford, CA
jonc101@stanford.edu

Changes in R 4.0–4.1

by Tomas Kalibera, Sebastian Meyer and Kurt Hornik

Abstract We give a selection of the most important changes in R 4.1.0. Some statistics on source code commits and bug tracking activities are also provided.

R 4.1.0 selected changes

R 4.1.0 (codename "Camp Pontanezen") was released on 2021-05-18. The following gives a selection of the most important changes.

- R now provides a simple native forward pipe syntax `|>`. The simple form of the forward pipe inserts the left-hand side as the first argument in the right-hand side call. The pipe implementation as a syntax transformation was motivated by suggestions from Jim Hester and Lionel Henry. The current implementation does not provide a shorthand for passing the left-hand side as an argument other than the first. Several options for providing such a shorthand are currently under consideration.
- R now provides a shorthand notation for creating functions, e.g. `\(x) x + 1` is parsed as `function(x) x + 1`.
- The base environment and its namespace are now locked (so one can no longer add bindings to these or remove from these)
- Support for gradient fills, pattern fills, clipping paths and masks has been added to the R graphics engine. An R-level interface for these new features has been added to the **grid** graphics package. See Paul Murrell's blog post for more details.
- Graphics devices can now specify `deviceClip`. If `TRUE`, the graphics engine will never perform any clipping of output itself. The clipping that the graphics engine does perform (for both `canClip = TRUE` and `canClip = FALSE`) has been improved to avoid producing unnecessary artifacts in clipped output. See Paul Murrell's blog post for more details.
- New palettes `"Rocket"` and `"Mako"` for `hcl.colors()` (approximating palettes of the same name from the 'viridisLite' package). Contributed by Achim Zeileis.
- `Rterm`, the command-line R front-end on Windows, now supports line editing and cursor motion with multi-byte and multi-width printable characters. It is now possible to use `RTerm` with non-European characters when the current locale (e.g. double-byte) supports them. Previously, users of non-European languages had to resort to other front ends, e.g. `RGui`. On (still experimental) UCRT builds of R on recent Windows 10, the native encoding is UTF-8 and hence all characters supported by Windows and the font can be used in `Rterm`, including characters outside of Basic Multilingual Plane (surrogate pairs in UTF-16LE). This required a significant rewrite of code originating from getline library. See Tomas Kalibera's blog post for more details.
- Data set `esoph` in package **datasets** now provides the correct numbers of controls; previously it had the numbers of cases added to these. Reported by Alexander Fowler in PR#17964.
- Using `c()` to combine a factor with other factors now gives a factor (an ordered factor when combining ordered factors with identical levels).
- New function `charClass()` has been added to package **utils** to query the wide-character classification functions in use by R (such as `iswprint`, `iswalpha`, `islower`). On Windows and by default on macOS and AIX, the classes are determined by R's internal tables. On Linux, the C99 functions and the classification provided by the platform are used. `charClass()` accepts UTF-8 encoded R strings and integer vectors of Unicode points on input.
- R's internal Unicode tables for character classification and character width were updated to Unicode 13.0.0 (used on Windows and by default on macOS and AIX). Handling of `\U` escapes in the parser was improved. String truncation is now more careful: most instances in R have been fixed not to produce incomplete multi-byte characters. Additionally, there were several encoding-related bug fixes.
- R and CRAN package binaries are now available also for the new Apple silicon Macs (M1 and higher) as native 64-bit ARM builds. Fortran code is compiled by a development version of GNU Fortran compiler from Iain Sandoe as no free Fortran 90 compiler for the platform has been released, yet. Only minimal changes to base R were needed for this: now R turns off floating-point ARM RunFast mode, hence disabling flush-to-zero and default-NaN modes. The default-NaN mode is not desirable for R because it causes R `NA` values to become `NaN` even in

operations involving otherwise only finite values (`NA * 1` would be NaN). For more details on the NaN/NA issue, see an otherwise already outdated blog post of Tomas Kalibera and Simon Urbanek. For more information on M1 support in R, see R Installation and Administration.

- An experimental build of R and binaries of CRAN packages and their Bioconductor dependencies is now available for Windows. The builds use UCRT as the C runtime (previous builds used MSVCRT) to allow setting UTF-8 as the native encoding on recent Windows 10, hence substantially reducing the amount of encoding issues in R on that platform. All required external libraries had to be rebuilt, because all code linked statically on Windows needs to use the same C runtime. This used a new GCC 10 MinGW-w64 cross- and native toolchains compiled using MXE. These builds use R-devel and did so also at the time of 4.1.0 release, but were still experimental at that time and only selected patches have been ported to 4.1.0 (accepting UTF-8 as native encoding, the rewrite of RTerm/getline, Windows installer improvements). More details are available in Tomas Kalibera's blog posts from March 2021, July 2020, and May 2020.

R 4.1.0 code statistics

From the source code Subversion repository, the overall change between April 25, 2020 and May 28, 2021 (so between R 4.0.0 and R 4.1.0) was: 33,000 added lines, 14,000 deleted lines and 1000 changed files. This is rounded to thousands/hundreds and excludes changes to common generated files, bulk re-organizations, etc. (translations, parsers, autoconf, LAPACK, R Journal bibliography, test outputs, Unicode tables, incorporated M4 macros). This change is slightly bigger than that between R 3.6.0 and R 4.0.0 (24% more insertions, 10% more deletions, 4% more changed files), see News and Notes from the December 2020 issue of the R Journal.

Figure 1 shows commits by month and weekday, respectively, counting line-based changes in individual commits, excluding the files as above. The statistics are computed the same way as in the previous issue, hence allowing direct comparisons, but monthly statistics are impacted by the release date which varies across versions, hence impacting the numbers for April and May. The statistics cover code directly committed to the R-devel trunk, plus commits from the R-defs branch (graphics code from Paul Murrell). The latter was merged into R-devel in July 2020, but the statistics is based on months/days the original commits were made to R-defs, including from December 2019.

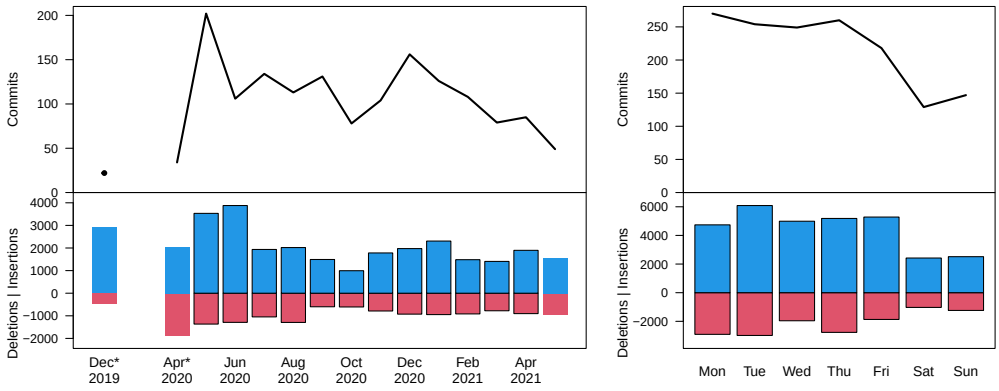


Figure 1: Commit statistics by month (left) and weekday (right) during R 4.1.0 development. *Note that the counts for April 2020 and May 2021 do not cover full months. Commits from December 2019 represent work of Paul Murrell on the later merged R-defs branch.

We observe an activity peak just after the release of R 4.0.0, a minimum in October 2020, and otherwise relatively stable amounts of code changes. The large numbers in May/June do not seem to follow a general pattern: here Paul Murrell did most of his changes on graphics. The right-hand plot shows that there is still a number of contributions even during the weekends.

R 4.1.0 bugs statistics

Summaries of bug-related activities during the development of R 4.1.0 (from April 25, 2020 to May 18, 2021) were derived from the database underlying R's Bugzilla system. Figure 2 shows statistics of reported/closed bugs and number of added comments (on any bug report) by calendar month and weekday, respectively. Deviating from the previous issue, new bug reports (comment 0) are not

counted as comments, so these numbers cannot be compared directly. Note that monthly statistics are impacted by truncation at the release dates in April 2020 and May 2021, respectively.

Comments are added by reporters of the bugs, R Core members and external volunteers. When a bug report is closed, the bug is either fixed or the report is found invalid. In principle, this can happen multiple times for a single report, but those cases are rare. Hence the number of comments is a measure of effort (yet a coarse one which does not distinguish thorough analyses from one-liners) and the number of bug closures is a measure of success in dealing with bugs.

R 4.1.0 was released by about 3 weeks later than usual, so the period which is summarized is also longer. The bug-related activities have still increased much more than what could be explained by that: about 17% more bugs closed than for (during development of) 4.0.0, but 55% more bugs reported). The increase from 3.6.0 to 4.0.0 was 45% more bug reports and 92% more bugs closed.

There was a significant increase in the number of comments following a blog post of Tomas Kalibera and Luke Tierney, published October 9, 2019 (so during development of 4.0.0), asking the R community for help with the bugs. The rate of comments stayed relatively high until now, so for the development of 4.1.0. This increased activity also came with more bug reports and more bugs closed. Initially, more bugs were closed than reported (4.0.0 development), but this changed during 4.1.0. It may be that the bugs fixed initially with the help of external volunteers were the older ones easy to handle, but now there is space for external volunteers to help with the new/harder ones.

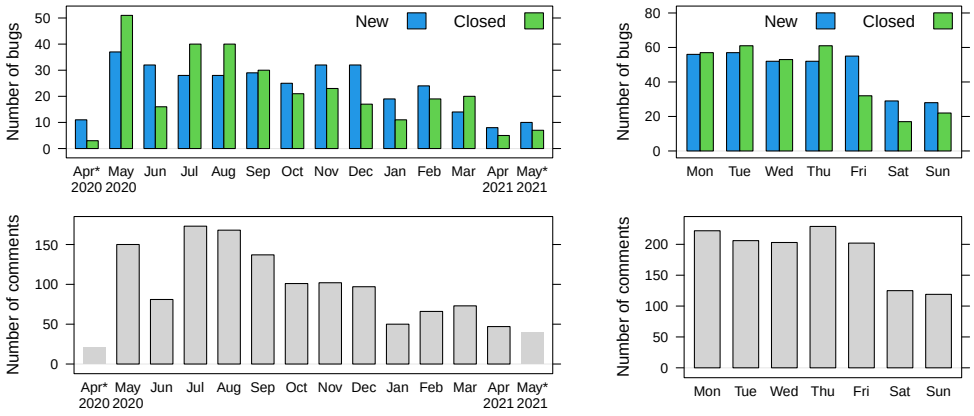


Figure 2: Bug tracking activity by month (left) and weekday (right) during R 4.1.0 development. *Note that the counts for April 2020 and May 2021 do not cover full months.

From the numbers by weekday in the right panel of Figure 2 we see that the R community still keeps working during the weekends. Still, the overall bigger bug-related activity seems relatively more reduced during the weekends than during 4.0.0 and 3.6.0 development.

Acknowledgements

Tomas Kalibera's work on the article and R development has received funding from the Czech Ministry of Education, Youth and Sports from the Czech Operational Programme Research, Development, and Education, under grant agreement No.CZ.02.1.01/0.0/0.0/15_003/0000421, from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme, under grant agreement No. 695412, and from the National Science Foundation award 1925644.

Tomas Kalibera
Czech Technical University, Czech Republic
Tomas.Kalibera@R-project.org

Sebastian Meyer
Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany
seb.meyer@fau.de

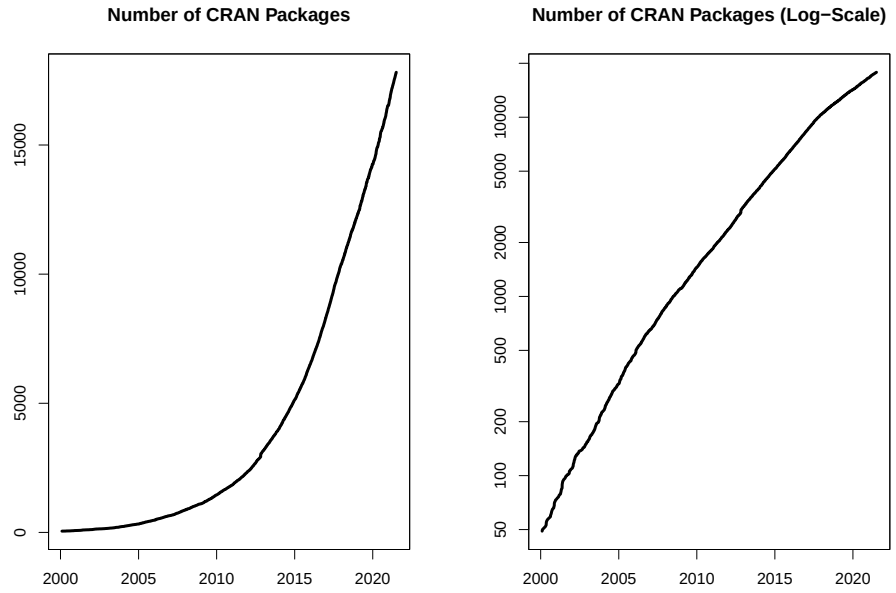
Kurt Hornik
WU Wirtschaftsuniversität Wien, Austria
Kurt.Hornik@R-project.org

Changes on CRAN

2021-01-01 to 2021-06-30

by Kurt Hornik, Uwe Ligges and Achim Zeileis

In the past 6 months, 1290 new packages were added to the CRAN package repository. 116 packages were unarchived and 467 were archived. The following shows the growth of the number of active packages in the CRAN package repository:



On 2021-06-30, the number of active packages was around 17778.

CRAN package submissions

During the the last three quarters (September 2020 to June 2021), CRAN received 25426 package submissions. For these, 44844 actions took place of which 29039 (65%) were auto processed actions and 15805 (35%) manual actions.

Minus some special cases, a summary of the auto-processed and manually triggered actions follows:

	archive	inspect	newbies	pending	pretest	publish	recheck	waiting
auto	6191	5988	5585	0	0	7244	2429	1602
manual	6586	117	874	855	249	5467	1322	335

These include the final decisions for the submissions which were

action	archive	publish
auto	5573 (22.5%)	6131 (24.8%)
manual	6478 (26.2%)	6544 (26.5%)

where we only count those as *auto* processed whose publication or rejection happened automatically in all steps.

CRAN mirror security

Currently, there are 103 official CRAN mirrors, 82 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 causact, dina, edina, errum, greta, mcmcensemble, rrum, shinybrms.

ClinicalTrials DTAT, Keyboard, MinEDfind, PowerUpR, UnifiedDoseFinding, cosa, crmPack, presize, randomizeR, replicateBE, rpact, simglm.

Cluster mdendro, mixR.

DifferentialEquations RxODE, mrgsolve, nlmixr.

Econometrics collapse, ivreg, lfe.

Finance AssetCorr, LSMRealOptions, NFCP, frenchdata, garchmodels.

FunctionalData face, mfaces, sparseFLMM.

HighPerformanceComputing proffer, profile, profmem, targets.

Hydrology AWAPer, HydroMe, LWFBrook90R, MODISTsp, airGRdatassim, synthesis, telemac.

MachineLearning DoubleML, lightgbm*, mlpack, splitTools.

MetaAnalysis CoTiMA, DTAplots, EvidenceSynthesis, MetaIntegration, RoBMA, RobustBayesianCopas, bnma, boutliers, forplo, fsn, gmeta, metaSurvival, metamicrobiomeR, metapack, nmaplateplot, smd.

MissingData InformativeCensoring, NADIA, dejaVu, grf, idem, lqr, misaem, missRanger, mixture, norm2, samon, semTools.

NumericalMathematics sanic.

OfficialStatistics collapse, longCatEDA, sdcMicro, simPop.

Optimization QPmin, SPOT, psqn, rminizinc, rmoo.

Psychometrics ata, tidyLPA.

ReproducibleResearch knitcitations, reportfactory, targets.

Robust clubSandwich, clusterSEs, skewlmm.

Spatial GWmodel, RCzechia, chilemapas, dbmss, geobr, geouy, giscoR, ipdw, mapSpain, osmextract, rgee, rgugik, terra.

Survival Cyclops, DAAG, InformativeCensoring, LTRCtrees, LogicReg, SGL, SimSurvNMarker, YPmodel, asbio, bayesSurv, bujar, concreg, etm, frailtyHL, frailtySurv, frailtypack, joineRML, kmc, kmi, mets, mlr3proba, npsurv, plsRcox, reReg, rstanarm, simPH, smoothSurv, spef, superpc, tranSurv.

TimeSeries HDTSA, LSTS, Rcatch22, Rsfar, VARDetect, autostsm, bayesforecast, blocklength, clock, collapse, dynr, fpcb, gsignal, legion, mfbvar, strucchangeRcpp, tensorTS, tsrobprep.

WebTechnologies AzureCosmosR, AzureGraph, AzureKusto, AzureQstor, AzureTableStor, AzureVision, Microsoft365R.

(* = 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

News from the Bioconductor Project

by Bioconductor Core Team

Bioconductor provides tools for the analysis and comprehension of high-throughput genomic data. Bioconductor 3.13 was released on 20 May, 2021. It is compatible with R 4.1.0 and consists of 2042 software packages, 406 experiment data packages, 965 up-to-date annotation packages, and 29 workflows.

Books were introduced in Bioconductor 3.12 and production continues in this release. These are built regularly from source and therefore fully reproducible; an example is the community-developed Orchestrating Single-Cell Analysis with Bioconductor.

The Bioconductor 3.13 release announcement includes descriptions of 133 new software packages, and updates to NEWS files for many additional packages. Start using Bioconductor by installing the most recent version of R and evaluating the commands

```
if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")
BiocManager::install()
```

Install additional packages and dependencies, e.g., **SingleCellExperiment**, with

```
BiocManager::install("SingleCellExperiment")
```

Docker images provides a very effective on-ramp for power users to rapidly obtain access to standardized and scalable computing environments.

Key learning resources include:

- bioconductor.org to install, learn, use, and develop Bioconductor packages.
- A list of available software, linking to pages describing each package.
- A question-and-answer style user support site and developer-oriented mailing list.
- A community slack (sign up) for extended technical discussion.
- The F1000Research Bioconductor channel for peer-reviewed Bioconductor work flows.
- The Bioconductor YouTube channel includes recordings of keynote and talks from recent conferences including Bioc2020 and BiocAsia2020, in addition to video recordings of training courses.
- Our package submission repository for open technical review of new packages.

The 2021 Bioconductor conference will be virtual, August 4-6, 2021.

In conjunction with the Mexican Bioinformatics Network and the Nodo Nacional de Bioinformática CCG UNAM, the Comunidad de Desarrolladores de Software en Bioinformática have arranged two week-long online workshops addressing development of workflows with RStudio and shiny and analysis of single-cell RNA-seq experiments, August 9-13, 2021.

BiocAsia 2021 will be held November 1-4 2021 as a virtual event The website and call for contributed talks are not open yet. Keep an eye on the events page for updates. The Biopackathon project has many points of contact with Bioconductor and recurs monthly.

The National Human Genome Research Institute's Analysis and Visualization Laboratory (AnVIL) is developing with contributions from Bioconductor core team members. A series of recorded workshops on the use of Bioconductor to explore this cloud computing system is available; additional workshops will be presented in the Fall of 2021.

The Bioconductor project continues to mature as a community. The Technical and Community Advisory Boards provide guidance to ensure that the project addresses leading-edge biological problems with advanced technical approaches, and adopts practices (such

as a project-wide Code of Conduct) that encourages all to participate. We look forward to welcoming you!

Bioconductor Core Team
Biostatistics and Bioinformatics
Roswell Park Comprehensive Cancer Center, Buffalo, NY
USA maintainer@bioconductor.org

R Foundation News

by Torsten Hothorn

Donations and members

Membership fees and donations received between 2021-01-29 and 2021-07-05.

Donations

Joaquín Baquer-Miravete (Spain) Generators in Smithfield, Virginia (United States) Ruedi Epple (Switzerland) Vancouver Fencing (Canada) Abbotsford Handyman (Canada) ken ikeda (Japan) Roofers Akron Ohio (United Kingdom) Seattle Handyman (Canada) Vancouver Handyman (Canada) Langley Handyman (Canada) Concrete Maple Ridge (Canada) Concrete Maple Ridge (Canada) Concrete Maple Ridge (Canada) Juan Perez-Franco (Chile) Vancouver Roofing (United States) Tuckpointing Vancouver (United States) Bend Fencing Companies (United States) Dr. Alfred Wagner (Germany) Daniel Wollschläger (Germany) Duct Cleaning Winnipeg (Canada) Merck Research Laboratories, Kenilwort (United States)

Supporting institutions

Alfred Mueller Analytic Services, München (Germany) Department of Clinical Research, University Hospital Basel, Basel (Switzerland) Dotcom-Tools, Wayzata (United States)

Supporting members

Kristoffer Winther Balling (Denmark) Ashanka Beligaswatte (Australia) Frederic BERTRAND (France) Michael Blanks (United States) Ryan Bonifacino (United States) Wesley Brooks (United States) Jack Brown (United Kingdom) John Chandler (United States) Gerard Conaghan (United Kingdom) Michael Dorman (Israel) Werner Engl (Austria) Guenter Faes (Germany) Bernd Fröhlich (Germany) Jan Marvin Garbuszus (Germany) Gabriel Gersztein (Brazil) Robert Hickman (United Kingdom) Alexander Huelle (Germany) Péter Kalicz (Hungary) Christian Kampichler (Netherlands) Jungjoon Kim (Korea, Republic of) Gavin Kirby (United Kingdom) Sebastian Koehler (Germany) Sebastian Krantz (Germany) Chris Kuty (United States) HOONJEONG KWON (Korea, Republic of) Adrien Le Guillou (France) Michal Majka (Austria) Myriam Maumy (France) Ernst Molitor (Germany) David Monterde (Spain) Stefan Moog (Germany) Steffen Moritz (Germany) Jens Oehlschlägel (Germany) Bill Pikounis (United States) VASILEIOS PLESSAS (United Kingdom) Stefano Rezzonico (Canada) Harald Sterly (Germany) Robert van den Berg (Austria) Fredrik Wartenberg (Sweden) Jason Wyse (Ireland)

Torsten Hothorn
Universität Zürich, Switzerland Torsten.Hothorn@R-project.org

News from the Forwards Taskforce

by Heather Turner

Forwards is an R Foundation taskforce working to widen the participation of under-represented groups in the R project and in related activities, such as the *useR!* conference. This report rounds up activities of the taskforce during the first half of 2021.

Community engagement

Kevin O'Brien has been conducting a series of video interviews as part of the *Why R? World* series. A number of interviews are ready to watch on the Why R? Foundation YouTube channel and several more rehearsal interviews have been conducted preparing the ground for a later recording. The interviewees include organizers of local groups, such as Nontsikelelo Shongwe (Eswatini); other community-builders, such as Reinaldo Zazela (Mozambique) connecting R users in Lusophone Africa, and other data scientists/related professionals talking about their work, such as Lais Carvalho (Brazil/Ireland) working in Developer Relations.

Kevin also facilitated the R-Ladies Remote Takeover of the *Why R? Webinar* series. Co-organized with Janani Ravi (R-Ladies Remote/R-Ladies East Lansing) and Heather Turner (Forwards/R-Ladies Remote) the *Takeover* featured three talks from speakers from across the globe, spread throughout one day. First up was Afiqah Masrani (Malaysia) on *Using {gtsummary} For Public Health Research*, then Tuli Amutenya (Namibia), on *Natural Language Processing for Survey Text Data* and finally Beatriz Valdez (Venezuela) on *Using R and Text Mining to find a Common Ground for Action in Polarized Contexts*. We were happy to provide a platform for these R-Ladies who live far from an active group (both R-Ladies and R User groups are being started in Caracas, Venezuela, however). The speaker from each talk co-chaired the next talk helping to make connections and a good atmosphere for the Q&A. We hope to run a similar event in future.

R Contribution Working Group

The R Contribution Work Group has begun work on some initiatives to encourage new contributors to R core, having laid the groundwork in the second half of 2020.

The R Foundation funded a 10-week project to develop a first version of the R Developer's Guide, a user-friendly introduction to contributing to R core. This project was undertaken by Saranjeet Kaur, mentored by Michael Lawrence and Heather Turner. Currently the guide covers identifying, reporting and reviewing bugs; preparing and submitting a patch; contributing to documentation, and testing pre-release versions of R. It also includes some technical help (building and installing R-devel on Windows, developer tools) and some community orientation (list of R core developers/contributors, where to get help and keep up-to-date with R project news). This guide benefited from review by RCWG members during and after the project, and we welcome members of the wider R developer community to review the guide and contribute to its further development as documented in Chapter 1 of the guide.

The R-devel Slack group now has over 100 members and is gradually seeing more activity. We welcome anyone interested in contributing to R to join - it provides a space for wider discussion compared to the R-devel mailing list and a supportive community for people new to contributing.

useR! 2021

Forwards was involved in a wide range of activities related to *useR! 2021*.

The Conference Team provided advice to the organizers on aspects such as code of conduct and events for first-timers. Liz Hare was heavily involved with supporting accessibility practices, including helping to develop the accessibility guidelines for presenters, providing input to the communications team and testing the accessibility of conference tools. The latter led to some improvements for screen-reader users in The Lounge chat platform; although this tool was ultimately abandoned for *useR!*, these changes will benefit future users. Noa Tamir began work as a Senior Writer/Editor on the R Project's Google Season of Docs project to develop documentation supporting *useR!* organization.

The Community Team helped with encouraging participation from under-represented countries, directly contacting R users to let them know about *useR!* and the fee waivers available for those without funding. They also helped to enlist reviewers for MiR's pre-review service, as well as Zoom hosts and volunteers for the team providing online chat support during the conference.

The Survey Team supported the organizers in running a Diversity Survey, the preliminary results of which were presented in the Closing Session.

The On-ramps Team partnered with the R Contribution Working Group to arrange two contributor-focused tutorials. The first was on *Translating R to Your Language*, lead by Michael Chirico in collaboration with Michael Lawrence. This was attended by an enthusiastic group, that worked on translating R messages into Spanish, Bahasa Indonesia, Hindi and Hungarian. The second was on *Contributing to R* lead by Gabriel Becker in collaboration with Martin Mächler. Gabriel shared his experiences as an external contributor to R core, before the participants split into small groups to debug a past issue in R 3.3.2. Martin reviewed how the bug was fixed in later versions of R and the session was completed with some parting advice and a round table, with Michael Lawrence joining as a guest. The participants reported that the tutorial was accessible and rewarding. A sub-group has met since to work through some of the further exercises prepared by the tutors.

Jonathan Godfrey, long-time member of Forwards, was part of a group keynote on responsible programming. He touched on the importance of acknowledging disabled people in our communities and using appropriate language when speaking about disability. However, his main theme was choosing tools that are inclusive. "To be disabled means you have been excluded in some way. But when I am included, I am no longer disabled." An example is that HTML documentation is much more accessible to screen-readers than PDF. As a community/developers, we should aim to "make the right things easy to do, and the incorrect ones, harder".

The tutorials and sessions mentioned above are expected to be published soon on the R Consortium YouTube channel - links will also be shared on the *useR! 2021* website.

Package Development Modules

As reported in the last issue, the Teaching Team have been modularizing the Forwards Package Development workshop. The first 3 modules were run in February 2021, with a total of 51 people. Only 19 people took all three modules, suggesting the modular approach enables participants to select modules according to their existing knowledge.

The modules will be re-run in September, with an additional module on documentation and testing. The modules will be run on separate days: 20, 21, 23 and 24 September at 13:00 UTC. Registration pages for each module can be found from the Forwards Eventbrite Page.

Heather Turner
University of Warwick, UK
Heather.Turner@R-project.org

R Medicine 2020: The Power of Going Virtual

by Elizabeth J. Atkinson, Peter D. Higgins, Denise Esserman, Michael J. Kane, Steven J. Schwager, Joseph B. Rickert, Daniella Mark, Mara Alexeev, and Stephan Kadauke

Abstract The third annual R/Medicine conference was planned as a physical event to be held in Philadelphia at the end of August 2020. However, a nationwide lockdown induced by the COVID-19 pandemic required a swift transition to a virtual conference. This article describes the challenges and benefits we encountered with this transition and provides an overview of the conference content.

Introduction

R/Medicine is an active working group of the R consortium, with the goal of supporting R users in medicine, medical research, and related health fields. The group includes a mix of statisticians and physicians who use R regularly in their work, as well as members from RStudio and the R Foundation.

The 2018 inaugural R/Medicine conference was a two day event held near Yale University with Rob Tibshirani providing the initial keynote address; each day began with a two hour tutorial. In 2019 the conference was held in Boston and included, among others, keynote addresses by Terry Therneau and Frank Harrell. Two half-day workshops were held on the day prior to the conference.

In 2020, the R/Medicine Working Group organized the third annual R/Medicine conference which was originally planned to be held in Philadelphia. By early April 2020 it was clear that the conference would need to be held virtually. This article describes some of the challenges as well as unexpected benefits we encountered when pivoting to a virtual conference, with the hope of providing useful information for other conference organizers.

Conference Planning and Logistics

Initial planning for the conference began in January 2020. By February, we had identified keynote speakers and workshop leaders, and started negotiations with a hotel near the Children's Hospital of Philadelphia which was pegged as the site of the conference. However, by the beginning of April we realized that we would need to switch to a virtual event.

The initial challenge for the planning committee was identifying technology that would allow us to run a main session of presentations, teach short courses, and host Birds-of-a-Feather (BoF) activities that allow like-minded R users to network and socialize. Because of our limited budget we had limited choices for an online conferencing platform. With help from the Linux Foundation, which provided recommendations of some of the available conference platforms, we chose to use Crowdcast (`https://www.crowdcast.io`). This tool was simple to use for the attendees and the organizers, had a green room to line up speakers, an interactive chat feature, and the ability to automatically provide a recording of the session for replay immediately after its conclusion. Compared to other conference platforms, Crowdcast has a rather minimalist set of features. However, feedback from conference participants confirmed that the user interface, consisting of a single-track video feed and sidebar participant chat worked well. One downside of Crowdcast was that it did not support screen readers or closed captioning.

For the short courses we chose Zoom (`https://zoom.us`) as the delivery platform, primarily because of its widespread use and because of a mature breakout room functionality. We found that breakout rooms are a useful tool for teaching, but should be used sparingly, because the overhead of sending participants into breakout rooms can interrupt the flow of teaching the material. For the *Introduction to R for Clinicians* course, we had a meet-and-greet breakout near the beginning of the course as well as a final "Hackathon" breakout in which participants were able to team up to design a dashboard.

Often, the most effective way to teach programming topics is live coding (Wilson, 2019). Much effort has been devoted to create teaching environments that allow participants to live code during a short course without requiring installation of any software. In the R ecosystem, the most notable and laudable effort in this regard is RStudio Cloud (`https://rstudio.cloud`), which is based on the RStudio Server Integrated Development Environment (IDE). From previous experience teaching R to an audience of clinicians who have no programming experience and are unfamiliar with the R ecosystem, we knew that the RStudio Cloud environment could be challenging because it requires each participant to first register a personal account on the platform before being granted access to the training environment. To address this, we created a training environment with pre-configured generic

sequentially numbered training accounts (e.g., `train001`, `train002`, etc.). After receiving training account credentials, a participant could directly access their training environment from a web browser without any further sign-ups. The training environment was implemented as a Docker-based web application based on RStudio Server which was configured with custom R packages and exercise files and hosted on a cloud provider. The application is available for free online (Kadauke, 2020a).

Zoom was used for the BoF sessions, and we used Slack for back-channel communication between the organizers, participants, and presenters. Some Slack channels were created for participants while other channels were limited to the conference planners. After the conclusion of the conference, all the recorded presentations were loaded onto the R/Medicine 2020 YouTube channel which is hosted by the R Consortium (2020). Since September 13, 2020 when the videos were uploaded, there have been over 3,200 views. Registration and abstract submission were facilitated by infrastructure provided by the Linux Foundation. Google Sheets were used to centralize organization of the program and the list of volunteers. We used Google Docs for creating conference planning documents and Google Forms for course sign-ups and surveys.

Since the `@r_medicine` Twitter account has a large following (2,557 as of June 2021), we used Twitter as the main outlet to promote the event. We also directly engaged specific groups that we wanted to participate to improve the conference experience, including R/Ladies, Minorities in R, as well as medical professional organizations including the American Association for Clinical Chemistry (AACC), and Mass Spectrometry & Advances in the Clinical Lab (MSACL).

Volunteers

The number of volunteers needed for the virtual conference was significantly higher than for the previous in-person R/Medicine conferences. For each short course we had at least one teaching assistant for every 10 participants. Additionally, we had someone monitoring the Zoom chat to escalate issues when necessary.

For the main session we had two moderators who alternated between speakers in order to orient the upcoming speaker in the green room. We also had a Crowdcast host who helped behind the scenes and moved participants from one presentation to the next. One person was also in charge of playing any of the talks that were pre-recorded. Each BoF room had a facilitator. Because this was a virtual conference, we also identified back-up volunteers in case there were power outages or issues with internet access. Pre-conference training and discussion sessions were held for short course teaching assistants. Pre-conference technical checks were held prior to the conference for the speakers, instructors, and volunteers.

The Conference

In the previous two years, attendance at each conference was around 150 participants. In 2020, 587 people registered for the conference, including 225 students, 185 academics, 104 from industry, and 73 who were working on COVID-19 related projects. Of these, 32% were from outside the United States coming from 43 countries (Figure 1). Of those who registered, 452 attended at least some sessions live and 431 watched some sessions via the replay feature in Crowdcast. The main conference was held over two days and included 4 keynote addresses, 16 regular talks, 11 lightning talks, and one panel discussion. The schedule included three BoF time slots, each with 4-5 different topics.

We conducted a brief post-conference survey, which garnered 163 responses. We included the question "How likely are you to recommend this conference to a friend or colleague?" with a 1-5 scale. The "net promoter score", which was calculated by subtracting the fraction of respondents rating the aforementioned question between 1-3 from the fraction of respondents rating it a 5, was 55%. Of all respondents, 98% indicated that they planned to attend R/Medicine in the future.

Short Courses

Pre-conference events included two short courses presented on the day prior to the conference. The first course, *Introduction to R for Clinicians*, is an adaptation of the popular *Welcome to the Tidyverse* course (Grolemund, 2019) for an audience of healthcare professionals and clinical researchers. The four-hour course placed heavy emphasis on reproducibility and using R Markdown as a computational document format and introduced fundamental concepts of data visualization and data transformation using the `tidyverse` set of tools (Wickham et al., 2019). Course materials (Kadauke, 2020b) and recordings (Kadauke, 2020c) are available online.

The second course, *Introduction to Machine Learning with Tidymodels*, was taught by Alison Hill and focused on the data scientist/statistician audience. This four-hour course aimed to provide a gentle

Figure 1: Location of those who registered for the 2020 R/Medicine conference

introduction to machine learning with R using the modern suite of predictive modeling packages called `tidymodels`. Participants practiced building, evaluating, comparing, and tuning predictive models interactively. The course introduced fundamental concepts including resampling, overfitting, the holdout method, the bias-variance trade-off, ensembling, cross-validation, and feature engineering; the course materials (Hill, 2020) is available online.

Attendance at the short courses was limited by request of the instructors. The *Introduction to R for Clinicians* class drew 143 participants, and 80 attended the *Introduction to Machine Learning with Tidymodels* course. Course recordings were available for anyone registered for the conference.

Scientific Program

We received 43 abstracts and accepted 25. Additionally, we held space for late-breaking COVID-19 related presentations and for two sponsors. The Scientific Program of the conference was divided into six sessions, each covering a broad theme.

R in Clinical Research/Clinical Trials: This session started with a presentation by Daniela Witten, in which the keynote speaker presented a theoretical framework for re-using data that was collected for testing pre-specified hypotheses to drive new hypothesis generation. Additional sessions focused on analyzing and reporting clinical trial data as well as outlier and anomaly detection in clinical trial data sets.

Collaboration/Reproducibility: The second session started with a keynote by Robert Gentleman in which he outlined the value of large, well-curated data sets as well as how the R ecosystem will be essential for developing new treatments as well as validating and deploying healthcare analytics and clinical decision support tools. Additional talks discussed the `{drake}` and `{holepunch}` packages.

New analysis approaches and packages: The last session of the first day started with a dazzling presentation by Travis Gerke and Garrick Aden-Buie in which the speakers discussed the development of internal R packages. An early prototype of a package for creating CONSORT diagrams was presented, as well as summaries of more mature packages including `{gtsummary}`, `{nDSPA}`, and `{treeheatr}`.

R in Clinical Practice, Education, and Bioethics: The first session of the second conference day started with a keynote speech by Ewen Harrison in which he highlighted the flexibility and ease of using R which makes it an ideal platform for clinician-researchers or other researchers who are not primarily programmers, as well as the collaborative nature of the R community. Additional presentations discussed the role of R in processing laboratory data in clinical practice as well as

teaching R to healthcare professionals. Finally, a team presentation by Joy Payton and Paulette McRae highlighted the problematic history of racism in medical research and practice, discussed root causes of bias in data, and offered suggestions to mitigate these biases.

Dashboards and Shiny Apps: This session discussed a variety of web applications developed using the Shiny framework that were being used either in clinical practice or clinical research.

COVID-19 research: The final session started with a keynote by Patrick Mathias, who shared his experience directing both the clinical operations and operational analytics efforts of one of the busiest clinical COVID-19 testing laboratories in the United States (Greenwich, 2020). Additional presentations highlighted the role R has played in coordinating the response to the COVID-19 pandemic.

Lessons learned

Overall, the conference succeeded well beyond our expectations. Some of the lessons we learned are as follows.

- The chat feature helped build community and allowed active participation in the conference. For instance, some presenters tag-teamed where one person presented and another answered questions via chat as they arose. Other presenters had pre-recorded their talk and were thus able to answer live questions during the playback of their presentation.
- Slack was essential for the behind-the-scenes coordination, especially as issues arose. Additionally, we had participants create their own channels and Zoom sessions for further discussion on certain topics.
- Having back-ups for all the major coordinating roles was essential. Fortunately we had limited issues, but one keynote speaker experienced a power outage, leading to an unavoidable delay.
- Although Crowdcast allows for smooth transitioning between speakers, time to transition between presenters needs to be built into the schedule, since presenters run over, technical issues emerge, and bio-breaks are important. It is also wise to build in some flex-time in case there are technical difficulties.
- During the presentations it was challenging to let speakers know when they were running out of time. All moderators downloaded a app to their phones that allowed them to play a doorbell sound to indicate to the speaker that their time is almost up, however that was not always effective. For 2021 we are planning to require that lightning talks be pre-recorded.
- Having a virtual meeting greatly expanded our reach. In 2019 we held the conference in Boston and had roughly 150 attendees from a handful of countries whereas in 2020 we had 452 attendees from 43 countries. We hope that the larger attendance will increase our impact as well as our ability to obtain sponsors for future conferences.
- Recruitment for high caliber keynote speakers was perhaps easier because the time commitment and travel effort were substantially less than for an in-person conference.

Sponsors

We would like to thank all the sponsors that supported the 2020 conference: American Association for Clinical Chemistry (AACC), Children's Hospital of Philadelphia (CHOP), Association for Mass Spectrometry & Advances in the Clinical Lab (MSACL), Procogia, the R Consortium, RStudio, and Yale School of Public Health.

Acknowledgements

As members of the R/Medicine 2020 Organizing and Programming Committees, we would like to thank all the volunteers who contributed to the success of the conference, the course instructors and teaching assistants, the keynote and contributed speakers, and all the attendees who dialed in from around the globe.

Links

Further information about R/Medicine 2020 and the contributions presented during the conference can be found at the conference website: https://events.linuxfoundation.org/r-medicine/

Bibliography

R/medicine 2020 youtube channel, 2020. URL https://www.youtube.com/playlist?list=PL4IzsxWztPdljYo7uE5G_R2PtYw3fUReo. [p643]

A. Greenwich. Interview with dr. alex greenwich, 2020. URL https://muckrack.com/broadcast/savedclips/view/ZfQEE5xUEW. [p645]

G. Grolemund. Welcome to the tidyverse, 2019. URL https://github.com/rstudio-education/welcome-to-the-tidyverse. [p643]

A. Hill. Tidymodels, virtually - course notes, 2020. URL https://github.com/rstudio-education/tidymodels-virtually. [p644]

S. Kadauke. Rstudio server pro training environment, 2020a. URL https://github.com/skadauke/rsp-train. [p643]

S. Kadauke. Intro to r for clinicians - course notes, 2020b. URL https://github.com/skadauke/intro-to-r-for-clinicians-rmed2020. [p643]

S. Kadauke. Intro to r for clinicians - recordings, 2020c. URL https://docs.google.com/document/d/e/2PACX-1vSSeD39D4_nxC4S7lVgyLx9mZjQiU5W96S_4r3Nne8fFVD4aoEiGwnjz0E7nEh-_YYtEyt-9HFWYB78/pub. [p643]

H. Wickham, M. Averick, J. Bryan, W. Chang, L. D. McGowan, R. François, G. Grolemund, A. Hayes, L. Henry, J. Hester, M. Kuhn, T. L. Pedersen, E. Miller, S. M. Bache, K. Müller, J. Ooms, D. Robinson, D. P. Seidel, V. Spinu, K. Takahashi, D. Vaughan, C. Wilke, K. Woo, and H. Yutani. Welcome to the tidyverse. *Journal of Open Source Software*, 4(43):1686, 2019. doi: 10.21105/joss.01686. URL https://doi.org/10.21105/joss.01686. [p643]

G. Wilson. *Teaching Tech Together*. Taylor and Francis, 2019. [p642]

Elizabeth J. Atkinson
Department of Quantitative Health Sciences, Mayo Clinic
200 First Street SW
Rochester, MN 55905

ORCiD: 0000-0002-1191-3775
atkinson@mayo.edu

Peter D. Higgins
Department of Internal Medicine, University of Michigan Health
1500 E Medical Center Dr.
Ann Arbor, MI 48109

phiggins@med.umich.edu

Denise Esserman
Department of Biostatistics, Yale School of Public Health
300 George Street, Ste 555
New Haven, CT 06511

denise.esserman@yale.edu

Michael J. Kane
Department of Biostatistics, Yale School of Public Health
300 George Street, Ste 555
New Haven, CT 06511

michael.kane@yale.edu

Steven J. Schwager
Cornell University
129 Garden Ave.

Ithaca, NY 14853

steven.schwager@cornell.edu

Joseph B. Rickert
R Consortium, RStudio
548 Market St.
San Francisco, CA 94104

joseph.rickert@rstudio.edu

Daniella Mark
Procogia
3600 136th Pl SE Suite 300
Bellevue, WA 98006

daniella@procogia.com

Mara Alexeev
Department of Biomedical Informatics, Boston Children's Hospital
10 Shattuck Street, Suite 514
Boston, MA 02115

mara.alexeev@gmail.com

Stephan Kadauke
Department of Pathology and Laboratory Medicine, Children's Hospital of Philadelphia, Perelman School of Medicine, University of Pennsylvania
3401 Civic Center Blvd.
Philadelphia, PA 19104

ORCiD: 0000-0003-2996-8034
kadaukes@chop.edu

Conference Report of Why R? Turkey 2021

by Mustafa Cavus, Olgun Aydin, Ozan Evkaya, Ozancan Ozdemir, Deniz Bezer, Ugur Dar

Abstract The Why R? Turkey 2021 as a three-day online conference was organized to bring together researchers and professionals from Turkey on April 16-17-18, 2021. We hereby aimed to promote the R community in Turkey by bringing R users with different backgrounds such as genetics, sociology, finance, economy, bio-statistics. There were 8 thematic sessions and 18 invited speakers. In this article, it is aimed to describe the preparation phase, technical details, and the impact of the conference on audience.

Why R? Turkey 2021

The Why R? Turkey 2021 is a pre-meeting of Why R? 2021 conference. Why R? conferences are international conference series organized annually by the Why R? Foundation since 2017. It is one of the largest annual R conferences in Central Europe (Burdukiewicz et al., 2019). In addition to the main conference, pre-meetings are held in different cities from all over the world. In 2017, four pre-meetings were organized in Poland. Thereafter, eleven pre-meetings across four different countries were held in 2018. In 2019, twelve pre-meetings took place in eight different countries. In 2020, in a pre-meeting framework, totally six organizations appeared in four different countries (Poland (2), Turkey (2), Ireland (1), and Germany (1)). Among those six pre-meetings, two of them were held in Istanbul (24.04.2020) and Ankara (27.04.2020) consecutively, as a one-day organization (Why R, 2020).

Figure 1: The logo of Why R? Turkey 2021.

Two primary goals of Why R ? Turkey 2021 are;

1. to bring together Turkish R users from all over the world
2. to broaden the horizon of the students with respect to the diverse disciplines

For this purpose, experienced researchers, who are R users, from various disciplines were invited to the conference as a speaker. The conference program was designed comprehensively to cover various research fields such as bio-statistics, sociology, educational sciences, psychology, molecular biology, genomics, football analytics, and economics. The detailed program is shown in Table 1.

Participants

There were 2180 registered participants, 60% were students and 40% were professionals who work for governmental institutions, companies from the private sector, and academicians from the universities in Turkey. With regards to the education level of the student participants, 50% of them was graduate and rest was undergraduate students. 60% of the students study Statistics. The number of unique participants for the conference is 880. Additionally, the share of each thematic session is counted as 540, 435, 380, 355, 390, 360, respectively. The whole event was organized and moderated by the organization committee including 6 members.

Most of the participants were registered from İstanbul, Ankara, and İzmir, respectively. In the conference, we had participants from 73 provinces out of 81 provinces in Turkey. The distribution of registered participants by provinces is illustrated in Figure 2.

Figure 2: The Distribution of Participants by Provinces

In addition to Turkey, we had participants from the USA, England, Belgium, Sweden, Switzerland, Qatar, Nederland, Italy, France, and Spain. In that respect, the conference was successful to bring Turkish R users from different countries as well.

Conference Program

The detailed conference program, shown in Table 1, consists of 6 main sessions over three days. For each speaker, there were 30 minutes to complete his/her presentation and thereafter the participants are guided to the breakout rooms in the Zoom platform for Q&A session with the speaker.

Promotion of the Event and Reactions

The social media channels (Instagram, Twitter, Facebook and, LinkedIn) were used for the promotion of the conference. Conference social media accounts were created to conduct the promotion except LinkedIn. Organization committee members' personal LinkedIn accounts were preferred to promote the event.

Instagram ads were used to reach out bigger audience. Advertisements went live two weeks before starting date of the conference and finished 4 days before the conference started. At the end of this period, we obtained more than 500 followers most of whom are young adults as shown in Figure 3. Besides, the Instagram account of the conference was visited more than 2.000 times during the advertisement period.

In addition to Instagram, Twitter was also used efficiently for the promotion. The Twitter account for this event was created in January, but the posts started to be shared at the end of February. To reach R users, all contents were posted using the `#Rstats` hashtag. As of May 8, the Twitter account of the conference was visited by more than 18000 users, and tweets were viewed more than 174000 times.

According to our calendar, we started to share our posts once a week on Twitter. Although post frequency was lower at the beginning, the number of followers reached over three hundreds quickly. However, the account gained the highest number of followers in April. (Figure 4).

Thus, visits and view statistics of the Twitter account show that Twitter was the most efficient platform to reach R users.

Technical Solutions

The official website was developed and used as the primary information source for the conference. The website was hosted on Why R? Foundation's domain. This allowed us to be more visible on Google Search. The abstract book was prepared using the bookdown R package and was served via the website.

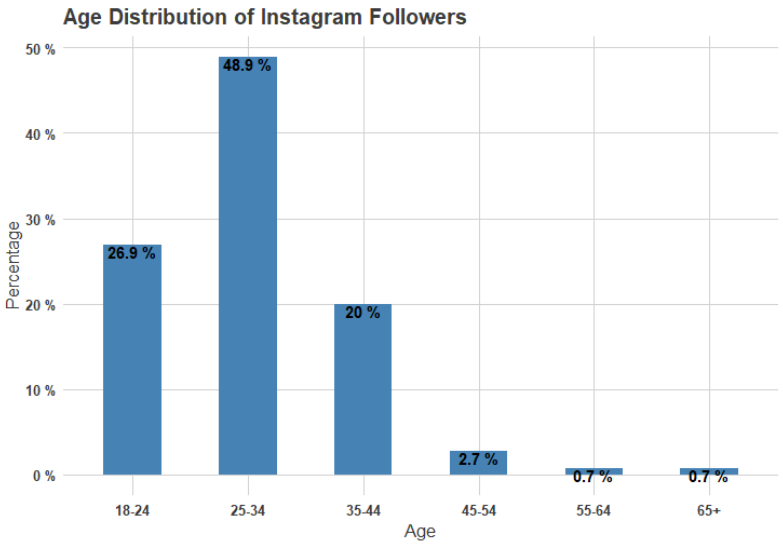


Figure 3: Age Distribution of Instagram Followers

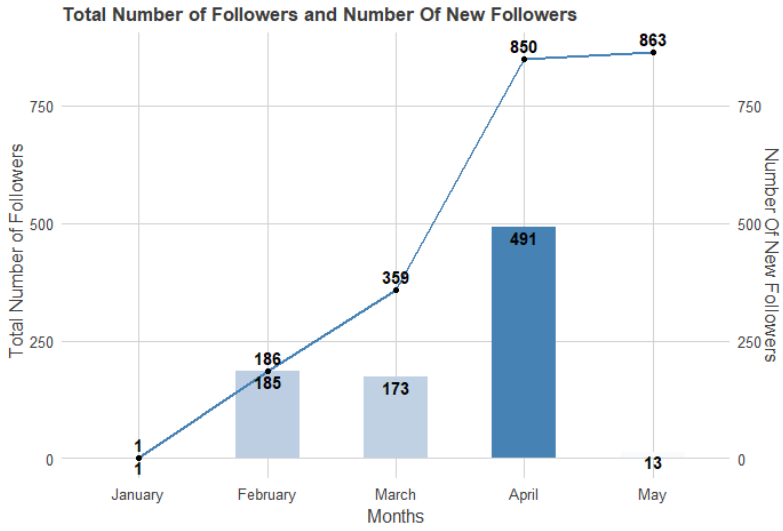


Figure 4: Evolution of the number of followers of the WhyR? Turkey 2021 Twitter Account

Instant updates in terms of program, speakers, program, schedule, were shared with the audience via Instagram, and Twitter accounts, dedicated to the conference. Twitter and Instagram account caught the attention of people interested in participating in the conference very quickly.

Google Forms was preferred for registration purposes and used as a contact form. Hyperlinks to the registration form and contact form were added to the website. During registration, participants were asked to answer questions regarding their professional background, education level, R knowledge. Thanks to this, we had a chance to better understand participants' profiles.

All speeches were streamed on Zoom. Premium zoom account was purchased to be able to have 1000 participants online at the same time. Separate zoom meetings were set for each session. After each talk, dedicated breakout rooms were created and Q&A sessions were conducted in breakout rooms. This allowed the audience to have more interaction with speakers. The cost of the Zoom account was covered by the funding received from Why R? Foundation and R Consortium. URLs for the recordings, as well as slide decks and R scripts used by speakers during presentations, shared publicly on the GitHub repository of the conference.

Summary

The three-day online conference of Why R? Turkey 2021 aimed to bring R users from different fields and make a connection between the Turkish R users and possible future users from various disciplines. In that respect, the attained total number of participants, the fruitful Q&A sessions and the overall positive attitudes from all speakers/participants revealed that the meeting reached its main objective. To sum up, Why R? Turkey 2021 is the best comprehensive online meeting held for Turkish R users and it promises both national and international organizations in the near future.

Acknowledgment

We would like to thank all the sponsors that support the organization of Why R? Turkey 2021: Why R? Foundation, R Consortium, and StickerMule.

Additional Information

Further information about Why R? Turkey 2021 and the contributions presented during the conference can be found at the following links:

- Website: http://whyr.pl/2021/turkey/
- Twitter: https://twitter.com/whyr2021turkey
- YouTube Channel: https://www.youtube.com/channel/UCr8qT7gK9WQZjCNz7Agi7sA
- Materials: https://github.com/whyr2021turkey
- Abstract Book: http://whyr.pl/2021/turkey/abstract_book/

Bibliography

M. Burdukiewicz, F. Pietluch, J. Chilimoniuk, K. Sidorczuk, D. Rafacz, L. Jessen, S. Rödiger, M. Kosinski, and P. Wojcik. Conference report: Why r? 2019. *R Journal*, 12(1):484–493, 2019. URL https://doi.org/10.1080/10618600.1996.10474713. [p648]

F. Why R. 2020. URL http://whyr.pl/foundation/events/#2020. [p648]

Mustafa Cavus
Eskisehir Technical University
Department of Statistics
Eskisehir, Turkey
https://orcid.org/0000-0002-6172-5449
mustafacavus@eskisehir.edu.tr

Olgun Aydın
Gdańsk University of Technology
Gdańsk, Poland
Department of Statistics
https://orcid.org/0000-0002-7090-0931
olgun.aydin@pg.edu.pl

Ozan Evkaya
Padova University
Department of Statistical Sciences
Padova, Italy
https://orcid.org/0000-0002-5076-8144
ozanevkaya@gmail.com

Ozancan Ozdemir
Middle East Technical University
Department of Statistics
Ankara, Turkey
https://orcid.org/0000-0002-7850-3885
ozancan@metu.edu.tr

Deniz Bezer
Middle East Technical University
Department of Statistics
Ankara, Turkey
deniz.bezer@metu.edu.tr

Ugur Dar
Eskisehir Technical University
Department of Statistics
Eskisehir, Turkey
ugurdar@eskisehir.edu.tr

Session	Speaker	Title
1	Gökmen Zararsız	Using R in Medicine and Diagnostic Domains: The Case of the Alzheimer Project
	Dinçer Göksülük	BioSoft: A cloud-based platform for Biostatistics/Bioinformatics softwares developed using R programming language
	Erdal Coşgun	Genetic research on cloud-based systems: The Bioconductor experience
2	Tuba Bircan	R for whom?
	Gökhan Kaya	Data Analysis and using theory in R for sociological researches: Opportunities and challenges
	Dilek Yıldız	Using R in demography
	Ali O. İlhan	R and bibliometric analysis: a brief introduction
3	Kadir Kızılkaya	Using R for animal breeding and genomic selection
	Hilal Özkılınç	Using R in genomics studies
	Burcu Mestav	Using R in aquaculture and ecology
4	Mehmet Can Demir	Research studies in educational science using R
	Kübra Atalay Kabasakal	R packages for educational science
	Eren Halil Özberk	Using R in psychology
5	Yeşim Güney	Structural Breakpoints in Turkey COVID-19 data
	İpek Güler	R packages for joint modeling of survival and recurrent data
	Emre Toros	Soccer, data, why, how, and R
6	Burak Saltoğlu	Behavioral financial analysis by using R
	Ahmet Akgül	Regression analysis in R: house price prediction model for Istanbul
	Ayhan Yüksel	Algoritmic trade using R

Table 1: Program of the Why R? Turkey 2021