The R Journal

Volume 12/2, December 2020

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

Contents

Editorial 4

Contributed Research Articles

The **biglasso** Package: A Memory- and Computation-Efficient Solver for Lasso Model Fitting with Big Data in R 6
Six Years of Shiny in Research - Collaborative Development of Web Tools in R 20
A Fast and Scalable Implementation Method for Competing Risks Data with the R Package fastcmprsk 43
ordinalClust: An R Package to Analyze Ordinal Data 61
KSPM: A Package For Kernel Semi-Parametric Models 82
Comparing Multiple Survival Functions with Crossing Hazards in R 107
A Unified Algorithm for the Non-Convex Penalized Estimation: The **ncpen** Package . 120
TULIP: A Toolbox for Linear Discriminant Analysis with Penalties 134
fitzRoy - An R Package to Encourage Reproducible Sports Analysis. 155
Assembling Pharmacometric Datasets in R - The **puzzle** Package 163
RNGforGPD: An R Package for Generation of Univariate and Multivariate Generalized Poisson Data 173
Testing the Equality of Normal Distributed and Independent Groups' Means Under Unequal Variances by **doex** Package 189
AQuadtree: an R Package for Quadtree Anonymization of Point Data. 209
miWQS: Multiple Imputation Using Weighted Quantile Sum Regression 226
User-Specified General-to-Specific and Indicator Saturation Methods 251
Kuhn-Tucker and Multiple Discrete-Continuous Extreme Value Model Estimation and Simulation in R: The **rmdcev** Package 266
NTS: An R Package for Nonlinear Time Series Analysis 293
Species Distribution Modeling using Spatial Point Processes: a Case Study of Sloth Occurrence in Costa Rica 311
A Graphical EDA Tool with **ggplot2**: **brinton** 322
MoTBFs: An R Package for Learning Hybrid Bayesian Networks Using Mixtures of Truncated Basis Functions 343
Analyzing Basket Trials under Multisource Exchangeability Assumptions 360

OpenLand: Software for Quantitative Analysis and Visualization of Land Use and Cover Change . . . 373
FarmTest: An R Package for Factor-Adjusted Robust Multiple Testing . . . 389

News and Notes

Changes in R 3.6–4.0 . . . 403
Changes on CRAN . . . 408
News from the Bioconductor Project . . . 411
R Foundation News . . . 412
News from the Forwards Taskforce . . . 414
e-Rum2020: how we turned a physical conference into a successful virtual event . . . 417

Editorial

by Michael J. Kane

On behalf of the editorial board, I am pleased to present Volume 12 Issue 2 of the R Journal. This is my third and final issue as the Editor-in-Chief. In the last year, we have made some substantial changes to the journal that I believe will continue to increase our capacity to support the growing data science and computational statistics communities, and continue to raise the visibility of the journal. In the last few months we recruited 10 Associate Editors and we are continuing the recruitment process. I'd like to publicly welcome our new Associate Editors, and thank each of them for joining us, and for their contributions thus far to the journal.

We have also been making substantial improvements to the R Journal infrastructure, allowing us to more efficiently usher manuscripts through the review process. This effort has been made possible through an investment by the R Consortium. Thanks very much to Di Cook, Mitchell O'Hara-Wild, and Stephanie Kobakian for the new capabilities - they have made my job a lot easier.

I'd also like to welcome Di as the new Editor-in-Chief of the journal. She has been an instrumental member of the editorial team, she has provided me with insight and guidance with regard to the journal. I look forward to seeing how the journal progresses under her direction.

In this issue

News from the R Foundation and CRAN are included in this issue along an update on the e-Rum2020 conference that was held earlier. In addition, this issue features 23 contributed research articles that have been categorized below.

Papers focusing on health and clinical trial data

- A Fast and Scalable Implementation Method for Competing Risks Data with the R Package **fastcmprsk**
- Assembling Pharmacometric Datasets in R - The **puzzle** Package
- Analyzing Basket Trials under Multisource Exchangeability Assumptions
- Comparing multiple survival functions with crossing hazards in R

Supervised and unsupervised model fitting

- The **biglasso** Package: A Memory- and Computation-Efficient Solver for Lasso Model Fitting with Big Data in R
- User-Specified General-to-Specific and Indicator Saturation Methods
- **miWQS**: Multiple Imputation Using Weighted Quantile Sum Regression
- **NTS**: An R Package for Nonlinear Time Series Analysis
- **ordinalClust**: An R Package to Analyse Ordinal Data
- **TULIP**: A Toolbox for Linear Discriminant Analysis with Penalties
- A Unified Algorithm for the Non-Convex Penalized Estimation: The **ncpen** Package
- **KSPM**: A Package For Kernel Semi-Parametric Models

Probability distributions and processes

- Testing the Equality of Normally Distributed Groups' Means Under Unequal Variances by **doex** Package

- **MoTBFs**: An R Package for Learning Hybrid Bayesian Networks Using Mixtures of Truncated Basis Functions
- Kuhn-Tucker and Multiple Discrete-Continuous Extreme Value Model Estimation and Simulation in R: The **rmdcev** Package
- Species Distribution Modeling using Spatial Point Processes: a Case Study of Sloth Occurrence in Costa Rica
- **AQuadtree**: an R Package for Quadtree Anonymization of Point Data
- **RNGforGPD**: An R Package for Generation of Univariate and Multivariate Generalized Poisson Data
- **FarmTest**: An R Package for Factor-Adjusted Robust Multiple Testing

Visualization, reproducibilty, and collaboration

- A Graphical EDA Tool with **ggplot2**: **brinton**
- Six Years of Shiny in Research; Collaborative Development of Web Tools in R
- **fitzRoy**: An R Package to Encourage Reproducible Sports Analysis
- **OpenLand**: Software for Quantitative Analysis and Visualization of Land Use and Cover Change

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

The biglasso Package: A Memory- and Computation-Efficient Solver for Lasso Model Fitting with Big Data in R

by Yaohui Zeng and Patrick Breheny

Abstract Penalized regression models such as the lasso have been extensively applied to analyzing high-dimensional data sets. However, due to memory limitations, existing R packages like **glmnet** and **ncvreg** are not capable of fitting lasso-type models for ultrahigh-dimensional, multi-gigabyte data sets that are increasingly seen in many areas such as genetics, genomics, biomedical imaging, and high-frequency finance. In this research, we implement an R package called **biglasso** that tackles this challenge. **biglasso** utilizes memory-mapped files to store the massive data on the disk, only reading data into memory when necessary during model fitting, and is thus able to handle out-of-core computation seamlessly. Moreover, it's equipped with newly proposed, more efficient feature screening rules, which substantially accelerate the computation. Benchmarking experiments show that our **biglasso** package, as compared to existing popular ones like **glmnet**, is much more memory- and computation-efficient. We further analyze a 36 GB simulated GWAS data set on a laptop with only 16 GB RAM to demonstrate the out-of-core computation capability of **biglasso** in analyzing massive data sets that cannot be accommodated by existing R packages.

Introduction

The lasso model proposed by Tibshirani (1996) has fundamentally reshaped the landscape of high-dimensional statistical research. Since its original proposal, the lasso has attracted extensive studies with a wide range of applications to many areas, such as signal processing (Angelosante and Giannakis, 2009), gene expression data analysis (Huang and Pan, 2003), face recognition (Wright et al., 2009), text mining (Li et al., 2015) and so on. The great success of the lasso has made it one of the most popular tools in statistical and machine-learning practice.

Recent years have seen the evolving era of Big Data where ultrahigh-dimensional, large-scale data sets are increasingly seen in many areas such as genetics, genomics, biomedical imaging, social media analysis, and high-frequency finance (Fan et al., 2014). Such data sets pose a challenge to solving the lasso efficiently in general, and for R specifically, since R is not naturally well-suited for analyzing large-scale data sets (Kane et al., 2013). Thus, there is a clear need for scalable software for fitting lasso-type models designed to meet the needs of big data.

In this project, we develop an R package, **biglasso** (Zeng and Breheny, 2016), to extend lasso model fitting to Big Data in R. Specifically, sparse linear and logistic regression models with lasso and elastic net penalties are implemented. The most notable features of **biglasso** include:

- It utilizes memory-mapped files to store the massive data on the disk, only loading data into memory when necessary during model fitting. Consequently, it's able to seamlessly handle out-of-core computation.
- It is built upon pathwise coordinate descent algorithm and "warm start" strategy, which has been proven to be one of fastest approaches to solving the lasso (Friedman et al., 2010).
- We develop new, hybrid feature screening rules that outperform state-of-the-art screening rules such as the sequential strong rule (SSR) (Tibshirani et al., 2012), and the sequential EDPP rule (SEDPP) (Wang et al., 2015) with additional 1.5x to 4x speedup.
- The implementation is designed to be as memory-efficient as possible by eliminating extra copies of the data created by other R packages, making **biglasso** at least 2x more memory-efficient than **glmnet**.
- The underlying computation is implemented in C++, and parallel computing with OpenMP is also supported.

The methodological innovation and well-designed implementation have made **biglasso** a much more memory- and computation-efficient and highly scalable lasso solver, as compared to existing popular R packages like **glmnet** (Friedman et al., 2010), **ncvreg** (Breheny and Huang, 2011), and **picasso** (Ge et al., 2015). More importantly, to the best of our knowledge, **biglasso** is the first R package that enables the user to fit lasso models with data sets that are larger than available RAM, thus allowing for powerful big data analysis on an ordinary laptop.

Method

Memory mapping

Memory mapping (Bovet and Cesati, 2005) is a technique that maps a data file into the virtual memory space so that the data on the disk can be accessed as if they were in the main memory. Technically, when the program starts, the operating system (OS) will cache the data into RAM. Once the data are in RAM, the computation is at the standard in-memory speed. If the program requests more data after the memory is fully occupied, which is inevitable in the data-larger-than-RAM case, the OS will move data that is not currently needed out of RAM to create space for loading in new data. This is called the *page-in-page-out* procedure, and is automatically handled by the OS.

The memory mapping technique is commonly used in modern operating systems such as Windows and Unix-like systems due to several advantages:

(1) it provides faster file read/write than traditional I/O methods since data-copy from kernel to user buffer is not needed due to page caches;

(2) it allows random access to the data as if it were in the main memory even though it physically resides on the disk;

(3) it supports concurrent sharing in that multiple processes can access the same memory-mapped data file simultaneously, making parallel computing easy to implement in data-larger-than-RAM cases;

(4) it enables out-of-core computing thanks to the automatic page-in-page-out procedure.

We refer the readers to Rao et al. (2010), Lin et al. (2014), and Bovet and Cesati (2005) for detailed techniques and some successful applications of memory mapping.

To take advantage of memory mapping, **biglasso** creates memory-mapped big matrix objects based upon the R package **bigmemory** (Kane et al., 2013), which uses the Boost C++ library and implements memory-mapped big matrix objects that can be directly used in R. Then at the C++ level, **biglasso** uses the C++ library of **bigmemory** for underlying computation and model fitting.

Efficient feature screening

Another important contribution of **biglasso** is our newly developed *hybrid safe-strong rule*, named SSR-BEDPP, which substantially outperforms existing state-of-the-art ones in terms of the overall computing time of obtaining the lasso solution path. Here, we describe the main idea of hybrid rules; for the technical details, see Zeng et al. (2021).

Feature screening aims to identify and discard inactive features (i.e., those with zero coefficients) from the lasso optimization. It often leads to dramatic dimension reduction and hence significant computation savings. However, these savings will be negated if the screening rule itself is too complicated to execute. Therefore, an efficient screening rule needs to be powerful enough to discard a large portion of features and also relatively simple to compute.

Existing screening rules for the lasso can be divided into two types: (1) heuristic rules, such as the sequential strong rule (SSR) (Tibshirani et al., 2012), and (2) safe rules, such as the basic and the sequential EDPP rules (Wang et al., 2015), denoted here as BEDPP and SEDPP respectively. Safe rules, unlike heuristic ones, are guaranteed to never incorrectly screen a feature with a nonzero coefficient. Figure 1 compares the power of the three rules in discarding features. SSR, though most powerful among the three, requires a cumbersome post-convergence check to verify that it has not incorrectly discarded an active feature. The SEDPP rule is both safe and powerful, but is inherently complicated and time-consuming to evaluate. Finally, BEDPP is the least powerful, and discards virtually no features when λ is smaller than 0.45 (in this case), but is both safe and involves minimal computational burden.

The rule employed by **biglasso**, SSR-BEDPP, as its name indicates, combines SSR with the simple yet safe BEDPP rule. The rationale is to alleviate the burden of post-convergence checking for strong rules by not checking features that can be safely eliminated using BEDPP. This hybrid approach leverages the advantages of each rule, and offers substantial gains in efficiency, especially when solving the lasso for large values of λ.

Table 1 summarizes the complexities of the four rules when applied to solving the lasso along a path of K values of λ for a data set with n instances and p features. SSR-BEDPP can be substantially faster than the other three rules when BEDPP is effective. Furthermore, it is important to note that SSR (with post-convergence checking) and SEDPP have to scan the entire feature matrix at every value of λ, while SSR-BEDPP only needs to scan the features not discarded by BEDPP. This advantage

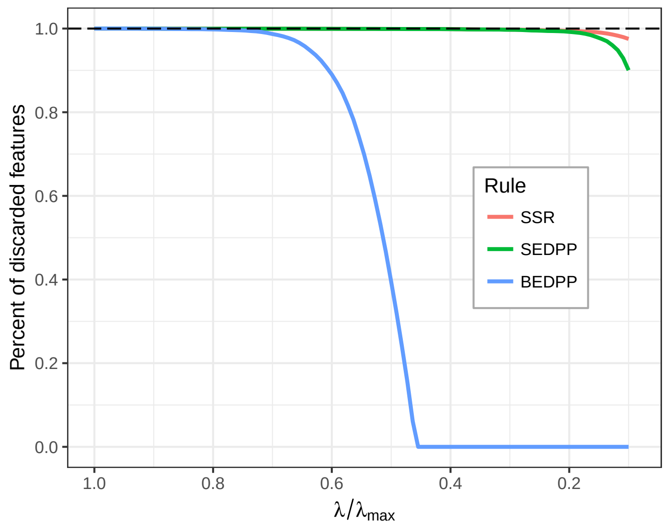


Figure 1: Percent of features discarded.

of SSR-BEDPP is particularly appealing in out-of-core computing, where fully scanning the feature matrix requires disk access and therefore becomes the computational bottleneck of the procedure.

Rule	Complexity
SSR	$O(npK)$
SEDPP	$O(npK)$
BEDPP	$O(np)$
SSR-BEDPP	$O(n\sum_k^K \lvert\mathcal{S}_k\rvert))$

Table 1: Complexity of computing screening rules along the entire path of K values of λ. $|\mathcal{S}_k|$ denotes the cardinality of $\mathcal{S}_k$, the safe set of features not discarded by BEDPP screening.

The hybrid screening idea is straightforward to extend to other lasso-type problems provided that a corresponding safe rule exists. For the **biglasso** package, we also implemented a hybrid screening rule, SSR-Slores, for lasso-penalized logistic regression by combining SSR with the so-called Slores rule (Wang et al., 2014), a safe screening rule developed for sparse logistic regression.

Implementation

Memory-efficient design

In penalized regression models, the feature matrix $\mathbf{X} \in \mathbb{R}^{n\times p}$ is typically standardized to ensure that the penalty is applied uniformly across features with different scales of measurement. In addition, standardization contributes to faster convergence of the optimization algorithm. In existing R packages such as **glmnet**, **ncvreg**, and **picasso**, a standardized feature matrix $\widetilde{\mathbf{X}}$ is calculated and stored, effectively doubling memory usage. This problem is compounded by cross-validation, where these packages also calculate and store additional standardized and unstandardized copies of $\mathbf{X}$ for each fold. This approach does not scale up well for big data.

To make the memory usage more efficient, **biglasso** doesn't store $\widetilde{\mathbf{X}}$. Instead, it saves only the means and standard deviations of the columns of $\mathbf{X}$ as two vectors, denoted as $\mathbf{c}$ and $\mathbf{s}$. Then wherever $\widetilde{x}_{ij}$ is needed, it is retrieved by "cell-wise standardization", i.e., $\widetilde{x}_{ij} = (x_{ij} - c_j)/s_j$. Additionally, the estimated coefficient matrix is sparse-coded in C++ and R to save memory space.

Simplification of computations

Cell-wise standardization saves a great deal of memory space, but at the expense of computational efficiency. To minimize this, **biglasso** uses a number of computational strategies to eliminate redundant calculations.

We first note that the computations related to $\widetilde{\mathbf{X}}$ during whole model fitting process are mainly of three types, and all can be simplified so that naïve cell-wise standardization can be avoided:

(1) $\widetilde{\mathbf{x}}_j^\top \widetilde{\mathbf{x}}_* = \sum_i \frac{x_{ij}-c_j}{s_j} \frac{x_{i*}-c_*}{s_*} = \frac{1}{s_j s_*}\left(\sum_i x_{ij}x_{i*} - nc_jc_*\right);$

(2) $\widetilde{\mathbf{x}}_i^\top \mathbf{y} = \sum_i \frac{x_{ij}-c_j}{s_j} y_i = \frac{1}{s_j}\left(\sum_i x_{ij}y_i - c_j \sum_i y_i\right);$

(3) $\widetilde{\mathbf{x}}_j^\top \mathbf{r} = \sum_i \frac{x_{ij}-c_j}{s_j} r_i = \frac{1}{s_j}\left(\sum_i x_{ij}r_i - c_j \sum_i r_i\right);$

where $\widetilde{\mathbf{x}}_j$ is the jth column of $\widetilde{\mathbf{X}}$, $\widetilde{\mathbf{x}}_*$ is the column corresponding to $\lambda_{\max}$, $\mathbf{y}$ is the response vector, and $\mathbf{r} \in \mathbb{R}^n$ is current residual vector.

Type (1) and (2) are used only for initial feature screening, and require only one-time execution. Type (3) occurs in both the coordinate descent algorithm and the post-convergence checking. Since the coordinate descent algorithm is fast to converge and only iterates over features in the active set $\mathcal{A}$ of nonzero coefficients, whose size is much smaller than p, the number of additional computations this introduces is small. Moreover, we pre-compute and store $\sum_i r_i$, which saves a great deal of computation during post-convergence checking since $\mathbf{r}$ does not change during this step. As a result, our implementation of cell-wise standardization requires only $O(p)$ additional operations compared to storing the entire standardized matrix.

Scalable cross-validation

Cross-validation is integral to lasso modeling in practice, as it is by far the most common approach to choosing λ. It requires splitting the data matrix $\mathbf{X}$ into training and test sub matrices, and fitting the lasso model multiple times. This procedure is also memory-intensive, especially if performed in parallel.

Existing lasso-fitting R packages split $\mathbf{X}$ using the "slicing operator" directly in R (e.g., `X[1:1000,]`). This introduces a great deal of overhead and hence is quite slow when $\mathbf{X}$ is large. Worse, the training and test sub-matrices must be saved into memory, as well as their standardized versions, all of which result in considerable memory consumption.

In contrast, **biglasso** implements a much more memory-efficient cross-validation procedure that avoids the above issues. The key design is that the main model-fitting R function allows a subset of $\mathbf{X}$, indicated by the row indices, as input. To cope with this design, all underlying C++ functions are enabled to operate on a subset of $\mathbf{X}$ given a row-index vector is provided.

Consequently, instead of creating and storing sub-matrices, only the indices of the training/test sets and the descriptor of $\mathbf{X}$ (essentially, an external pointer to $\mathbf{X}$) are needed for parallel cross validation thanks to the concurrency of memory-mapping. The net effect is that only one memory-mapped data matrix $\mathbf{X}$ is needed for K-fold parallel cross-validation, whereas other packages need up to $2K$ copies of $\mathbf{X}$: a copy and a standardized copy for each fold.

Parallel computation

Another important feature of **biglasso** is its parallel computation capability. There are two types of parallel computation implemented in **biglasso**.

At the C++ level, single model fitting (as opposed to cross validation) is parallelized with OpenMP. Though the pathwise coordinate descent algorithm is inherently sequential and thus does not lend itself to parallelization, several components of the algorithm (computing $\mathbf{c}$ and $\mathbf{s}$, matrix-vector multiplication, post-convergence checking, feature screening, etc.) do, and are parallel-enabled in **biglasso**.

Parallelization can also be implemented at the R level to run cross-validation in parallel. This implementation is straightforward and also implemented by **ncvreg** and **glmnet**. However, as mentioned earlier, the parallel implementation of **biglasso** is much more memory- and computation-efficient by avoiding extra copies and the overhead associated with copying data to parallel workers. Note that when cross-validation is run in parallel in R, parallel computing at C++ level for single model-fitting is disabled to avoid nested parallelization.

Benchmarking experiments

In this section, we demonstrate that our package **biglasso** (1.2-3) is considerably more efficient at solving for lasso estimates than existing popular R packages **glmnet** (2.0-5), **ncvreg** (3.9-0), and **picasso** (0.5-4). Here we focus on solving lasso-penalized linear and logistic regression, respectively, over the entire path of 100 λ values which are equally spaced on the scale of $\lambda/\lambda_{\max}$ from 0.1 to 1. To ensure a fair comparison, we set the convergence thresholds to be equivalent across all four packages. All experiments are conducted with 20 replications, and the average computing times (in seconds) are reported. The benchmarking platform is a MacBook Pro with Intel Core i7 @ 2.3 GHz and 16 GB RAM.

Memory efficiency

To demonstrate the improved memory efficiency of **biglasso** compared to existing packages, we simulate a feature matrix with dimensions $1,000 \times 100,000$. The raw data is 0.75 GB, and stored on the hard drive as an R data file and a memory-mapped file. We used `Syrupy`[1] to measure the memory used in RAM (i.e., the resident set size, RSS) every 1 second during lasso-penalized linear regression model fitting by each of the packages.

The maximum RSS during the model fitting is reported in Table 2. In the single fit case, **biglasso** consumes 0.84 GB memory in RAM, 50% of that used by **glmnet** and 22% of that used by **picasso**. Note that the memory consumed by **glmnet**, **ncvreg**, and **picasso** are respectively 2.2x, 2.1x, and 5.1x larger than the size of the raw data.

More strikingly, **biglasso** does not require additional memory to perform cross-validation, unlike other packages. For serial 10-fold cross-validation, **biglasso** requires just 27% of the memory used by **glmnet** and 23% of that used by **ncvreg**, making it 3.6x and 4.3x more memory-efficient than **glmnet** and **ncvreg**, respectively.

The memory savings offered by **biglasso** would be even more significant if cross-validation were conducted in parallel. However, measuring memory usage across parallel processes is not straightforward and not implemented in `Syrupy`.

Package	picasso*	ncvreg	glmnet	biglasso
Single fit	3.84	1.60	1.67	0.84
10-fold CV (1 core)	-	3.74	3.18	0.87

* Cross-validation is not implemented in **picasso**.

Table 2: The maximum RSS (in GB) for a single fit and 10 fold cross-validation (CV) with the raw data of 0.75 GB.

Computational efficiency: Linear regression

Simulated data

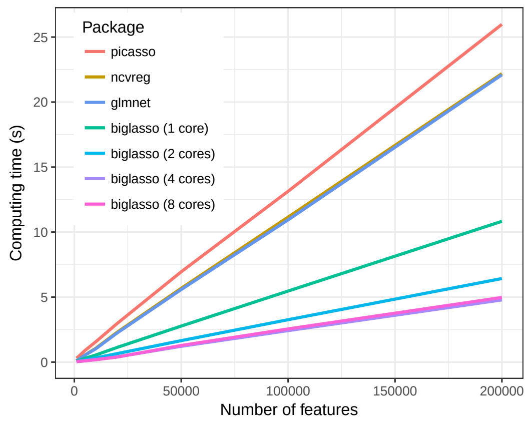
We now show with simulated data that **biglasso** is more scalable in both n and p (i.e., number of instances and features). We adopt the same model in Wang et al. (2015) to simulate data: $\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + 0.1\boldsymbol{\epsilon}$, where $\mathbf{X}$ and $\boldsymbol{\epsilon}$ are i.i.d. sampled from $N(0,1)$. We consider two different cases: (1) Case 1: varying p. We set $n = 1,000$ and vary p from 1,000 to 20,000. We randomly select 20 true features, and sample their coefficients from Unif[-1, 1]. After simulating $\mathbf{X}$ and $\boldsymbol{\beta}$, we then generate $\mathbf{y}$ according to the true model; (2) Case 2: varying n. We set $p = 10,000$ and vary n from 200 to 20,000. $\boldsymbol{\beta}$ and $\mathbf{y}$ are generated in the same way as in Case 1.

Figure 2 compares the mean computing time of solving the lasso over a sequence of 100 λ values by the four packages. In all the settings, **biglasso** (1 core) is uniformly 2x faster than **glmnet** and **ncvreg** (which overlap in the figure), and 2.5x faster than **picasso**. Moreover, the computing time of **biglasso** can be further reduced by half via parallel-computation of 4 cores. Using 8 cores doesn't help due to the increased overhead of communication between cores.

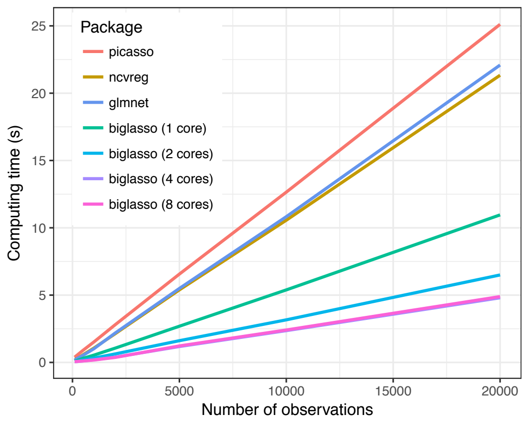
Real data

In this section, we compare the performance of the packages using diverse real data sets: (1) Breast cancer gene expression data[2] (GENE); (2) MNIST handwritten image data (MNIST) (LeCun et al., 1998); (3) Cardiac fibrosis genome-wide association study data (GWAS) (Breheny, 2016); and (4) Subset of New York Times bag-of-words data (NYT) (Dheeru and Karra Taniskidou, 2017). Note that for data sets MNIST and NYT, a different response vector is randomly sampled from a test set at each replication.

[1]`https://github.com/jeetsukumaran/Syrupy`
[2]`http://myweb.uiowa.edu/pbreheny/data/bcTCGA.html`



(a) Varying p, $n = 1,000$.



(b) Varying n, $p = 10,000$.

Figure 2: Mean computing time (in seconds) of solving the lasso over a sequence 100 λ values as a function of p (Left) and n (Right).

The size of the feature matrices and the average computing times are summarized in Table 3. In all four settings, **biglasso** was fastest at obtaining solutions, providing 2x to 3.8x speedup compared to **glmnet** and **ncvreg**, and 2x to 4.6x speedup compared to **picasso**.

Package	GENE	MNIST	GWAS	NYT
	$n = 536$	$n = 784$	$n = 313$	$n = 5,000$
	$p = 17,322$	$p = 60,000$	$p = 660,495$	$p = 55,000$
picasso	1.50 (0.01)	6.86 (0.06)	34.00 (0.47)	44.24 (0.46)
ncvreg	1.14 (0.02)	5.60 (0.06)	31.55 (0.18)	32.78 (0.10)
glmnet	1.02 (0.01)	5.63 (0.05)	23.23 (0.19)	33.38 (0.08)
biglasso	0.54 (0.01)	1.48 (0.10)	17.17 (0.11)	14.35 (1.29)

Table 3: Mean (SE) computing time (seconds) for solving the lasso along a sequence of 100 λ values.

Computational efficiency: Logistic regression

Simulated data

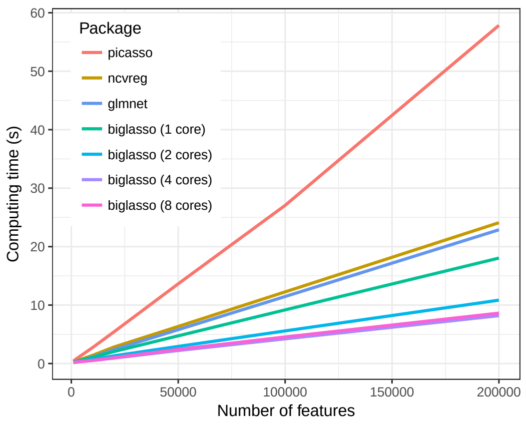
Similar to Section 4.2, here we first illustrate that **biglasso** is faster than other packages in fitting the logistic regression model with simulated data. The true data-generating model is: $y_i \sim Bin(1, prob)$; $\text{logit}(prob) = \mathbf{x}_i\boldsymbol{\beta}$, where each entry of $\mathbf{x}_i$ is i.i.d. sampled from standard Gaussian distribution. Again, two cases – varying p and varying n – are considered. 20 true features are randomly chosen and their coefficients are sampled from Unif[-1, 1].

Figure 3 summarizes the mean computing times of solving the lasso-penalized logistic regression over a sequence of 100 values of λ by the four packages. In all the settings, **biglasso** (1 core) is around 1.5x faster than **glmnet** and **ncvreg** (which again largely overlap), and more than 3x faster than **picasso**. Parallel computing with 4 cores using **biglasso** reduces the computing time by half.

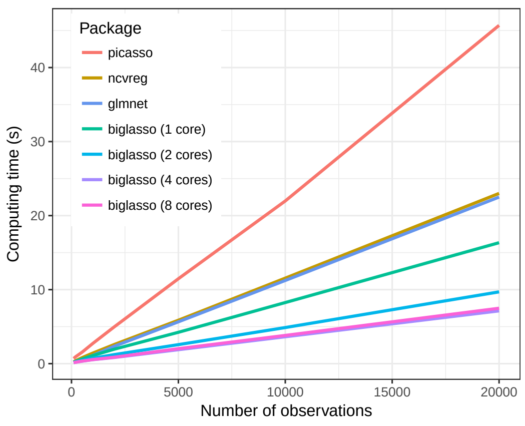
Real data

We also compare the computing time of **biglasso** with other packages for fitting lasso-penalized logistic regression based on four real data sets: (1) Subset of Gisette data set (Guyon et al., 2005); (2) P53 mutants data set (Danziger et al., 2009); (3) Subset of NEWS20 data set (Keerthi and DeCoste, 2005); (4) Subset of RCV1 text categorization data set (Lewis et al., 2004). The P53 data set can be found on the UCI Machine Learning Repository website[3] (Lichman, 2013). The other three data sets are obtained from the LIBSVM data repository site.[4]

[3]`https://archive.ics.uci.edu/ml/datasets/p53+Mutants`
[4]`https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html`



(a) Varying p, $n = 1,000$.



(b) Varying n, $p = 10,000$.

Figure 3: Mean computing time (in seconds) of solving the lasso-penalized logistic regression over a sequence 100 λ values as a function of p (Left) and n (Right).

Table 4 presents the dimensions of the data sets and the mean computing times. Again, **biglasso** outperforms all other packages in terms of computing time in all the real data cases. In particular, It's significantly faster than **picasso** with the speedup ranging from 2 to 5.5 times (for P53 data and RCV1 data, respectively). On the other hand, compared to **glmnet** or **ncvreg**, **biglasso** doesn't provide as much improvement in speed as in the linear regression case. The main reason is that safe rules for logistic regression do not work as well as safe rules for linear regression: they are more computationally expensive and less powerful in discarding inactive features.

Package	Gisette $n = 5,000$ $p = 5,000$	P53 $n = 16,592$ $p = 5,408$	NEWS20 $n = 2,500$ $p = 96,202$	RCV1 $n = 5,000$ $p = 47,236$
picasso	6.15 (0.03)	19.49 (0.06)	68.92 (8.17)	53.23 (0.13)
ncvreg	5.50 (0.03)	10.22 (0.02)	38.92 (0.56)	19.68 (0.07)
glmnet	3.10 (0.02)	10.39 (0.01)	25.00 (0.16)	14.51 (0.04)
biglasso	2.02 (0.01)	9.47 (0.02)	18.84 (0.22)	9.72 (0.04)

Table 4: Mean (SE) computing time (in seconds) for solving the lasso-penalized logistic regression along a sequence of 100 λ values on real data sets.

Validation

To validate the numerical accuracy of our implementation, we contrast the model fitting results from **biglasso** to those from **glmnet** based on the following relative difference criterion:

$$RD(\lambda) = \frac{\hat{Q}(\hat{\boldsymbol{\beta}}^B; \lambda) - \hat{Q}(\hat{\boldsymbol{\beta}}^G; \lambda)}{\hat{Q}(\hat{\boldsymbol{\beta}}^G; \lambda)}, \tag{1}$$

where $\hat{\boldsymbol{\beta}}^B$ and $\hat{\boldsymbol{\beta}}^G$ denote the **biglasso** and **glmnet** solutions, respectively. Four real data sets are considered, including MNIST and GWAS for linear regression, and P53 and NEWS20 for logistic regression. For the GWAS and P53 data sets, we obtain 100 RD values, one of each value of λ along the regularization path. For the MNIST and NEWS20 data sets, we obtained solutions for 20 different response vectors, each with a path of 100 λ values, resulting in 2,000 RD values.

Table 5 presents the summary statistics of $RD(\lambda)$ for the 4 real data sets. For both linear and logistic regression cases, all values of $RD(\lambda)$ values are extremely close to zero, demonstrating that **biglasso** and **glmnet** converge to solutions with virtually identical values of the objective function.

Statistic	Linear regression		Logistic regression	
	MNIST	GWAS	P53	NEWS20
Minimum	-7.7e-3	-3.9e-4	-6.4e-3	-1.6e-3
1^{st} Quantile	-1.6e-3	-2.7e-5	1.7e-5	-2.2e-4
Median	-9.5e-4	1.6e-4	2.0e-4	-1.1e-4
Mean	-1.1e-3	8.3e-4	2.2e-4	-1.2e-4
3^{rd} Quantile	-1.3e-4	1.3e-3	7.7e-4	1.0e-10
Maximum	4.2e-3	4.2e-3	2.0e-4	2.2e-3

Table 5: Summary statistics of $RD(\lambda)$ based on real data sets.

Data analysis example

In this section, we illustrate the usage of **biglasso** with a real data set `colon` included in **biglasso**. The `colon` data contains contains expression measurements of 2,000 genes for 62 samples from patients who underwent a biopsy for colon cancer. There are 40 samples from positive biopsies (tumor samples) and 22 from negative biopsies (normal samples). The goal is to identify genes that are predictive of colon cancer.

biglasso package has two main model-fitting R functions as below. Detailed syntax of the two functions can be found in the package reference manual.[5]

- **biglasso**: used for a single model fitting.
- `cv.biglasso`: used for performing cross-validation and selecting parameter λ.

We first load the data: `X` is the 62-by-2000 raw data matrix, and `y` is the response vector with 1 indicating tumor sample and 0 indicating normal sample.

```
R> library("biglasso")
R> data(colon)
R> X <- colon$X
R> y <- colon$y
```

Some information about `X` and `y` are as follows.

```
R> dim(X)
[1]   62 2000
R> X[1:5, 1:5]
  Hsa.3004 Hsa.13491 Hsa.13491.1 Hsa.37254 Hsa.541
t  8589.42   5468.24     4263.41   4064.94 1997.89
n  9164.25   6719.53     4883.45   3718.16 2015.22
t  3825.71   6970.36     5369.97   4705.65 1166.55
n  6246.45   7823.53     5955.84   3975.56 2002.61
t  3230.33   3694.45     3400.74   3463.59 2181.42
R> y
 [1] 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 1 1 1 1 1 1 1 1
[34] 1 1 1 1 1 0 1 1 0 0 1 1 1 1 0 1 0 0 1 1 0 0 1 1 1 1 0 1 0
```

Set up the design matrix

It's important to note that **biglasso** requires that the design matrix `X` must be a `big.matrix` object - an external pointer to the data. This can be done in two ways:

- If the size of `X` is small, as in this case, a `big.matrix` object can be created via:

  ```
  R> X.bm <- as.big.matrix(X)
  ```

  `X.bm` is a pointer to the data matrix, as shown in the following output.

  ```
  R> str(X.bm)
  Formal class 'big.matrix' [package "bigmemory"] with 1 slot
    ..@ address:<externalptr>
  ```

[5]https://cran.r-project.org/web/packages/biglasso/biglasso.pdf

```
R> dim(X.bm)
[1]   62 2000
R> X.bm[1:5, 1:5]
  Hsa.3004 Hsa.13491 Hsa.13491.1 Hsa.37254 Hsa.541
t  8589.42   5468.24     4263.41   4064.94 1997.89
n  9164.25   6719.53     4883.45   3718.16 2015.22
t  3825.71   6970.36     5369.97   4705.65 1166.55
n  6246.45   7823.53     5955.84   3975.56 2002.61
t  3230.33   3694.45     3400.74   3463.59 2181.42
```

- If the size of the data is large, the user must create a file-backed `big.matrix` object via the utility function `setupX` in **biglasso**. Specifically, `setupX` reads the massive data stored on disk, and creates memory-mapped files for that data set; this is demonstrated in the next section. A detailed example can also be found in the package vignettes.[6]

Single fit and cross-validation

After the setup, we can now fit a lasso-penalized logistic regression model.

```
R> fit <- biglasso(X.bm, y, family = "binomial")
```

The output object `fit` is a list of model fitting results, including the sparse matrix `beta`. Each column of `beta` corresponds to the estimated coefficient vector at one of the 100 values of λ.

In practice, cross-validation is typically conducted to select λ and hence the model with the best prediction accuracy. The following code snippet conducts a 10-fold (default) cross-validation using parallel computing with 4 cores.

```
R> cvfit <- cv.biglasso(X.bm, y, family = "binomial",
+                       seed = 1234, nfolds = 10, ncores = 4)
R> par(mfrow = c(2, 2), mar = c(3.5, 3.5, 3, 1) ,mgp = c(2.5, 0.5, 0))
R> plot(cvfit, type = "all")
```

Figure 4 displays the cross-validation curves with standard error bars. The vertical, dashed, red line indicates the λ value corresponding to the minimum cross-validation error.

Similar to **glmnet** and other packages, **biglasso** provides `coef`, `predict`, and `plot` methods for both **biglasso** and `cv.biglasso` objects. Furthermore, `cv.biglasso` objects contain the **biglasso** fit to the full data set, so one can extract the fitted coefficients, make predictions using it, etc., without ever calling **biglasso** directly. For example, the following code displays the full lasso solution path, with a red dashed line indicating the selected λ (Figure 5).

```
R> plot(cvfit$fit)
R> abline(v = log(cvfit$lambda.min), col = 2, lty = 2)
```

The coefficient estimates at the selected λ can be extracted via: `coef`:

```
R> coefs <- as.matrix(coef(cvfit))
```

Here we output only nonzero coefficients:

```
R> coefs[coefs != 0, ]
  (Intercept)      Hsa.8147     Hsa.36689     Hsa.42949     Hsa.22762     Hsa.692.2
 7.556421e-01 -6.722901e-05 -2.670110e-03 -3.722229e-04  1.698915e-05 -1.142052e-03
    Hsa.31801      Hsa.3016      Hsa.5392      Hsa.1832     Hsa.12241     Hsa.44244
 4.491547e-04  2.265276e-04  4.518250e-03 -1.993107e-04 -8.824701e-04 -1.565108e-03
     Hsa.2928     Hsa.41159     Hsa.33268      Hsa.6814      Hsa.1660
 9.760147e-04  7.131923e-04 -2.622034e-03  4.426423e-03  5.156006e-03
```

The `predict` method, in addition to providing predictions for a feature matrix `X`, has several options to extract different quantities from the fitted model, such as the number and identity of the nonzero coefficients:

```
R> as.vector(predict(cvfit, X = X.bm, type = "class"))
 [1] 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0
[43] 0 1 1 1 1 0 1 0 1 1 1 0 1 0 1 1 1 0 1 0
R> predict(cvfit, type = "nvars")
0.0522
```

[6]https://cran.r-project.org/web/packages/biglasso/vignettes/biglasso.pdf

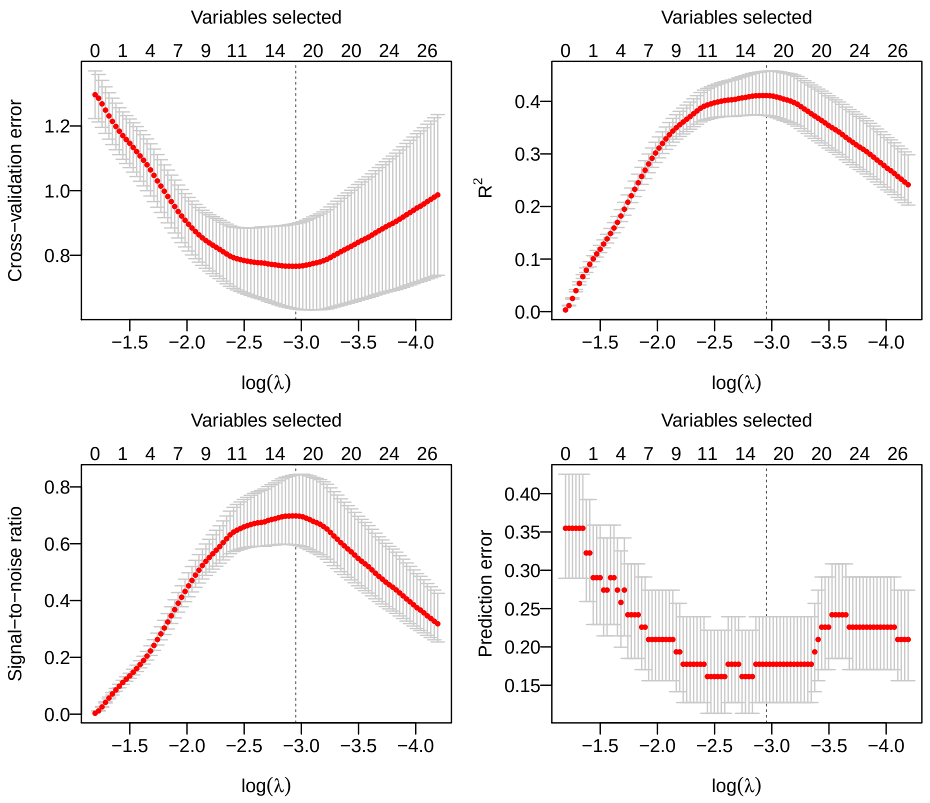


Figure 4: The cross-validation curves with standard error bars.

```
    16
R> predict(cvfit, type = "vars")
 Hsa.8147 Hsa.36689 Hsa.42949 Hsa.22762 Hsa.692.2 Hsa.31801  Hsa.3016  Hsa.5392
      249       377       617       639       765      1024      1325      1346
 Hsa.1832 Hsa.12241 Hsa.44244  Hsa.2928 Hsa.41159 Hsa.33268  Hsa.6814  Hsa.1660
     1423      1482      1504      1582      1641      1644      1772      1870
```

In addition, the summary method can be applied to a `cv.biglasso` object to extract useful cross-validation results:

```
R> summary(cvfit)
lasso-penalized logistic regression with n=62, p=2000
At minimum cross-validation error (lambda=0.0522):
-------------------------------------------------
  Nonzero coefficients: 16
  Cross-validation error (deviance): 0.77
  R-squared: 0.41
  Signal-to-noise ratio: 0.70
  Prediction error: 0.177
```

Application: Big Data case

Perhaps the most important feature of **biglasso** is its capability of out-of-core computing. To demonstrate this, we use it to analyze a simulated GWAS data set that consists of 3,000 observations and 1,340,000 features. Each feature cell is randomly assigned a value of 0 or 1 or 2. 200 features have nonzero coefficients, where 100 of which being 0.5 and the rest being -0.5. The size of the resulting raw feature matrix is over 36 GB data, which is more than 2x larger than the installed 16 GB RAM.

In this Big Data case, the data is stored in an external file on the disk. To use **biglasso**, memory-mapped files are first created via the following command.

```
R> library("biglasso")
R> X <- setupX(filename = "X_3000_1340000_200_gwas.txt")
```

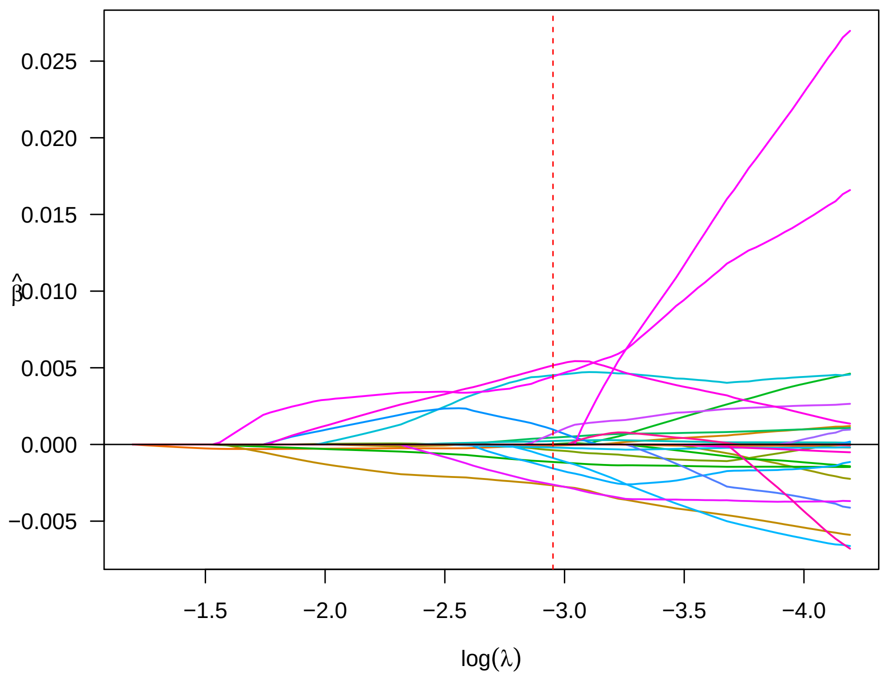


Figure 5: The solution path of lasso-penalized logistic regression model for the `colon` data.

This command creates two files in the current working directory:

- a memory-mapped file cache of the data, "X_3000_1340000_200_gwas.bin";
- a descriptor file, "X_3000_1340000_200_gwas.desc", that contains the backingfile description.

Note that this setup process takes a while if the data file is large. However, this only needs to be done once, during data processing. Once the cache and descriptor files are generated, all future analyses using **biglasso** can use the `X` object. In particular, should one close R and open a new R session at a later date, `X` can be seamlessly retrieved by attaching its descriptor file as if it were already loaded into the main memory:

```
R> X <- attach.big.matrix("X_3000_1340000_200_gwas.desc")
```

The object `X` returned from `setupX` or `attach.big.matrix` is a `big.matrix` object that is ready to be used for model fitting. Details about `big.matrix` and its related functions such as `attach.big.matrix` can be found in the reference manual of **bigmemory** package (Kane et al., 2013).

Note that the object `X` that we have created is a `big.matrix` object and is therefore stored on disk, not in RAM, but can be accessed as if it were a regular R object:

```
R> str(X)
Formal class 'big.matrix' [package "bigmemory"] with 1 slot
  ..@ address:<externalptr>
R> dim(X)
[1]    3000 1340000
R> X[1:10, 1:10]
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
 [1,]    1    1    0    1    1    2    2    2    1     0
 [2,]    0    1    2    1    0    0    1    2    0     2
 [3,]    2    2    2    1    1    0    0    1    0     0
 [4,]    1    2    1    1    1    0    2    2    0     1
 [5,]    0    0    0    0    2    2    0    1    0     2
 [6,]    2    0    0    0    1    2    1    0    0     0
 [7,]    1    0    1    2    1    1    2    0    2     2
 [8,]    2    2    0    2    2    0    0    0    0     2
 [9,]    0    2    2    2    0    0    2    0    2     2
[10,]    1    0    2    1    0    1    1    0    1     0
R> table(y)
y
   0    1
1487 1513
```

Here we fit both sparse linear and logistic regression models with the lasso penalty over the entire path of 100 λ values equally spaced on the scale of $\lambda/\lambda_{\max}$. The ratio of $\lambda_{\min}/\lambda_{\max}$ is set to be 0.05 for both models. Parallel computation with 4 cores is applied:

```
R> fit <- biglasso(X, y, ncores = 4)
R> fit <- biglasso(X, y, family = "binomial", ncores = 4)
```

The above code, which solves the full lasso path for a 36 GB feature matrix, required 147 minutes for the linear regression fit and 151 minutes for the logistic regression fit on an ordinary laptop with 16 GB RAM installed. Figure 6 depicts the lasso solution path for the sparse linear regression model. The following code extracts the nonzero coefficient estimates and the number of selected variables of the lasso model when $\lambda = 0.04$:

```
R> coefs <- as.matrix(coef(fit, lambda = 0.04))
R> coefs[coefs != 0, ]
  (Intercept)            V1            V4           V71           V76           V78
 4.917257e-01 -1.396769e-03 -1.198865e-02 -5.289779e-04 -1.475436e-03 -5.829812e-05
          V86           V97          V115          V127          V136          V152
-1.283901e-03 -3.437698e-03  1.672246e-04  1.012488e-03  5.913265e-03  9.485837e-03
         V157          V161          V176          V185       V118862       V160312
 1.992574e-04  1.654802e-03  1.731413e-03  2.411654e-04  4.871443e-03 -6.270115e-05
      V273843       V406640       V437742       V559219       V607177       V688790
-2.395813e-03 -5.189343e-03  6.079211e-03 -1.438325e-03  2.635234e-05 -3.645285e-04
      V814818       V849229       V916411       V981866      V1036672      V1036733
-3.611999e-04  9.293857e-03  2.637108e-03 -3.130641e-04  6.890073e-05  2.010702e-03
     V1110042      V1170636      V1279721
-8.323210e-04 -1.539764e-03 -3.729763e-05
R> predict(fit, lambda = 0.04, type = "nvars")
0.04
  32
```

Conclusion

We developed a memory- and computation-efficient R package **biglasso** to extend lasso model fitting to Big Data. The package provides functions for fitting regularized linear and logistic regression models with both lasso and elastic net penalties. Equipped with the memory-mapping technique and more efficient screening rules, **biglasso** is not only is 1.5x to 4x times faster than existing packages, but consumes far less memory and, critically, enables users to fit lasso models involving data sets that are too large to be loaded into memory.

Bibliography

D. Angelosante and G. B. Giannakis. Rls-weighted lasso for adaptive estimation of sparse signals. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 3245–3248, 2009. [p6]

D. P. Bovet and M. Cesati. *Understanding the Linux kernel*. O'Reilly, 2005. [p7]

P. Breheny. Marginal false discovery rates for penalized regression models. *arXiv preprint arXiv:1607.05636*, 2016. [p10]

P. Breheny and J. Huang. Coordinate descent algorithms for nonconvex penalized regression, with applications to biological feature selection. *Annals of Applied Statistics*, 5(1):232–253, 2011. [p6]

S. A. Danziger, R. Baronio, L. Ho, L. Hall, K. Salmon, G. W. Hatfield, P. Kaiser, and R. H. Lathrop. Predicting positive p53 cancer rescue regions using most informative positive (mip) active learning. *PLoS computational biology*, 5(9):e1000498, 2009. [p11]

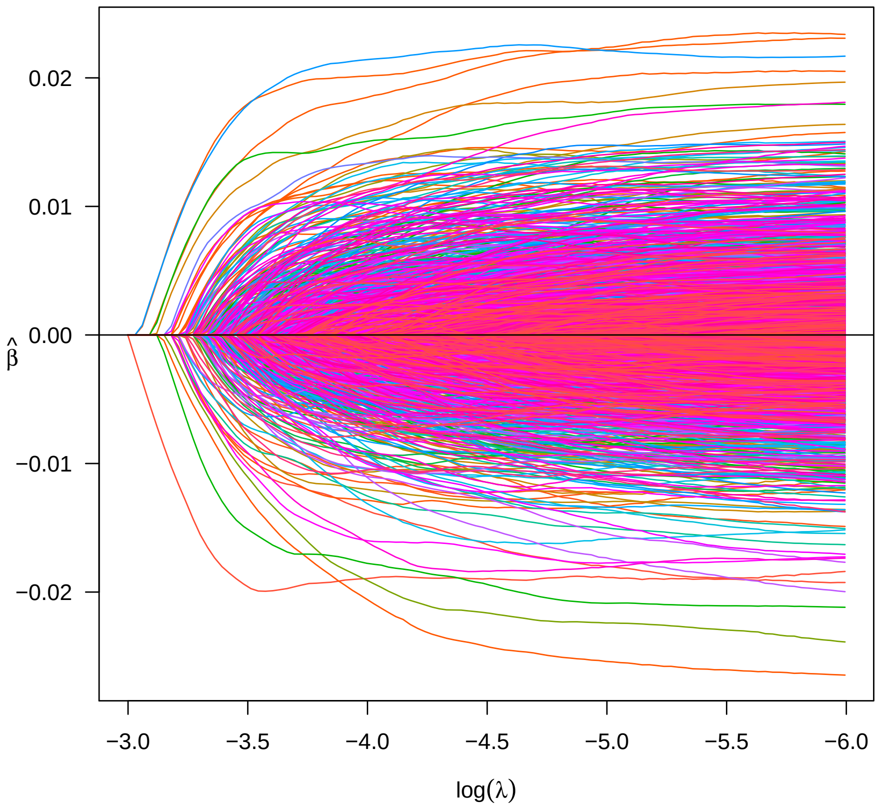


Figure 6: The solution path of the sparse linear regression model for the 36 GB GWAS data.

D. Dheeru and E. Karra Taniskidou. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`. [p10]

J. Fan, F. Han, and H. Liu. Challenges of big data analysis. *National Science Review*, 1(2):293–314, 2014. [p6]

J. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):1, 2010. [p6]

J. Ge, X. Li, M. Wang, T. Zhang, H. Liu, and T. Zhao. `picasso`*: Pathwise Calibrated Sparse Shooting Algorithm*, 2015. URL `https://CRAN.R-project.org/package=picasso`. R package version 0.5-4. [p6]

I. Guyon, S. Gunn, A. Ben-Hur, and G. Dror. Result analysis of the nips 2003 feature selection challenge. In *Advances in neural information processing systems*, pages 545–552, 2005. [p11]

X. Huang and W. Pan. Linear regression and two-class classification with gene expression data. *Bioinformatics*, 19(16):2072–2078, 2003. [p6]

M. J. Kane, J. Emerson, and S. Weston. Scalable strategies for computing with massive data. *Journal of Statistical Software*, 55(14):1–19, 2013. [p6, 7, 16]

S. S. Keerthi and D. DeCoste. A modified finite newton method for fast solution of large scale linear svms. *Journal of Machine Learning Research*, 6(Mar):341–361, 2005. [p11]

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. [p10]

D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. Rcv1: A new benchmark collection for text categorization research. *Journal of machine learning research*, 5(Apr):361–397, 2004. [p11]

Y. Li, A. Algarni, M. Albathan, Y. Shen, and M. A. Bijaksana. Relevance feature discovery for text mining. *IEEE Transactions on Knowledge and Data Engineering*, 27(6):1656–1669, 2015. [p6]

M. Lichman. UCI machine learning repository, 2013. URL `http://archive.ics.uci.edu/ml`. [p11]

Z. Lin, M. Kahng, K. M. Sabrin, D. H. P. Chau, H. Lee, and U. Kang. Mmap: Fast billion-scale graph computation on a pc via memory mapping. In *IEEE International Conference on Big Data*, pages 159–164, 2014. [p7]

S. T. Rao, E. Prasad, and N. Venkateswarlu. A critical performance study of memory mapping on multi-core processors: An experiment with k-means algorithm with large data mining data sets. *International Journal of Computers and Applications*, 1(9):90–98, 2010. [p7]

R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society Series B*, 58(1):267–288, 1996. ISSN 00359246. [p6]

R. Tibshirani, J. Bien, J. Friedman, T. Hastie, N. Simon, J. Taylor, and R. J. Tibshirani. Strong rules for discarding predictors in lasso-type problems. *Journal of the Royal Statistical Society Series B*, 74(2): 245–266, 2012. [p6, 7]

J. Wang, J. Zhou, J. Liu, P. Wonka, and J. Ye. A safe screening rule for sparse logistic regression. In *Advances in Neural Information Processing Systems*, pages 1053–1061, 2014. [p8]

J. Wang, P. Wonka, and J. Ye. Lasso screening rules via dual polytope projection. *Journal of Machine Learning Research*, 16:1063–1101, 2015. [p6, 7, 10]

J. Wright, A. Y. Yang, A. Ganesh, S. S. Sastry, and Y. Ma. Robust face recognition via sparse representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(2):210–227, 2009. [p6]

Y. Zeng and P. Breheny. `biglasso`*: Extending Lasso Model Fitting to Big Data*, 2016. URL `https://CRAN.R-project.org/package=biglasso`. `R` package version 1.3-1. [p6]

Y. Zeng, T. Yang, and P. Breheny. Hybrid safe–strong rules for efficient optimization in lasso-type problems. *Computational Statistics & Data Analysis*, 153:107063, 2021. ISSN 0167-9473. [p7]

Yaohui Zeng
Department of Biostatistics
University of Iowa
N301 CPHB 145 North Riverside Drive
Iowa City, IA 52242, United States of America
Email: `yaohui.zeng@gmail.com`

Patrick Breheny
Department of Biostatistics
University of Iowa
N336 CPHB 145 North Riverside Drive
Iowa City, IA 52242, United States of America
E-mail: `patrick-breheny@uiowa.edu`
URL: `http://myweb.uiowa.edu/pbreheny/index.html`

Six Years of Shiny in Research - Collaborative Development of Web Tools in R

by Peter Kasprzak, Lachlan Mitchell, Olena Kravchuk and Andy Timmins

Abstract The use of **Shiny** in research publications is investigated over the six and a half years since the appearance of this popular web application framework for R, which has been utilised in many varied research areas. While it is demonstrated that the complexity of **Shiny** applications is limited by the background architecture, and real security concerns exist for novice app developers, the collaborative benefits are worth attention from the wider research community. **Shiny** simplifies the use of complex methodologies for people of different specialities, at the level of proficiency appropriate for the end user. This enables a diverse community of users to interact efficiently, and utilise cutting edge methodologies. The literature reviewed demonstrates that complex methodologies can be put into practice without insisting on investment in professional training, for a comprehensive understanding from all participants. It appears that **Shiny** opens up concurrent benefits in communication between those who analyse data and other disciplines, that would enrich much of the peer-reviewed research.

Introduction

Data is the backbone of research. With the rise of automated data gathering tools, data size and complexity of analysis have driven a growing gap between research disciplines and the required data analysis. Another issue is the fact that different approaches to the same data can compromise validity, as seen in an analysis on effect sizes in observational studies, which found that varied methodological workflows could reverse conclusions regarding the studied intervention (Donoho, 2017). Collaborative learning which employs common task frameworks can help interpret, quantify, and possibly cap methodological variation across disciplines (Donoho, 2017). Software such as Matlab Moler and Mathworks (2012), Minitab Arend (2010), Genstat Payne et al. (2007) and SPSS Landau and Everitt (2004) have attempted to bridge this gap by creating more user friendly interfaces that either make coding more intuitive and easier to learn, or use drop down menus and radio button selection to bypass the command line. Current analytical software, such as those mentioned above, each have their own limitations which include non publication ready quality graphics, non-intuitive drop down menus, restrictive interfacing with other software, price point (including the cost of licensing the proprietary software) and the difficulties that inevitably occur when colleagues attempt to run code originating from other software on their preferred platform. Despite its own weaknesses, which include a very steep learning curve and non-intuitive programming language, R (R Core Team, 2019) has grown to become the most popular programming language for statistics and biological data analysis, spawning over 14,000 free to use packages over a wide range of subject material (Li et al., 2018).

While code of any language can be shared easily between users, general use requires a level of familiarity with the specific program. Transforming a piece of R code into a interactive app capable of use by a broad audience recently required either knowledge of other coding languages (Gunuganti, 2018), or consultation/collaboration with a computer scientist/app developer and the added time and cost justified. Specialist apps that benefit only a small number of people often do not meet cost/benefit benchmarks, which means that many useful advancements have stalled due to experience requirements with data analysis software, or an understanding of the underlying theory for day to day use, constituting a complexity barrier (DePalma, 2013). **Shiny** can generalise R code for all levels of users, bringing the latest advancements in methodology measurably closer to everyone. This does create new issues relating to data security, as novice app developers will now require a knowledge of web internet protocols for secure data transfers to be assured.

The increased use of technology, sensors and other data capture devices have brought an interesting issue to light. Researchers and practitioners without a background in data analysis now have the ability to gather large amounts of data (LaZerte et al., 2017). Limited options exist for those without data analysis training to correctly analyse data gathered from the field and experiments, which has arguably led to issues impacting experimental reproducibility. A Nature survey of 1,576 researchers from the disciplines of chemistry, physics and engineering, earth and environment, biology, medicine and other, found more than 50% surveyed believed that low statistical power or poor analysis was a strong contributor to irreproducibility (Baker, 2016). In the same survey more than 90% of respondents believed that a better understanding of statistics was required to drive reproducibility of research. Learning analytical methodologies and programs is a non-trivial task, and subcontracted analysis, even within house, generally comes with a wait for results. Purchasing proprietary software can be inflexible and often expensive which takes resources away from research, and open source software is dependent on a minimum level of computer literacy, and the ability to test the software to ensure correct results is essential (LaZerte et al., 2017).

Open source and free, **Shiny** has grown in popularity with the first **Shiny** Developer Conference held in January 2016 and a growing use in peer reviewed academic papers. While the number of papers has steadily increased each year, **Shiny** remains an incompletely explored topic, with the potential for **Shiny** to make a significant positive contribution to the general field of science not yet properly examined. To the best of our knowledge this is the first **Shiny** review.

The rest of this review is organised as follows. Section 2 details the literature search, the keywords and findings. Section 3 presents the technical aspects of **Shiny**, including hosting costs and security, along with restrictions. Section 4 discusses the use of **Shiny** in research with relevant examples from the literature, and finally section 5 gives the conclusion along with the authors opinion.

Algorithms/methods for literature search

A thorough search for **Shiny** results in the academic literature was undertaken to investigate the growth from 2012 - 2018 in research, and which publications and subject areas were represented. The focus of this paper is the use of **Shiny** to bridge specialist academic and theoretical innovations, and its role in disseminating knowledge to government, industry and the general community, therefore, it is acknowledged that this is a non-exhaustive list of **Shiny** case uses. We acknowledge that the literature search is not fully comprehensive, as newspaper articles, blogs and other non-academic areas were filtered, which makes this review biased towards an academic standpoint. The search was conducted with four major data bases. Web of Science and Scopus were used due to their reputation as multi-disciplinary databases, with Google Scholar and the University of Adelaide (UofA) utilised as their algorithms search the entire document and all fields for keywords. The keywords used in all searches were of the form "Shiny Web Application" OR "Shiny Web App", with an exact search not suitable in this case, and "R" not included to avoid the inevitable non-related hits. The search was then filtered by year to span 2012 - 2018 and the document type was limited to Dissertations, Articles, Conference proceedings and Reviews (where allowed), to investigate the use of **Shiny** in the research literature only. Books were excluded from the search due to the small number of published materials. A separate search conducted for books showed that as of 2016, only two books were written on the use of **Shiny**, with both being structured as instructional manuals. Beeley (2013) takes the beginner from their first application and walks them through the major concepts to more complicated applications, while Moon (2016) uses **Shiny** to teach **ggplot2** (Wickham, 2016) graphics. As of 2018 a Google search for "Shiny Web Application Books" yielded seven results, including one second edition release.

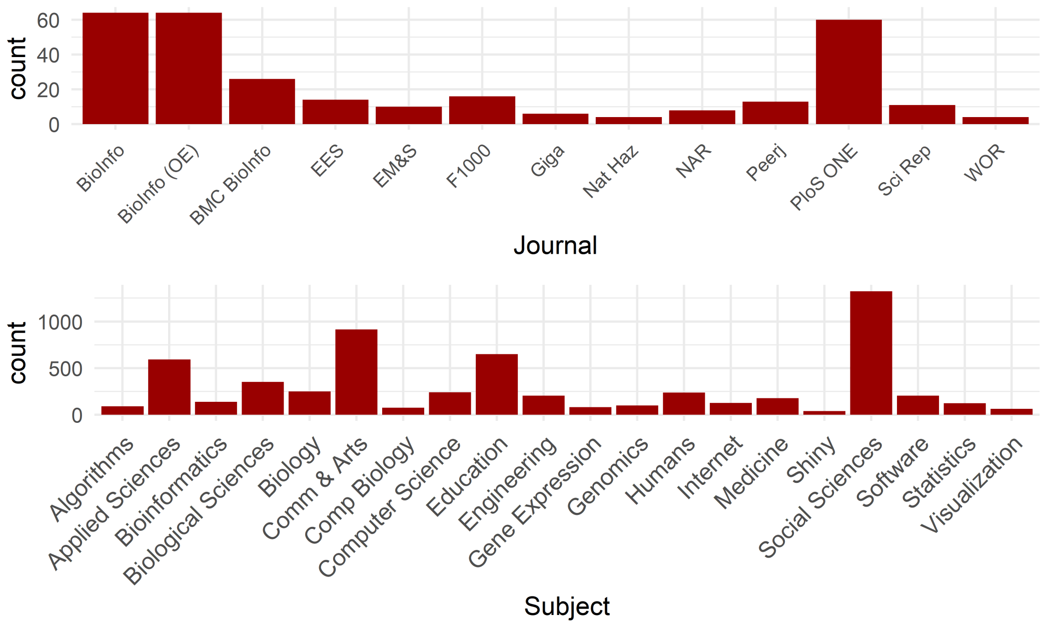
The UofA search engine is powered by ExLibris Primo, which includes all resources owned or subscribed to by the library and selected free and open access resources. It includes 345 databases, and links to the major collections of articles and eBooks totalling over 50 million items, which can expand out to 100s of millions. The UofA search was conducted with the terms "Shiny web app OR Shiny web application" and returned 5,251 results with 1,391 peer reviewed articles, 3,456 dissertations, 18 reviews and 119 conference proceedings, which are broken into results by journal title, subject tag and languages published in, displayed in Figure 1.

The search in Scopus used TITLE-ABS-KEY(Shiny AND web AND app*) AND PUBYEAR $>$ 2012 AND PUBYEAR $<$ 2019 as its search terms, with the same document limitations. The decision was made to only check the title, abstract and keywords as too many irrelevant results were being returned when including other fields, with a final result of 155 items. These were restricted once again to articles (114), conference papers (38), conference reviews (2), and reviews (1). These are broken into number of records published by year, journal title and subject tag displayed in Figure 2. There were 154 records published in English, with one record published in Spanish.

The Web of Science search returned 144 results using the search criteria ALL = (Shiny Web App*) and filtered to the same time frame. Choices of document criteria included Articles and Proceedings papers which resulted in 110 Articles and 34 Proceedings papers, with dissertations not returned in this search. These are once again broken into publications per year, journal title and subject tag displayed in Figure 3.

Again the predominate language was English with 142 records, one Spanish, and one Portuguese record found.

The Google Scholar search terms used first were [Shiny web app | application] which returned 16,400 results. A range of additional terms were used to narrow down results including, "security OR complexity OR architecture OR hosting", with "Shiny" being a required keyword, which returned approximately 10,400 results. Unfortunately by record 135 irrelevant results were found that did not contain the required term "Shiny", which appeared to be an error in the algorithm. Given that the search returned over 100% more results than the UofA search, it was decided to not use these results to create this paper, as the UofA search utilised the Google Scholar databases.



Language	English	French	German	Chinese	Portuguese
Count	5232	5	8	5	4
Language	Spanish	Chamorro	Slavic	Italian	Norwegian
Count	4	1	1	1	1

Figure 1: Summary of UofA search results partitioned into number of results by journal for records > 3 with abbreviations given in Table 1, results by subject tag with abbreviations given in Table 2, and published languages.

Shiny is a relatively novel tool with the total number of papers found quite small in comparison to larger bodies of work. Assuming the UofA library search completely covers the other 3 databases (which it is advertised to do), there is an approximate total of 5,000 unique peer reviewed pieces of work utilising **Shiny** since 2012, an average of over 700 papers per year. All searches showed that Bioinformatics journals published the largest number of **Shiny** papers, however, the vast majority of papers were published by a diverse range of titles, in a diverse range of fields. This indicates that **Shiny** is a flexible tool and not area specific. Only the Scopus search returned slightly different information with Computer Science, Biochemistry, Mathematics, and Other subjects tags registering the largest number of relevant hits. On closer inspection, while Bioinformatics did not register as a subject heading, the journal that published the greatest number of papers was Bioinformatics, followed by BMC Bioinformatics, which suggests that there is simply a difference in subject labelling. Far more papers were found by keyword searches in the body of the document, as evidenced by the total numbers of papers found by the library search from the UofA. This suggests that **Shiny** has been utilised as a general tool, and not as a new discovery in the later years. The vast majority of all papers were written in English, with some European countries represented, but very few Chinese papers.

The results of the search algorithms are reasonably reproducible, with some fluctuation occurring depending on the sources of publications, and performance of the search engine. In our experience the fluctuation is less than 10%. Google scholar

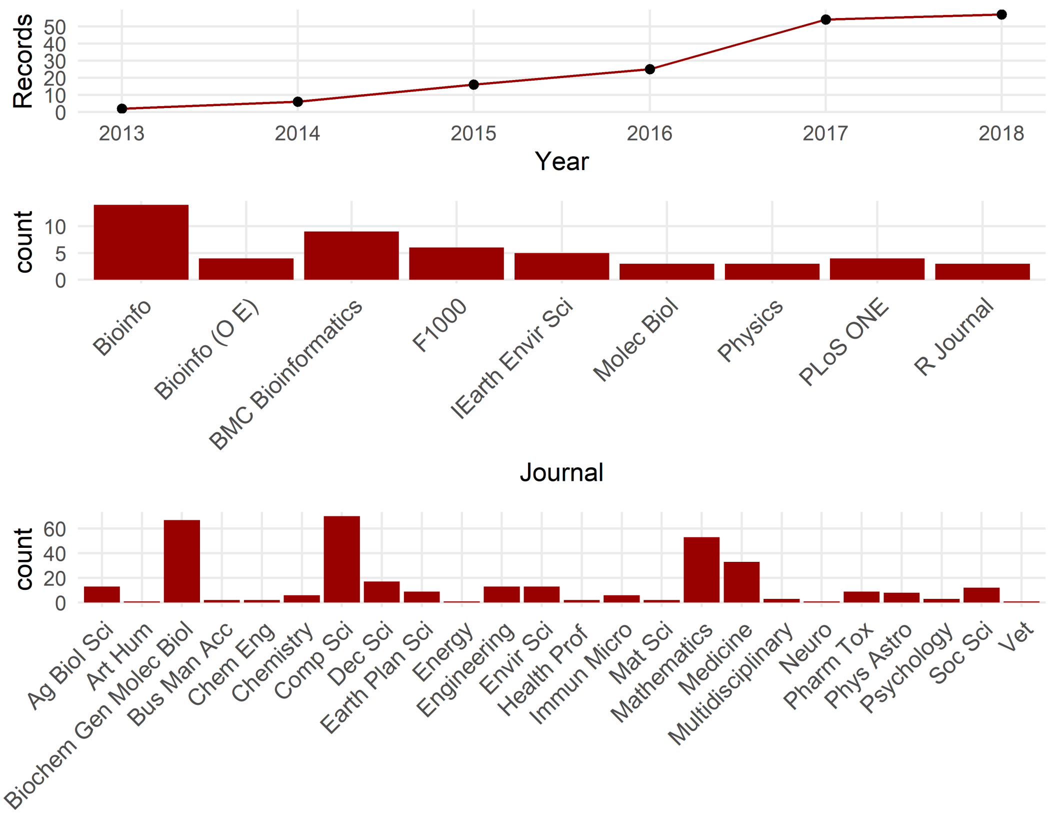


Figure 2: Summary of Scopus search results partitioned into number of published results by year, journal title for records > 2 with abbreviations given in Table 1 and results by subject tag with abbreviations given in Table 2.

significantly alters the number of found papers depending on sorting. If sorting by relevance is checked then 127,000 results are found. Sorting by date reduces this number to what is stated above.

A subset of 600 papers was chosen for thorough reading to inform this report. These were the top 600 results returned by the UofA records search when sorted via relevance. The relevance ranking employed by ExLibris Primo is comprised of four main criteria.

1. Degree of match: Fields such as title, author and subject field are given a higher ranking, along with order of the query terms and completeness of phrases.
2. Academic significance: Citations and journal impact factor.
3. Type of search: Primo infers if the search is broad-topic or specific-topic, with broad topic searches amplifying overview material such as reference articles.
4. Publication date: Newer material is given preference.

The papers that discussed **Shiny** generally had "Shiny" in the title and/or the subject fields, increasing their relevance score. Earlier papers were more likely to discuss **Shiny**, with newer papers more likely to mention **Shiny** in the text only. The relevance search yielded a high number of the older papers as high relevance, along with a very broad range of use cases. The limit of 600 papers was an empirical cut off

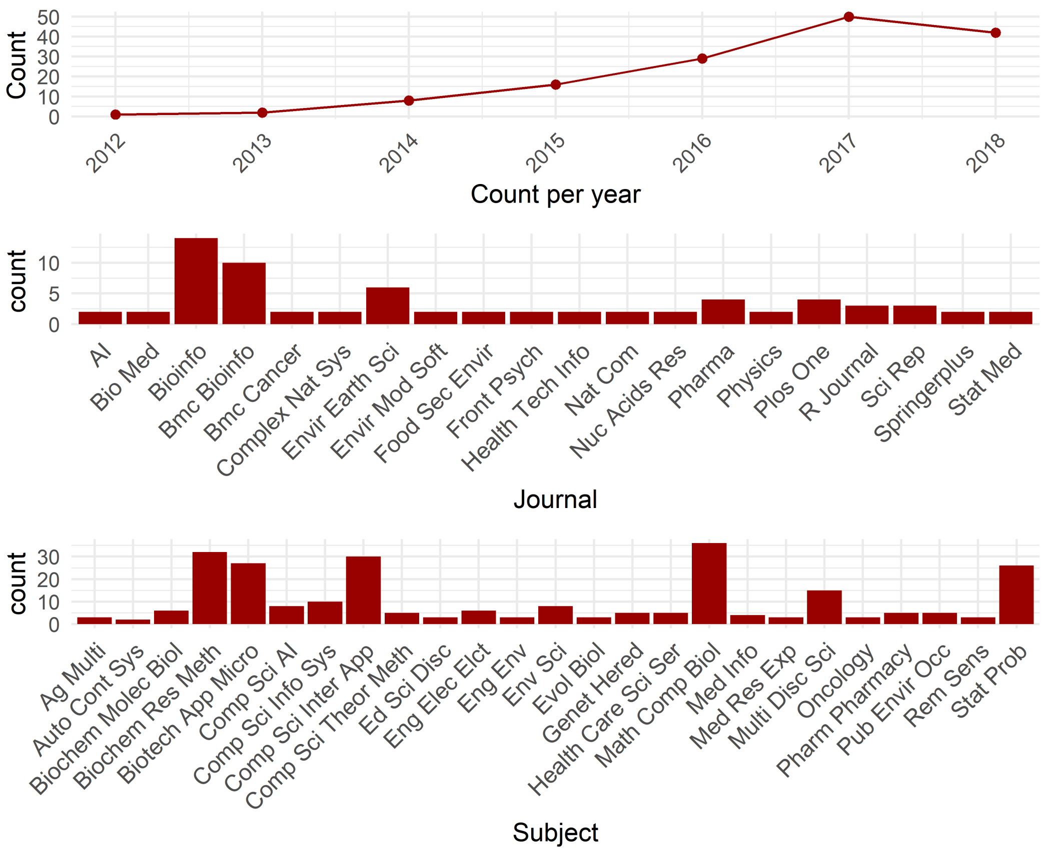


Figure 3: Summary of WOS search results partitioned into number of published results by year, journal title for records >2 with abbreviations given in Table 1 and results by subject tag >2 with abbreviations given in Table 2.

point, as this was the stage that papers had ceased discussing **Shiny**, and were only stating its use. It was decided that enough use cases had been examined to make comments regarding **Shiny**'s relatively widespread use in the academic work. 445 original applications were introduced in these papers, which utilised 373 unique R packages. 229 unique peer reviewed journals were represented with 55 published in Bioinformatics, 31 published in PLoS ONE, and 21 published in BMC Bioinformatics. The final subset of papers that most thoroughly discussed the implementation of **Shiny** were chosen to create this report, and are given as references.

Technical aspects

Architectural overview

A Web application framework for R, **Shiny** was conceptualised by RStudio's CTO Joe Cheng and announced at the Joint Statistical Meeting conference in July of 2012 as a tool designed to help R programmers create interactive web applications, reports and analysis without the need to know HTML, CSS, or JavaScript (Chang et al., 2018).

The power of **Shiny** comes from the ability for an R user to quickly and simply code a reactive framework. A reactive framework allows objects to be updated when a source is changed, along with all connected objects. For example, in an imperative programming paradigm such as the R language, setting the line

$$c = a + b$$

means that c is assigned the sum of previously defined terms a, b and will not change when the values of a, b are changed without the variable c being re-evaluated. Reactive programming allows the value of c to be updated almost instantaneously, including all other variables and outputs dependent on c, whenever a or b is changed. R completes this task with information travelling from input to output in a pull fashion. A pull fashion is when c learns of the new value of a or b when c is called. **Shiny** creates a system of alerts which flag changed expressions and the server re-evaluates all flags in an event known as a *flush* (Grolemund, 2015). Using two object classes called reactive values, such as a = reactive(), and observers, such as b = plot(), **Shiny** creates a *reactive context* between the two objects known as a *call-back* which is a command to re-evaluate the observer. Multiple observers can be linked to the same *reactive value* and the server will queue up all *call-backs* and run each *call-back* in the event of a *flush* (Grolemund, 2015).

This reactive framework allows user inputs to be evaluated via a UI (user interface) with a series of easily coded widgets such as text boxes, radio buttons and drop down menus from pre-programmed R code. **Shiny** then seamlessly updates outputs of tables, plots and summaries. A non R user can change the values of a and b via the user interface and explore the pre-coded results dependent on c.

A **Shiny** application has two main parts. A user interface object and a server function. The user interface contains code for the layout and appearance of the app, with default choices restricted in appearance. Layouts can be customised and changes to the appearance can be made if the programmer has some knowledge of HTML or CSS. For standard applications simple commands suffice and a knowledge of HTML or CSS language is not required for tweaks. The server function houses all the code that drives functionality of the application and can utilise all the built in programs available to R and RStudio users.

Hosting

For a small number of applications and limited run hours the cost of hosting a **Shiny** application is free, but it can become expensive quickly. Hosting on shinyapps.io requires no system administration knowledge and comes with layers of security and is supported by **Shiny**'s IT team. According to the RStudio pricing website (Core Team, 2012), the platform is free for 5 applications and 25 active hours which increases to $39 AUD a month for unlimited applications and 500 active hours, to the top tier of $299 AUD a month, which allows for unlimited applications and 10,000 active hours. **Shiny** also has the option of Shiny server, Shiny Server Pro or RStudio Connect. These require a level of system administration knowledge, and also requires the apps to be hosted on a physical or virtual machine. RStudio Server Pro costs $9,995 AUD per year (Core Team, 2012). RStudio connect allows installation of software on a server behind your existing firewall and costs between $14,995 AUD per year ($62 AUD per user/month) to $75,995 AUD per year ($6.25 AUD per user/month) for a larger, specified number of named users (Core Team, 2012). Shiny server prices were not available. For those with an in-depth knowledge of internet security, it is possible, and more economical, to host the application independently by their own means.

Security

As **Shiny** is primarily a web technology, a very strong focus on application security must be adhered to, with novices in computer science more likely to make critical mistakes (Charpentier, 2013). A well-known concept in cryptography and web security is *unknown unknowns* (Charpentier, 2013). Put simply, this refers to the fact that a developer cannot build defences for attack vectors they are unfamiliar with. For this reason, it is generally wise to leave the specifics of data security to experts in the field, with the end-developer instead relying on the vetted work that has been done for them.

For users of **Shiny** who elect to use shinyapps.io by RStudio, this is essentially what happens. Once uploaded, the application is secured behind best practices (Core Team, 2012). Unfortunately, this service is prohibitively expensive when compared to hosting the server on a cloud platform like Amazon Web Services (AWS) (Amazon, 2019) or Microsoft Azure (Microsoft, 2019), which requires application security to be taken into the app creator's hands. Certificates need to be created and kept up-to-date, servers need to be configured for HTTPS amongst other security protocols (Charpentier, 2013). Due to the local nature of R, this is likely to be a new issue, requiring a new set of skills, for many data analysts operating on the platform.

Architectural issues

Curiously there are only a small number of papers that explicitly mentioned concerns and limitations with respect to the use of **Shiny** to develop research focused apps. A paper by Dwivedi and Kowalski (2018) was the first to include a limitations section, emphasising the requirement for a fast internet connection when dealing with large data sets. This could be mitigated with the use of cloud based resources to store the data and host the app, with potentially faster network and processing speeds available with respect to local connections. Guo (2018) found R package updates a legitimate concern, as updates can occur without warning and crash an application. A less serious issue, the lack of flexibility of the dashboard is born from the simplification of creation, with **Shiny**'s dashboard not being as flexible as one created in Java (Ge et al., 2018). There are however challenges with the use of **Shiny**, with one of them being the background architecture.

While **Shiny** has many benefits, the architecture of **Shiny** will be a limiting factor when building complex applications. Previously this concerned dismissed with Joe Cheng stating more recently:

> In the past, we've responded rather glibly to these requests: "Just use functions!" (Cheng, 2019)

As of 2017 **Shiny** has made moves to address this issue with the creation of modulisation (Cheng, 2017), however, the more involved use cases would be handled better by other computing languages, for the reasons detailed below. An analogous way of conceptualising this would be in the difference between applets and applications. Applets are generally small, discrete, and of low complexity, and are developed to perform a small number of functions for a highly specific purpose. Most **Shiny** products would fit this description quite well, while applications on the other hand, are generally more complex (Fayram, 2011). They are built for a number of different use cases, and tend to have relatively large codebases. Well established and popular web application frameworks such as Angular and React exist to fit these situations, containing much more general functionality than **Shiny** with much less specific functionality (such as functions related to data visualization) (Mitchell, 2018). None of this is to say that complex applications can not be created with **Shiny**, just that it may not be the most mature solution for the task.

While **Shiny** will undoubtedly continue to evolve in much the same way as R has, and many issues today will be gone tomorrow, a number of well-established software development paradigms must be diverged from:

- **Shiny** actively encourages the use of single-file applications, generally referring to this singular file as `app.R` (Chang et al., 2018). Defining everything in a singular file works well for prototypes, but quickly falls apart as an application grows and increases in complexity. In general, code is compartmentalised into files which contain the logic for a single component. By allowing a single file to grow monolithic in size, code readability and re-usability is challenged, consequently making it harder to add additional components in the future (Fayram, 2011).
- **Shiny** insists on a reactive data-driven model over the more traditional and common event-driven model. While not necessarily a flaw in and of itself, many novice developers consider reactivity in programming to be a non-trivial concept (Fayram, 2011). Considering that **Shiny**, by nature, is aimed towards data analysts rather than computer scientists, it can increase the initial difficulty hurdle that beginners have to overcome. To further this issue, a bug has existed in RStudio since at least February 2018 that prevents automatic reloading from working with sourced files. When using multiple files like this, the server needs to be manually stopped and restarted between every change, making for a tedious development cycle. Concerns on the subject have not been addressed by either the RStudio or **Shiny** core developers (Hansen, 2018).

Both of the above points begin to cause major issues when put together. Encouragement of singular source files results in code quickly becoming unruly, threatening flexibility. This heightened complexity of source code will invariably be replicated within the reactive dependency graph, **Shiny**'s internal mapping of reactive nodes and their relationships. In the event that something is not working as expected, RStudio provides little to no internal tools for debugging this graph. A new addition to CRAN in the form of **Reactlog** (Schloerke and Cheng, 2019) is a first attempt to address this issue, which usually forces the developer to painstakingly debug the graph by hand. As the application becomes increasingly complex, this process gets closer and closer to impossible. Many of these cases are not as yet documented due to **Shiny** being a burgeoning technology, and to the best of our knowledge, this is the most in-depth look at the challenges in the peer reviewed literature.

A package, **ShinyTester** (Kohli, 2017), was added to CRAN `https://cran.r-project.org/` early in 2017. While it provides a promising first approach to debugging tools for **Shiny** (such as the inclusion of a dependency graph visualiser), it unfortunately seems to have been abandoned. Tools like this would likely alleviate the above outlined concerns.

Data size

Shiny is designed foremost as a server technology, with applications intended to be used remotely with a stable internet connection (R Core Team, 2017). **Shiny**

applications must be built with upload and download requirements in the fore. While **Shiny** applications can be run locally, doing so requires a base level of knowledge of R that may make it a sub-optimal approach, and limits accessibility. This is comparable to how mobile applications are generally shipped as pre-compiled binaries, rather than as raw source that the user would need to compile and install manually. One of the largest issues with this inherent reliance on connectivity is the need for data to be uploaded and downloaded. Since **Shiny** has no inbuilt data streaming functionality, it is not possible to work with parts of data while waiting for the rest to upload (R Core Team, 2017). An entire transfer must be completed before the dataset is made available to the application. This forces the application to require pre-partitioned uploads, which may not be possible for all types of datasets.

It is quite common to see a dataset approaching gigabytes in size, especially prevalent in areas such as genomic sequencing, which is generally technically unrealistic for datasets of this size to be worked with remotely, and would include extra data costs. If a large amount of bandwidth was made available to a single user, this could open up your service to potential denial-of-service attacks by malicious entities (Cloudflare, 2019). Furthermore, it may be legally unrealistic in terms of data ownership. Users are often uncomfortable providing sensitive data to unknown receivers, as there is no way for a **Shiny** app to prove that its not storing uploaded information permanently for the developer's own academic or financial gain (Kacha and Zitouni, 2018).

Literature analysis

Complexity barrier

The pattern of peer reviewed work, as shown in Figure 1, Figure 2, and Figure 3 shows that Bioinformatics is a popular and growing area for **Shiny** apps. Areas that traditionally have a lower focus on data analysis skills for researchers, such as Biological Sciences, Education and Index Medicus, appear to have higher usage levels. In the current literature **Shiny** is primarily used as a delivery/visualisation tool, and is not the focus with many papers referencing the use of **Shiny** but not discussing the merits. This trend becomes obvious in more recent papers, with much of the best discussion occurring in earlier papers.

To investigate the uptake of **Shiny** we must first understand some of the factors that determine the uptake of innovation. These are stated by Rogers (2001) as: (a) relative advantage, (b) compatibility, (c) complexity, (d) trialability and (e) observability. Rogers (2001) defined complexity as

> *...the degree to which an innovation is perceived as difficult to understand and use.*

Analysis methods are a non-trivial skill and the complexity of new methodologies in data analysis are a major hurdle for their uptake in fields such as biology and agriculture (DePalma et al., 2017). To drive innovation and uptake, tools must be accessible and usable by all interested parties (Jahanshiri and Shariff, 2014; Klein et al., 2017). Moraga (2017) noted in the area of public health, that while there had been progress in methodology and analysis

> *...these methods are still inaccessible for many researchers lacking the adequate programming skills to effectively use the required software.*

The first peer reviewed **Shiny** publications appeared in 2013, with the first two dissertations contributing the most to this discussion, as they give a glimpse to the vast potential for **Shiny**. The first dissertation using **Shiny** was published by DePalma (2013) which allowed non-computer literate clinicians the ability to harness powerful statistical methodologies in a robust framework, to conduct antimicrobial susceptibility tests which determine an unknown pathogens susceptibility to various antibiotics. DePalma (2013) noted that previously new methods have not been adopted due to

> *...various computational difficulties and an absence of easy to use software for clinicians.*

Complex methodologies were able to be immediately used by end users without an assumption of computational skills, to inform important medical checks. This direct transfer of method is a concrete example of how **Shiny** is able to make complex research available to all interested parties, regardless of knowledge level. Specialised applications such as this would be difficult and costly to create without **Shiny**, and without general use software the advanced methodology would stall in uptake due to complexity barriers. This was later followed up with dBETS (diffusion Breakpoint Estimation Testing Software) by DePalma et al. (2017) who once again acknowledged

> *...the computational complexities associated with these new approaches has been a significant barrier for clinicians.*

Shiny is a potential solution to the barrier of complexity for the uptake of new methodologies.

Cross collaboration and dialogue

Cross collaboration between researchers and the easy dissemination of results is key to external validity (Munafò et al., 2017). **Shiny** promotes collaboration by allowing people with varying skill levels access to more complex methodologies. This has a flow on benefit to promote the collaboration of practitioners with researchers, or field researches with theorists, in order to create specialised, fit for purpose applications. This is illustrated in a paper by Wages and Petroni (2018) which designs and conducts Phase 1 dose finding trials using the continual reassessment method, and was noted to

> *...facilitate more efficient collaborations within study teams.*

Klein et al. (2017) underscores the requirement that

> *...facilitating the deployment of web applications for data analysis is important to promote collaboration within the scientific community and between scientists and stakeholders.*

Further examples of **Shiny** being used to open discussion by using apps to bring relevant parties with differing skills sets into collaboration include Díaz-Gay et al. (2018) who stated

> *...analysis of somatic mutational signatures remains currently inaccessible for a substantial proportion of the scientific community.*

As well as Whateley et al. (2015) who noted the knowledge gap between relevant parties and

> *...demonstrates the use of the* **Shiny** *web framework to bridge that gap, allowing for collaborative development of web tools that can be coded in the widely-used and free R statistical computing language.*

The ability to bridge the gap between researchers and the tools required for their data analysis was mentioned by Chen et al. (2018) in the context of environmental DNA. eDNA is becoming an essential tool in ecology and conservation biology and is utilised by a range of people with varying skill levels with Kandlikar et al. (2018) stating

> *Results from eDNA analyses can engage and educate natural resource managers, students, community scientists and naturalists, but without significant training in bioinformatics, it can be difficult for this diverse audience to interact with eDNA results.*

Shiny allows discipline specialists outside of computer science to code their own apps, bridging the skill gap for other researchers (Niu, 2017). This was demonstrated by an app called *Armadillo Mapper* (Feng et al., 2017), which was designed specifically to decrease the time between synthesising distributional knowledge on a computer and carrying out conservation efforts in the field. This encourages those without the resources to conduct their own analysis, to closely collaborate with analysts to create specialist applications. Rather than sending final data to an analyst for analysis, discussion and collaboration is encouraged at the beginning of an experiment. This enables low quality data due to issues such as pseudo-replication, low power and confounding variables to be avoided at the design stage rather than the analysis stage.

Flexibility to link other software

Shiny has the flexibility to bridge the gap between specialised data gathering tools and available software. A **Shiny** app accompanying the R package **rHyperSpec** (Laney, 2013) was created to take complex data generated by hyperspectral cameras and link the data to available software packages in response to the problem of

> *...few free, open-source software packages that enable researchers to easily process and analyse such data in a manner that maximizes inter-comparison between studies.*

This showcases the flexibility of **Shiny** applications being able to upload information in various formats, make appropriate changes, and output the data in a form usable by another, completely independent piece of equipment/software. Previously there were precious few options to link independent software/equipment, especially without breaching warranty restrictions. **Shiny** shows tremendous flexibility in working with existing infrastructure to help decrease costs, especially when technologies are in their infancy.

Shiny gains flexibility and customisability directly from R. One of **Shiny**'s most useful abilities is to wrap existing, or new, R packages for general consumption. Beck (2014) created an app called *Seed* which bundled several R packages together and used **Shiny** to host them on the web allowing

...user's access to powerful R based functions and libraries through a simple user interface.

In the Precision Agriculture (PA) space, farmers have access to a multitude of proprietary sensors, few of which can be linked directly to analysis tools (Jayaraman et al., 2016). **Shiny**'s highly customisable framework facilitates the linking of several pieces of independent software, and can avoid manual data wrangling and transfer. As an example, Jahanshiri and Shariff (2014) took data from existing PA sensors and utilised R functions for its analysis and visualisation of results. **Shiny** has proved more than useful in the results visualising area, with packages such as **ShinyStan** (Gabry et al., 2018) created in order to visualise modelling parameters and results from MCMC simulations.

Generalising complex methodologies

R packages can be thought of as a level of abstraction down from the mathematical theory, as the packages can be used by those without a need to have full understanding of the methodology. **Shiny** can be thought of as another level of abstraction down again, as R packages can be utilised, without needing an understanding of R itself. This ability to generalise analysis methodologies makes **Shiny** available to any interested party, and is the mechanism that drives flexibility, dialogue and cross collaboration.

There is an overarching requirement when making tools available to a broader audience to ensure correct methodology. The first example of using **Shiny** to guide and educate the user came from Assaad et al. (2014) who created two **Shiny** apps intended to allow Microsoft Word users access to One Way Anova analysis and post hoc tests. The app gave instructions to guide users through the process, which greatly simplified the common statistical test, whilst promoting proper statistical methodology. A real world example of protecting the end user comes from Hsu et al. (2018), who created an app for proper randomisation when allocating participants to a three-armed, double-blinded, randomized controlled trial (RCT) for depression. One critical characteristic of the app was to ensure mistakes were not made when properly balancing strata. A fail safe against experimental error was employed by not allowing participants to have their experimental ID overwritten, which means that any accidental changes after treatment has begun would not impact on the treatment received.

Generalised applications must be flexible to differing individual parameters. **Shiny** makes it a trivial task to allow parameters of a methodology to be changed depending on individual circumstances. **Shiny** wrapped simulations were used to explore humanitarian response and financial institution resiliency for earthquake risk in Indonesia, with Hartell (2014) allowing the simulation to be tweaked by individuals so that adjustments to calibration parameters could be made based on specific interests or circumstances. Other apps that allowed the user to specify parameters were created by Zhou et al. (2014) for detecting differential expression in RNA sequencing and Yin (2014) who utilised Bayesian statistical modelling to investigate the networks of epidemics transmission.

Shiny makes complex methodologies accessible to those who would previously not be part of the conversation, most likely due to a lack of theoretical study, or lack of familiarity with coding or analysis programs. **Shiny** was explicitly noted to help increase engagement by LaZerte et al. (2017), who created FeedR in order to record and visualise RFID data from ecological studies. The huge amount of data from

RFID quickly becomes overwhelming and requires specialist methods to cope. The *feedR* **Shiny** app was created to wrap the paired R package in order that

> *...this framework will become a meeting point for science, education and community awareness...we aim to inspire citizen engagement while simultaneously enabling robust scientific analysis (LaZerte et al., 2017).*

Responsible and open research

Reproducibility of research is a critical cornerstone of responsible research practices. Studies, such as Munafò et al. (2017), have indicated that reproducibility is not at high enough levels, with results of a survey conducted by Baker (2016) and published in Nature found

> *...more than 70% of researchers have tried and failed to reproduce another scientist's experiments and more than half have failed to reproduce their own experiments.*

Eight practices are argued for by Munafò et al. (2017), which includes promoting transparency and open science to increase reproducibility. Open source software such as **Shiny** can aid these objectives by creating a vessel to preserve code, and allow a greater number of interested parties to critically evaluate methodologies and results.

One benefit of **Shiny** wrapped code is that methodology comparisons become much easier to conduct. Methodologies wrapped in **Shiny** applications can be compared on a known data set under various conditions by the end user. This is a powerful tool in the advancement of reproducible research. **Shiny** was explicitly used in a dissertation by Parvandeh (2018) as a vessel to show the strategy and to create reproducibility of results enhancing responsible and reproducible research goals.

A **Shiny** app, or at least the code behind it is enduring. A paper from Sieriebriennikov et al. (2014) included a **Shiny** application named Nematode Indicator Joint Analysis (NINJA) 2.0, to automate manual calculations previously carried out using spreadsheet software, which is time consuming and prone to errors. The aim for *NINJA* to remain freely accessible was validated when it was later used by Burkhardt et al. (2019) to aid nematode calculations in semi-arid wheat systems, 5 years after its release. This suggests that maintaining a **Shiny** application is not overly difficult. The benefit of **Shiny**'s easy maintenance and updating was mentioned for the first time in a dissertation by Niu (2017), which highlighted the fact that only the source code requires changing without having to download patches or modify individual applications.

Shiny also appeared in conjunction with machine learning to explore early phase drug discovery processes (Korkmaz et al., 2015), with Wojciechowski et al. (2015) noticing the power of **Shiny** to disseminate the results of research, stating

> *Interactive applications, developed using* **Shiny** *for the R programming language, have the potential to revolutionize the sharing and communication of pharmacometric model simulations.*

Free and open source software is ideally suited to disseminating the products of research (LaZerte et al., 2017), which drives collaboration and was noted to encourage local and direct monitoring of environmental data in Kenya (Mose et al., 2017). LaZerte et al. (2017) also found that **Shiny**'s open source nature has another important benefit which

> *...reduces financial barriers to its use and the open-source aspect permits and encourages collaboration which can result in better, more powerful software.*

Cross collaboration and use of open source **Shiny** will hopefully also help drive data sharing. Yi et al. (2017) noted the utility and importance of data sharing promoted by **Shiny** applications, which is also one of the key recommendations by Munafò et al. (2017) in order to drive transparency and openness, and is currently a policy by *Science* and *Springer Nature* journals.

An educational tool

The strengths shown by **Shiny** seems to fit very well in the educational sector and it was no surprise that **Shiny** has been used as a teaching aid in order to get complex ideas across to students (Williams and Williams, 2018). Educational tools such as those by Arnholt (2018) help teach the concept of power in hypothesis tests, with Williams and Williams (2018) creating a similar application for confidence intervals, and an app by Courtney and Chang (2018) which normalises large datasets and allows students to explore the results of differing transformations. There are other benefits to using **Shiny** in the education sector. Kandlikar et al. (2018) created the **Shiny** app *ranacapa* and found that

> *A key benefit of using ranacapa was that despite having no prior bioinformatics experience, students could begin exploring the biodiversity in their samples in a matter of minutes by using the online instance of the* **Shiny** *app.*

This had the flow on effect of allowing teachers to focus more on the theory instead of the inevitable problems when teaching new, more complex software and provided a useful aid to self-learning (Kandlikar et al., 2018).

Conclusion

This review examined **Shiny** in peer reviewed publications from 2012 to 2018 and mapped the growth through various research fields. A subset of 600 papers were used to inform the bulk of the paper, with the authors personal experiences of **Shiny** included. While **Shiny** is not a *silver bullet* solution to issues in the research field, it confers the ability for specialised applications to be created cheaply and easily, such that any level of end user maybe included, no matter the complexity level of the methodology. This primary benefit creates a direct pathway for new findings to be rapidly incorporated into established work flows. The flexibility of **Shiny** means that apps can be tailored to exact specifications in all regards, with changes and maintenance of the app made relatively easy as an ongoing product of consultation further promoting collaboration. If an app is considered worthwhile adopting to an existing work flow, widespread adoption across an entire workplace is as simple as sharing the web address. This will have the inevitable knock on effects of allowing fewer people to do more, which will necessarily mean existing jobs have the potential of becoming obsolete. The argument that other jobs will be created is true, but not necessarily within the same sector, or for the people whose job has become obsolete. As we progress further into this technological world, this argument will require mature debate and nuance to be resolved.

In the current literature, **Shiny** has been used primarily as a visualisation and dissemination tool, with many papers not exploring the concurrent benefits and challenges mentioned in this review. One benefit identified in the literature is the opportunity to increase high value dialogue between people with different skill sets. For example, field researchers, primary producers or marketers are able to sit down with theoretical researchers/consultants to create highly customised applications for up-coming experiments or daily work. Code published as a **Shiny** application has the useful attribute of making methodology comparisons easy, which promotes reproducible research and best practice standards.

With the ability to accelerate access to data analysis techniques comes the paramount issue of data security for those not familiar with web protocols. It is essential for those who host web based applications to become knowledgeable in this area. Web security protocols are likely to be a new skill set for many R programmers, and a non-trivial task potentially constituting a bottle neck for widespread **Shiny** uptake.

While the use of **Shiny** apps require minimal experience with computers, the creation of a **Shiny** application is a different story. The lack of debugging tools, the encouragement of single file applications, and the current implementation of the reactive data-driven model will limit the complexity of future applications.

Other open source and proprietary options are currently available, however, **Shiny**'s flexibility, customisability, and low cost is highly desirable. Open source software comes with a minimum knowledge requirement barrier to entry, and proprietary software can be expensive and inflexible to changing situations and circumstances. Maintenance is required with **Shiny**, although it is limited to updating code when R packages or dependencies change, and can be done via the source code for all users.

Shiny is one of the better tools available if one is an existing R programmer given its inherited scope from R. It helps promotes open and reproducible research, and offers a real pathway to making complicated methodologies usable to those outside of research. The ability to provide an avenue to increase high value collaboration and dialogue between interested parties with differing skills sets make **Shiny** a tool worth exploring.

Acknowledgements

We would like to gratefully acknowledge the scholarship for the M.Phil program of the first author from the Grains Research and Development Corporation (GRDC) Australia.

Bibliography

Amazon. Amazon EC2 Pricing, 2019. URL https://aws.amazon.com/ec2/pricing/. [p27]

D. Arend. Minitab 17 Statistical Software, 2010. [p20]

A. T. Arnholt. Using a Shiny app to teach the concept of power. *Teaching Statistics*, 2018. URL https://doi.org/10.1111/test.12186. [p34]

H. I. Assaad, L. Zhou, R. J. Carroll, and G. Wu. Rapid publication-ready MS-Word tables for one-way ANOVA. *SpringerPlus*, **3**(1):474, 2014. URL https://doi.org/10.1186/2193-1801-3-474. [p32]

M. Baker. 1,500 scientists lift the lid on reproducibility. *Nature News*, **533**(7604):452, 2016. URL https://doi.org/10.1038/533452a. [p21, 33]

D. Beck. *Investigating the Use of Classification Models to Study Microbial Community Associations with Bacterial Vaginosis*. PhD thesis, University of Idaho, 2014. [p31]

C. Beeley. *Web Application Development with R Using Shiny*. Olton: Packt Publishing Ltd, first edition, 2013. [p22]

A. Burkhardt, S. S. Briar, J. M. Martin, P. M. Carr, J. Lachowiec, C. Zabinski, D. W. Roberts, P. Miller, and J. Sherman. Perennial crop legacy effects on nematode community structure in semi-arid wheat systems. *Applied Soil Ecology*, 2019. URL https://doi.org/10.1016/j.apsoil.2018.12.020. [p33]

W. Chang, J. Cheng, J. J. Allaire, Y. Xie, and J. McPherson. **Shiny**: Web Application Framework for R, 2018. R package version 1.2.0. [p25, 28]

J. Charpentier. Web application Security. Technical Report Network Project, 7.5 hp, Halmstad University, 2013. [p27]

Z. Chen, Y. Zheng, Z. Wang, M. Kutner, W. J. Curran, and J. Kowalski. Interactive calculator for operating characteristics of phase I cancer clinical trials using standard 3+3 designs. *Contemporary Clinical Trials Communications*, **12**:145–153, 2018. URL https://doi.org/10.1016/j.conctc.2018.10.006. [p31]

J. Cheng. Shiny - Modularizing Shiny app code, 2017. URL https://shiny.rstudio.com/articles/modules.html. [p27]

J. Cheng. Shiny - Modularizing Shiny app code, 2019. URL http://shiny.rstudio-staging.com/articles/modules.html. [p27]

Cloudflare. What Is a Distributed Denial-of-Service (DDoS) Attack?, 2019. URL https://www.cloudflare.com/en-au/learning/ddos/what-is-a-ddos-attack/. [p29]

R. Core Team. RStudio Pricing, 2012. URL https://www.rstudio.com/pricing/. [p26, 27]

M. G. R. Courtney and K. C. Chang. Dealing with non-normality: An introduction and step-by-step guide using R. *Teaching Statistics*, **40**(2):51–59, 2018. URL https://doi.org/10.1111/test.12154. [p34]

G. DePalma. *Disk Diffusion Breakpoint Determination Using a Bayesian Nonparametric Variation of the Errors-in-Variables Model*. PhD thesis, Purdue University, 2013. [p21, 30]

G. DePalma, J. Turnidge, and B. A. Craig. Determination of disk diffusion susceptibility testing interpretive criteria using model-based analysis: Development and implementation. *Diagnostic Microbiology and Infectious Disease*, **87**(2):143–149, 2017. URL https://doi.org/10.1016/j.diagmicrobio.2016.03.004. [p29, 30]

M. Díaz-Gay, M. Vila-Casadesús, S. Franch-Expósito, E. Hernández-Illán, J. J. Lozano, and S. Castellví-Bel. Mutational Signatures in Cancer (**MuSiCa**): A web application to implement mutational signatures analysis in cancer samples. *BMC Bioinformatics*, **19**(1):224, 2018. URL https://doi.org/10.1186/s12859-018-2234-y. [p30]

D. Donoho. 50 Years of Data Science. *Journal of Computational and Graphical Statistics*, 26(4):745–766, 2017. URL https://doi.org/10.1080/10618600.2017.1384734. [p20]

B. Dwivedi and J. Kowalski. **shinyGISPA**: A web application for characterizing phenotype by gene sets using multiple omics data combinations. *PLoS ONE*, **13** (2), 2018. URL https://doi.org/10.1371/journal.pone.0192563. [p27]

D. Fayram. Functional Programming Is Hard, That's Why It's Good, 2011. URL https://www.pixelstech.net/article/1318920938-Functional-Programming-Is-Hard-That-s-Why-It-s-Good. [p27, 28]

X. Feng, M. C. Castro, E. Linde, and M. Papeş. **Armadillo Mapper**: A Case Study of an Online Application to Update Estimates of Species' Potential Distributions. *Tropical Conservation Science*, **10**, 2017. URL https://doi.org/10.1177/1940082917724133. [p31]

J. Gabry, S. D. Team, M. Andreae, M. Betancourt, B. Carpenter, Y. Gao, A. Gelman, B. Goodrich, D. Lee, D. Song, and R. Trangucci. Shinystan: Interactive Visual and Numerical Diagnostics and Posterior Analysis for Bayesian Models, 2018. [p32]

S. X. Ge, E. W. Son, and R. Yao. **iDEP**: An integrated web application for differential expression and pathway analysis of RNA-Seq data. *BMC Bioinformatics*, **19**, 2018. URL https://doi.org/10.1186/s12859-018-2486-6. [p27]

G. Grolemund. Shiny - How to understand reactivity in R, 2015. URL https://shiny.rstudio.com/articles/understanding-reactivity.html. [p26]

A. Gunuganti. Application Development Framework for R/Shiny. In *PharmaSUG 2018 Conference Proceedings*, volume AD-24, page 9. PharmaSUG, 2018. [p21]

J. Guo. *Developing a Visualization Tool for Unsupervised Machine Learning Techniques on *Omics Data*. PhD thesis, University of Washington, 2018. [p27]

K. Hansen. Rstudio - Reload Shiny App when using source'ed modules without restart, 2018. URL https://stackoverflow.com/questions/50169896/reload-shiny-app-when-using-sourceed-modules-without-restart. [p28]

J. Hartell. *Earthquake Risk in Indonesia: Parametric Contingent Claims for Humanitarian Response and Financial Institution Resiliency*. PhD thesis, University of Kentucky, 2014. [p32]

K. J. Hsu, K. Caffey, D. Pisner, J. Shumake, S. Risom, K. L. Ray, J. A. J. Smits, D. M. Schnyer, and C. G. Beevers. Attentional bias modification treatment for depression: Study protocol for a randomized controlled trial. *Contemporary Clinical Trials*, **75**: 59–66, 2018. URL https://doi.org/10.1016/j.cct.2018.10.014. [p32]

E. Jahanshiri and A. R. M. Shariff. Developing web-based data analysis tools for precision farming using R and Shiny. *IOP Conference Series: Earth and Environmental Science*, **20**(1), 2014. URL https://doi.org/10.1088/1755-1315/20/1/012014. [p29, 32]

P. P. Jayaraman, A. Yavari, D. Georgakopoulos, A. Morshed, and A. Zaslavsky. Internet of Things Platform for Smart Farming: Experiences and Lessons Learnt. *Sensors*, **16**(11):1884, 2016. URL https://doi.org/10.3390/s16111884. [p32]

L. Kacha and A. Zitouni. An Overview on Data Security in Cloud Computing. pages 250–261, 2018. URL https://doi.org/10.1007/978-3-319-67618-0_23. [p29]

G. S. Kandlikar, Z. J. Gold, M. C. Cowen, R. S. Meyer, A. C. Freise, N. J. Kraft, J. Moberg-Parker, J. Sprague, D. J. Kushner, and E. E. Curd. **Ranacapa**: An R package and Shiny web app to explore environmental DNA data with exploratory statistics and interactive visualizations. *F1000Research*, **7**, 2018. URL https://doi.org/10.12688/f1000research.16680.1. [p31, 34]

T. Klein, A. Samourkasidis, I. N. Athanasiadis, G. Bellocchi, and P. Calanca. **webXTREME**: R-based web tool for calculating agroclimatic indices of extreme events. *Computers and Electronics in Agriculture*, **136**:111–116, 2017. URL https://doi.org/10.1016/j.compag.2017.03.002. [p29, 30]

A. Kohli. Shinytester: Functions to minimize bonehead moves while working with 'shiny', 2017. URL https://CRAN.R-project.org/package=ShinyTester. R package version 0.1.0. [p28]

S. Korkmaz, G. Zararsiz, and D. Goksuluk. **MLViS**: A Web Tool for Machine Learning-Based Virtual Screening in Early-Phase of Drug Discovery and Development. *PLoS One; San Francisco*, **10**(4), 2015. URL https://doi.org/http://dx.doi.org.proxy.library.adelaide.edu.au/10.1371/journal.pone.0124600. [p33]

S. Landau and B. Everitt. *A Handbook of Statistical Analyses Using SPSS*. Chapman & Hall/CRC, 2004. [p20]

C. M. Laney. *Toward New Data and Information Management Solutions for Data-Intensive Ecological Research*. PhD thesis, The University of Texas at El Paso, 2013. [p31]

S. E. LaZerte, M. W. Reudink, K. A. Otter, J. Kusack, J. M. Bailey, A. Woolverton, M. Paetkau, A. de Jong, and D. J. Hill. Feedr and animalnexus.ca: A paired R package and user-friendly Web application for transforming and visualizing animal movement data from static stations. *Ecology and Evolution*, **7**(19):7884–7896, 2017. URL https://doi.org/10.1002/ece3.3240. [p21, 32, 33, 34]

J. Li, B. Cui, Y. Dai, L. Bai, and J. Huang. BioInstaller: A comprehensive R package to construct interactive and reproducible biological data analysis applications based on the R platform. *PeerJ*, **6**, 2018. URL https://doi.org/10.7717/peerj.5853. [p20]

Microsoft. Pricing – Linux Virtual Machines | Microsoft Azure, 2019. URL https://azure.microsoft.com/en-au/pricing/details/virtual-machines/linux/. [p27]

E. Mitchell. Shiny applications without Shiny, 2018. URL http://washstat.org/presentations/20181024/Mitchell.pdf. [p28]

C. Moler and Mathworks. MATLAB 8.0 and Statistics Toolbox 8.1, 2012. [p20]

K.-M. Moon. *Learn ggplot2 Using Shiny App*. Number 2197-5736 in Use R! Springer, 2016. [p22]

P. Moraga. SpatialEpiApp: A Shiny web application for the analysis of spatial and spatio-temporal disease data. *Spatial and Spatio-temporal Epidemiology*, **23**:47–57, 2017. URL https://doi.org/10.1016/j.sste.2017.08.001. [p29]

V. N. Mose, D. Western, and P. Tyrrell. Application of open source tools for biodiversity conservation and natural resource management in East Africa. *Ecological Informatics*, 2017. URL https://doi.org/10.1016/j.ecoinf.2017.09.006. [p34]

M. R. Munafò, B. A. Nosek, D. V. M. Bishop, K. S. Button, C. D. Chambers, N. Percie du Sert, U. Simonsohn, E.-J. Wagenmakers, J. J. Ware, and J. P. A. Ioannidis. A manifesto for reproducible science. *Nature Human Behaviour*, **1**(1), 2017. URL https://doi.org/10.1038/s41562-016-0021. [p30, 33, 34]

B. Niu. *Mass Spectrometry-Based Structural Proteomics: Methodology and Application of Fast Photochemical Oxidation of Proteins (FPOP)*. PhD thesis, Washington University in St. Louis, 2017. [p31, 33]

S. Parvandeh. *Epistasis Network and Machine Learning Methods for the Analysis of Biological Large Data*. PhD thesis, The University of Tulsa, 2018. [p33]

R. Payne, D. Murray, S. Harding, D. Baird, and D. Soutar. GenStat, 2007. [p20]

R Core Team. Shiny Welcome to Shiny, 2017. URL https://shiny.rstudio.com/tutorial/written-tutorial/lesson1/. [p28, 29]

R Core Team. ***R**: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2019. [p20]

M. Rogers. Evolution: Diffusion of Innovations. In N. J. Smelser and P. B. Baltes, editors, *International Encyclopedia of the Social & Behavioral Sciences*, pages 4982–4986. Pergamon, 2001. URL https://doi.org/10.1016/B0-08-043076-7/03094-1. [p29]

B. Schloerke and J. Cheng. **Reactlog**: Reactivity Visualizer for 'shiny', 2019. [p28]

B. Sieriebriennikov, H. Ferris, and R. G. M. de Goede. NINJA: An automated calculation system for nematode-based biological monitoring. *European Journal of Soil Biology*, **61**:90–93, 2014. URL https://doi.org/10.1016/j.ejsobi.2014.02.004. [p33]

N. A. Wages and G. R. Petroni. A web tool for designing and conducting phase I trials using the continual reassessment method. *BMC Cancer*, **18**, 2018. URL https://doi.org/10.1186/s12885-018-4038-x. [p30]

S. Whateley, J. D. Walker, and C. Brown. A web-based screening model for climate risk to water supply systems in the northeastern United States. *Environmental Modelling & Software*, **73**:64–75, 2015. URL https://doi.org/10.1016/j.envsoft.2015.08.001. [p31]

H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. ISBN 978-3-319-24277-4. URL https://ggplot2.tidyverse.org. [p22]

I. J. Williams and K. K. Williams. Using an R shiny to enhance the learning experience of confidence intervals. *Teaching Statistics*, **40**(1):24–28, 2018. URL https://doi.org/10.1111/test.12145. [p34]

J. Wojciechowski, A. M. Hopkins, and R. N. Upton. Interactive Pharmacometric Applications Using R and the Shiny Package. *CPT: Pharmacometrics & Systems Pharmacology*, **4**(3):146–159, 2015. URL `https://doi.org/10.1002/psp4.21`. [p33]

L. Yi, H. Pimentel, and L. Pachter. Zika infection of neural progenitor cells perturbs transcription in neurodevelopmental pathways. *PLoS One; San Francisco*, **12** (4), 2017. URL `https://doi.org/http://dx.doi.org.proxy.library.adelaide.edu.au/10.1371/journal.pone.0175744`. [p34]

J. Yin. *Bayesian Statistical Modeling in Epidemics and the Contact Networks That Transmit Them*. PhD thesis, The University of Iowa, 2014. [p32]

X. Zhou, H. Lindsay, and M. D. Robinson. Robustly detecting differential expression in RNA sequencing data using observation weights. *Nucleic Acids Research*, **42**(11), 2014. URL `https://doi.org/10.1093/nar/gku310`. [p32]

Peter Kasprzak
University of Adelaide
School of Agriculture Food and Wine, PMB 1, Glen Osmond, SA 5064
Australia
`peter.kasprzak@adelaide.edu.au`

Lachlan Mitchell
University of Adelaide
School of Agriculture Food and Wine, PMB 1, Glen Osmond, SA 5064
Australia
`lachlan.mitchell@icloud.com`

Olena Kravchuk
University of Adelaide
School of Agriculture Food and Wine, PMB 1, Glen Osmond, SA 5064
Australia
`olena.kravchuk@adelaide.edu.au`

Andy Timmins
University of Adelaide
School of Agriculture Food and Wine, PMB 1, Glen Osmond, SA 5064
Australia
`andy.timmins@adelaide.edu.au`

Appendix

Table 1: Table of journal abbreviations

Journal	Abbreviation
2nd Symposium On Lapan Ipb Satellite Lisat For Food Security And Environmental Monitoring	Food Sec Envir
Bioinformatics	Bioinfo
Bioinformatics (Oxford England)	Bioinfo (OE)
Bmc Bioinformatics	BMC Bioinfo
Bmc Cancer	BMC Cancer
Environmental Earth Sciences	Envir Earth Sci
Environmental Modelling And Software	Envir Mod Soft
F1000research	F1000
Frontiers In Psychology	Front Psych
Gigascience	Giga
Iop Conference Series Earth And Environmental Science	Earth Envir Sci
Journal Of Pharmacokinetics And Pharmacodynamics	Pharma
Journal Of Physics Conference Series	Physics
Lecture Notes In Artificial Intelligence	AI
Natural Hazards	Nat Haz
Nature Communications	Nat Com
Nucleic Acids Research	Nuc Acids Res
Peerj	Peerj
PloS ONE	PloS ONE
Procedia Environmental Sciences	Envir Sci
R Journal	R Journal
Scientific Reports	Sci Rep
Source Code For Biology And Medicine	Bio Med
Springerplus	Springerplus
Statistics In Medicine	Stat Med
Studies In Health Technology And Informatics	Health Tech Info
Wellcome Open Research	Well Open Res
Workshop And International Seminar On Science Of Complex Natural Systems	Complex Nat Sys

Table 2: Table of abbreviations for subject tag

Subject	Abbreviation
Agricultural Biological Sciences	Ag Biol Sci
Agriculture Multidisciplinary	Ag Multi
Arts Humanities	Art Hum
Automation Control Systems	Auto Cont Sys
Biochemical Research Methods	Biochem Res Meth
Biochemistry Genetics Molecular Biology	Biochem Gen Molec Biol
Biochemistry Molecular Biology	Biochem Molec Biol
Biotechnology Applied Microbiology	Biotech App Micro
Business Management Accounting	Bus Man Acc
Chemical Engineering	Chem Eng
Chemistry	Chemistry
Communication and the Arts	Comm & Arts
Computational Biology	Comp Biology
Computer Science	Comp Sci
Computer Science Artificial Intelligence	Comp Sci AI
Computer Science Information Systems	Comp Sci Info Sys
Computer Science Interdisciplinary Applications	Comp Sci Inter App
Computer Science Theory Methods	Comp Sci Theor Meth
Decision Sciences	Dec Sci
Earth Planetary Sciences	Earth Plan Sci
Education Scientific Disciplines	Ed Sci Disc
Energy	Energy
Engineering	Engineering
Engineering Electrical Electronic	Eng Elec Elct
Engineering Environmental	Eng Env
Environmental Science	Envir Sci
Environmental Sciences	Env Sci
Evolutionary Biology	Evol Biol
Genetics Heredity	Genet Hered
Health Care Sciences Services	Health Care Sci Ser
Health Professions	Health Prof
Immunology Microbiology	Immun Micro
Materials Science	Mat Sci
Mathematical Computational Biology	Math Comp Biol
Mathematics	Mathematics
Medical Informatics	Med Info
Medicine	Medicine
Medicine Research Experimental	Med Res Exp
Multidisciplinary	Multidisciplinary
Multidisciplinary Sciences	Multi Disc Sci
Neuroscience	Neuro
Oncology	Oncology
Pharmacology Pharmacy	Pharm Pharmacy
Pharmacology Toxicology Pharmaceutics	Pharm Tox
Physics Astronomy	Phys Astro
Psychology	Psychology
Public Environmental Occupational Health	Pub Envir Occ
Remote Sensing	Rem Sens
Social Sciences	Soc Sci
Statistics Probability	Stat Prob
Veterinary	Vet

A Fast and Scalable Implementation Method for Competing Risks Data with the R Package fastcmprsk

by Eric S. Kawaguchi, Jenny I. Shen, Gang Li, and Marc A. Suchard

Abstract Advancements in medical informatics tools and high-throughput biological experimentation make large-scale biomedical data routinely accessible to researchers. Competing risks data are typical in biomedical studies where individuals are at risk to more than one cause (type of event) which can preclude the others from happening. The Fine and Gray (1999) proportional subdistribution hazards model is a popular and well-appreciated model for competing risks data and is currently implemented in a number of statistical software packages. However, current implementations are not computationally scalable for large-scale competing risks data. We have developed an R package, **fastcmprsk**, that uses a novel forward-backward scan algorithm to significantly reduce the computational complexity for parameter estimation by exploiting the structure of the subject-specific risk sets. Numerical studies compare the speed and scalability of our implementation to current methods for unpenalized and penalized Fine-Gray regression and show impressive gains in computational efficiency.

Introduction

Competing risks time-to-event data arise frequently in biomedical research when subjects are at risk for more than one type of possibly correlated events or causes and the occurrence of one event precludes the others from happening. For example, one may wish to study time until first kidney transplant for kidney dialysis patients with end-stage renal disease. Terminating events such as death, renal function recovery, or discontinuation of dialysis are considered competing risks as their occurrence will prevent subjects from receiving a transplant. When modeling competing risks data the cumulative incidence function (CIF), the probability of observing a certain cause while taking the competing risks into account, is oftentimes a quantity of interest.

The most commonly-used model to draw inference about the covariate effect on the CIF and to predict the CIF dependent on a set of covariates is the Fine-Gray proportional subdistribution hazards model (Fine and Gray, 1999). Various statistical packages for estimating the parameters of the Fine-Gray model are popular within the R programming language (Ihaka and Gentleman, 1996). One package, among others, is the **cmprsk** package. The **riskRegression** package, initially implemented for predicting absolute risks (Gerds et al., 2012), uses a wrapper that calls the **cmprsk** package to perform Fine-Gray regression. Scheike and Zhang (2011) provide **timereg** that allows for general modeling of the cumulative incidence function and includes the Fine-Gray model as a special case. The **survival** package also performs Fine-Gray regression but does so using a weighted Cox (Cox, 1972) model. Over the past decade, there have been several extensions to the Fine-Gray method that also result in useful packages. The **crrSC** package allows for the modeling of both stratified (Zhou et al., 2011) and clustered (Zhou et al., 2012) competing risks data. Kuk and Varadhan (2013) propose a stepwise Fine-Gray selection procedure and develop the **crrstep** package for implementation. Fu et al. (2017) then introduce penalized Fine-Gray regression with the corresponding **crrp** package.

A contributing factor to the computational complexity for general Fine-Gray regression implementation is parameter estimation. Generally, one needs to compute the log-pseudo likelihood and its first and second derivatives with respect to its regression parameters for optimization. Calculating these quantities is typically of order $O(n^2)$, where n is the number of observations in the dataset, due to the repeated calculation of the subject-specific risk sets. With current technological advancements making large-scale data from electronic health record (EHR) data systems routinely accessible to researchers, these implementations quickly become inoperable or grind-to-a-halt in this domain. For example, Kawaguchi et al. (2020) reported a runtime of about 24 hours to fit a LASSO regularized Fine-Gray regression on a subset of the United States Renal Data Systems (USRDS) with $n = 125,000$ subjects using an existing R package **crrp**. To this end, we note that for time-to-event data with no competing risks, Simon et al. (2011), Breheny and Huang (2011), and Mittal et al. (2014), among many others, have made significant progress in reducing the computational complexity for the Cox (1972) proportional hazards model from $O(n^2)$ to $O(n)$ by taking advantage of the cumulative structure of the risk set. However, the counterfactual construction of the risk set for the Fine-Gray model does not retain the same structure and presents a barrier to reducing the complexity of the risk set calculation. To the best of our knowledge, no further advancements in reducing the computational complexity required for calculating the subject-specific risk sets exists.

The contribution of this work is the development of an R package **fastcmprsk** which implements a novel forward-backward scan algorithm (Kawaguchi et al., 2020) for the Fine-Gray model. By taking advantage of the ordering of the data and the structure of the risk set, we can calculate the log-pseudo likelihood and its derivatives, which are necessary for parameters estimation, in $O(n)$ calculations rather than $O(n^2)$. As a consequence, our approach is scalable to large competing risks datasets and outperforms competing algorithms for both penalized and unpenalized parameter estimation.

The paper is organized as follows. In the next section, we briefly review the basic definition of the Fine-Gray proportional subdistribution hazards model, the CIF, and penalized Fine-Gray regression. We highlight the computational challenge of lineaizing estimation for the Fine-Gray model and introduce the forward-backward scan algorithm of Kawaguchi et al. (2020) in Section 2.3. Then in Section 2.4, we describe the main functionalities of the **fastcmprsk** package that we developed for R which utilizes the aforementioned algorithm for unpenalized and penalized parameter estimation and CIF estimation. We perform simulation studies in Section 2.5 to compare the performance of our proposed method to some of their popular competitors. The **fastcmprsk** package is readily available on the Comprehensive R Archive Network (CRAN) at `https://CRAN.R-project.org/package=fastcmprsk`.

Preliminaries

Data structure and model

We first establish some notation and the formal definition of the data generating process for competing risks. For subject $i = 1, \ldots, n$, let T_i, C_i, and ϵ_i be the event time, possible right-censoring time, and cause (event type), respectively. Without loss of generality assume there are two event types $\epsilon \in \{1, 2\}$ where $\epsilon = 1$ is the event of interest (or primary event) and $\epsilon = 2$ is the competing risk. With the presence of right-censoring we generally observe $X_i = T_i \wedge C_i$, $\delta_i = I(T_i \leq C_i)$, where $a \wedge b = \min(a, b)$ and $I(\cdot)$ is the indicator function. Letting $\mathbf{z}_i$ be a p-dimensional vector of time-independent subject-specific covariates, competing risks data consist of the following independent and identically distributed quadruplets $\{(X_i, \delta_i, \delta_i\epsilon_i, \mathbf{z}_i)\}_{i=1}^n$. Assume that there also exists a τ such that 1) for some arbitrary time t, $t \in [0, \tau]$; 2) $\Pr(T_i > \tau) > 0$ and $\Pr(C_i > \tau) > 0$ for all $i = 1, \ldots, n$, and that for simplicity, no ties are observed.

The CIF for the primary event conditional on the covariates $\mathbf{z} = (z_1, \ldots, z_p)$ is $F_1(t; \mathbf{z}) = \Pr(T \leq t, \epsilon = 1|\mathbf{z})$. To model the covariate effects on $F_1(t; \mathbf{z})$, Fine and Gray (1999) introduced the now well-appreciated proportional subdistribution hazards (PSH) model:

$$h_1(t|\mathbf{z}) = h_{10}(t)\exp(\mathbf{z}'\boldsymbol{\beta}), \tag{1}$$

where

$$\begin{aligned} h_1(t|\mathbf{z}) &= \lim_{\Delta t \to 0} \frac{\Pr\{t \leq T \leq t + \Delta t, \epsilon = 1 | T \geq t \cup (T \leq t \cap \epsilon \neq 1), \mathbf{z}\}}{\Delta t} \\ &= -\frac{d}{dt}\log\{1 - F_1(t; \mathbf{z})\} \end{aligned}$$

is a subdistribution hazard (Gray, 1988), $h_{10}(t)$ is a completely unspecified baseline subdistribution hazard, and $\boldsymbol{\beta}$ is a $p \times 1$ vector of regression coefficients. As Fine and Gray (1999) mentioned, the risk set associated with $h_1(t; \mathbf{z})$ is somewhat unnatural as it includes subjects who are still at risk $(T \geq t)$ and those who have already observed the competing risk prior to time t $(T \leq t \cap \epsilon \neq 1)$. However, this construction is useful for direct modeling of the CIF.

Parameter estimation for unpenalized Fine-Gray regression

Parameter estimation and large-sample inference of the PSH model follows from the log-pseudo likelihood:

$$l(\boldsymbol{\beta}) = \sum_{i=1}^{n} \int_0^\infty \left[\mathbf{z}_i'\boldsymbol{\beta} - \ln\left\{\sum_k \hat{w}_k(u)Y_k(u)\exp\left(\mathbf{z}_k'\boldsymbol{\beta}\right)\right\}\right] \hat{w}_i(u)dN_i(u), \tag{2}$$

where $N_i(t) = I(X_i \leq t, \epsilon_i = 1)$, $Y_i(t) = 1 - N_i(t-)$, and $\hat{w}_i(t)$ is a time-dependent weight based on the inverse probability of censoring weighting (IPCW) technique (Robins and Rotnitzky, 1992). To parallel Fine and Gray (1999), we define the IPCW for subject i at time t as $\hat{w}_i(t) = I(C_i \geq T_i \wedge t)\hat{G}(t)/\hat{G}(X_i \wedge t)$, where $G(t) = \Pr(C \geq t)$ is the survival function of the censoring variable C and $\hat{G}(t)$ is the Kaplan-Meier estimate for $G(t)$. We can further generalize the IPCW to allow for dependence between C and $\mathbf{z}$.

Let $\hat{\boldsymbol{\beta}}_{mple} = \arg\min_{\boldsymbol{\beta}}\{-l(\boldsymbol{\beta})\}$ be the maximum pseudo likelihood estimator of $\boldsymbol{\beta}$. Fine and Gray (1999) investigate the large-sample properties of $\hat{\boldsymbol{\beta}}_{mple}$ and prove that, under certain regularity conditions,

$$\sqrt{n}(\hat{\boldsymbol{\beta}}_{mple} - \boldsymbol{\beta}_0) \to N(0, \Omega^{-1}\Sigma\Omega^{-1}), \tag{3}$$

where $\boldsymbol{\beta}_0$ is the true value of $\boldsymbol{\beta}$, Ω is the limit of the negative of the partial derivative matrix of the score function evaluated at $\boldsymbol{\beta}_0$, and Σ is the variance-covariance matrix of the limiting distribution of the score function. We refer readers to Fine and Gray (1999) for more details on Ω and Σ. This variance estimation procedure is implemented in the **cmprsk** package.

Estimating the cumulative incidence function

An alternative interpretation of the coefficients from the Fine-Gray model is to model their effect on the CIF. Using a Breslow-type estimator (Breslow, 1974), we can obtain a consistent estimate for $H_{10}(t) = \int_0^t h_{10}(s)ds$ through

$$\hat{H}_{10}(t) = \frac{1}{n}\sum_{i=1}^{n}\int_0^t \frac{1}{\hat{S}^{(0)}(\hat{\boldsymbol{\beta}}, u)}\hat{w}_i(u)dN_i(u),$$

where $\hat{S}^{(0)}(\hat{\boldsymbol{\beta}}, u) = n^{-1}\sum_{i=1}^{n}\hat{w}_i(u)Y_i(u)\exp(\mathbf{z}_i'\hat{\boldsymbol{\beta}})$. The predicted CIF, conditional on $\mathbf{z} = \mathbf{z}_0$, is then

$$\hat{F}_1(t; \mathbf{z}_0) = 1 - \exp\left\{\int_0^t \exp(\mathbf{z}_0'\hat{\boldsymbol{\beta}})d\hat{H}_{10}(u)\right\}.$$

We refer the readers to Appendix B of Fine and Gray (1999) for the large-sample properties of $\hat{F}_1(t; \mathbf{z}_0)$. The quantities needed to estimate $\int_0^t d\hat{H}_{10}(u)$ are already precomputed when estimating $\hat{\boldsymbol{\beta}}$. Fine and Gray (1999) proposed a resampling approach to calculate confidence intervals and confidence bands for $\hat{F}_1(t; \mathbf{z}_0)$.

Penalized Fine-Gray regression for variable selection

Oftentimes reserachers are interested in identifying which covariates have an effect on the CIF. Penalization methods (Tibshirani, 1996; Fan and Li, 2001; Zou, 2006; Zhang et al., 2010) offer a popular way to perform variable selection and parameter estimation simultaneously through minimizing the objective function

$$Q(\boldsymbol{\beta}) = -l(\boldsymbol{\beta}) + \sum_{j=1}^{p} p_\lambda(|\beta_j|), \tag{4}$$

where $l(\boldsymbol{\beta})$ is defined in (2), $p_\lambda(|\beta_j|)$ is a penalty function where the sparsity of the model is controlled by the non-negative tuning parameter λ. Fu et al. (2017) recently extend several popular variable selection procedures - LASSO (Tibshirani, 1996), SCAD (Fan and Li, 2001), adaptive LASSO (Zou, 2006), and MCP (Zhang, 2010) - to the Fine-Gray model, explore its asymptotic properties under fixed model dimension, and develop the R package **crrp** (Fu, 2016) for implementation. Parameter estimation in the **crrp** package employs a cyclic coordinate algorithm.

The sparsity of the model depends heavily on the choice of the tuning parameters. Practically, finding a suitable (or optimal) tuning parameter involves applying a penalization method over a sequence of possible candidate values of λ and finding the λ that minimizes some metric such as the Bayesian information criterion (Schwarz, 1978) or generalized cross validation measure (Craven and Wahba, 1978). A more thorough discussion on tuning parameter selection can partially be found in Wang et al. (2007); Zhang et al. (2010); Wang and Zhu (2011); Fan and Tang (2013); Fu et al. (2017); Ni and Cai (2018).

Parameter estimation in linear time

Whether interest is in fitting an unpenalized model or a series of penalized models used for variable selection, one will need to minimize the negated log-pseudo (or penalized log-pseudo likelihood. While current implementations can readily fit small to moderately-sized datasets, where the sample size can be in the hundreds to thousands, we notice that these packages grind to a halt for large-scale data such as, electronic health records (EHR) data or cancer registry data, where the number of observations easily exceed tens of thousands, as illustrated later in Section 2.5.1 (Table 2) on some simulated large competing risks data.

The primary computational bottleneck for estimating the parameters of the Fine-Gray model is due to the calculation of the log-pseudo likelihood and its derivatives, which are required for commonly-used optimization routines. For example, the cyclic coordinate descent algorithm requires the score function

$$\dot{l}_j(\boldsymbol{\beta}) = \sum_{i=1}^{n} I(\delta_i\epsilon_i = 1)z_{ij} - \sum_{i=1}^{n} I(\delta_i\epsilon_i = 1)\frac{\sum_{k\in R_i} z_{kj}\tilde{w}_{ik}\exp(\eta_k)}{\sum_{k\in R_i} \tilde{w}_{ik}\exp(\eta_k)}, \tag{5}$$

and the Hessian diagonals

$$\ddot{l}_{jj}(\boldsymbol{\beta}) = \sum_{i=1}^{n} I(\delta_i\epsilon_i = 1)\left[\frac{\sum_{k\in R_i} z_{kj}^2\tilde{w}_{ik}\exp(\eta_k)}{\sum_{k\in R_i} \tilde{w}_{ik}\exp(\eta_k)} - \left\{\frac{\sum_{k\in R_i} z_{kj}\tilde{w}_{ik}\exp(\eta_k)}{\sum_{k\in R_i} \tilde{w}_{ik}\exp(\eta_k)}\right\}^2\right], \tag{6}$$

where

$$\tilde{w}_{ik} = \hat{w}_k(X_i) = \hat{G}(X_i)/\hat{G}(X_i \wedge X_k), \quad k \in R_i,$$

$R_i = \{y : (X_y \geq X_i) \cup (X_y \leq X_i \cap \epsilon_y = 2)\}$ and $\eta_k = \mathbf{z}_k'\boldsymbol{\beta}$ for optimization. While the algorithm itself is quite efficient, especially for estimating sparse coefficients, direct evaluation of (5) and (6) will require $O(n^2)$ operations since for each i such that $\delta_i\epsilon_i = 1$ we must identify all $y \in \{1,\ldots,n\}$ such that either $X_y \geq X_i$ or $(X_y \leq X_i \cap \epsilon_y = 2)$. As a consequence, parameter estimation will be computationally taxing for large-scale data since runtime will scale quadratically with n. We verify this in Section 2.5 for the **cmprsk** and **crrp** packages. To the best of our knowledge, prior to Kawaguchi et al. (2020), previous work on reducing the computational of parameter estimation from $O(n^2)$ to a lower order has not been developed.

Before moving forward we will first consider the Cox proportional hazards model for right-censored data, which can be viewed as a special case of the Fine-Gray model when competing risks are not present (i.e. $R_i = \{y : X_y \geq X_i\}$, $\tilde{w}_{ik} = 1$ for all $k \in R_i$, and $\epsilon_i = 1$ whenever $\delta_i = 1$). Again, direct calculation of quantities such as the log-partial likelihood and score function will still require $O(n^2)$ computations; however, one can show that when event times are arranged in decreasing order, the risk set is monotonically increasing as a series of cumulative sums. Once we arrange the event times in decreasing order, these quantities can be calculated in $O(n)$ calculations. The simplicity of the data manipulation and implementation makes this approach widely adopted in several R packages for right-censored data including the **survival**, **glmnet**, **ncvreg**, and **Cyclops** packages.

Unfortunately, the risk set associated with the Fine-Gray model does not retain the same cumulative structure. Kawaguchi et al. (2020) propose a novel forward-backward scan algorithm that reduces the computational complexity associated with parameter estimation from $O(pn^2)$ to $O(pn)$, allowing for the analysis of large-scale competing risks data in linear time. Briefly, the risk set R_i partitions into two disjoint subsets: $R_i(1) = \{y : X_y \geq X_i\}$ and $R_i(2) = \{y : (X_y \leq X_i \cap \epsilon_y = 2)\}$, were $R_i(1)$ is the set of observations that have an observed event time after X_i and $R_i(2)$ is the set of observations that have observed the competing event before time X_i. Since $R_i(1)$ and $R_i(2)$ are disjoint, the summation over $k \in R_i$ can be written as two separate summations, one over $R_i(1)$ and one over $R_i(2)$. The authors continue to show that the summation over $R_i(1)$ is a series of cumulative sums as the event times decrease while the summation over $R_i(2)$ is a series of cumulative sums as the event times increase. Therefore, by cleverly separating the calculation of both summations, (5), (6), and consequently (2) are available in $O(n)$ calculations. We will show the computational advantage of this approach for parameter estimation over competing R packages in Section 2.5.

The fastcmprsk package

We utilize this forward-backward scan algorithm of Kawaguchi et al. (2020) for both penalized and unpenalized parameter estimation for the Fine-Gray model in linear time. Furthermore, we also develop scalable methods to estimate the predicted CIF and its corresponding confidence interval/band. For convenience to researchers and readers, a function to simulate two-cause competing risks data is also included. Table ?? provides a summary of the currently available functions provided in **fastcmprsk**. We briefly detail the use of some of the key functions below.

Simulating competing risks data

Researchers can simulate two-cause competing risks data using the `simulateTwoCauseFineGrayModel` function in **fastcmprsk**. The data generation scheme follows a similar design to that of Fine and

Function name	Basic description
Modeling functions	
`fastCrr`	Fits unpenalized Fine-Gray regression and returns an object of class "`fcrr`"
`fastCrrp`	Fits penalized Fine-Gray regression and returns an object of class "`fcrrp`"
Utilities	
`Crisk`	Creates an object of class "`Crisk`" to be used as the response variable for `fastCrr` and `fastCrrp`
`varianceControl`	Options for bootstrap variance for `fastCrr`.
`simulateTwoCauseFineGrayModel`	Simulates two-cause competing risks data
S3 methods for "`fcrr`"	
`AIC`	Generic function for calculating AIC
`coef`	Extracts model coefficients
`confint`	Computes confidence intervals for parameters in the model
`logLik`	Extracts the model log-pseudo likelihood
`predict`	Predict the cumulative incidence function given `newdata` using model coefficients.
`summary`	Print ANOVA table
`vcov`	Returns bootstrapped variance-covariance matrix if `variance = TRUE`.
S3 methods for "`fcrrp`"	
`AIC`	Generic function for calculating AIC
`coef`	Extracts model coefficients for each tuning parameter λ.
`logLik`	Extracts the model log-pseudo likelihood for each tuning parameter λ.
`plot`	Plot coefficient path as a function of λ

Table 1: Currently available functions in **fastcmprsk** (v.1.1.0).

Gray (1999) and Fu et al. (2017). Given a design matrix $\mathbf{Z} = (\mathbf{z}_1', \ldots, \mathbf{z}_n')$, $\boldsymbol{\beta}_1$, and $\boldsymbol{\beta}_2$, let the cumulative incidence function for cause 1 (the event of interest) be defined as $F_1(t; \mathbf{z}_i) = \Pr(T_i \leq t, \epsilon_i = 1 | \mathbf{z}_i) = 1 - [1 - \pi\{1 - \exp(-t)\}]^{\exp(\mathbf{z}_i'\boldsymbol{\beta}_1)}$, which is a unit exponential mixture with mass $1 - \pi$ at ∞ when $\mathbf{z}_i = \mathbf{0}$ and where π controls the cause 1 event rate. The cumulative incidence function for cause 2 is obtained by setting $\Pr(\epsilon_i = 2 | \mathbf{z}_i) = 1 - \Pr(\epsilon_i = 1 | \mathbf{z}_i)$ and then using an exponential distribution with rate $\exp(\mathbf{z}_i'\boldsymbol{\beta}_2)$ for the conditional cumulative incidence function $\Pr(T_i \leq t | \epsilon_i = 2, \mathbf{z}_i)$. Censoring times are independently generated from a uniform distribution $U(u_{\min}, u_{\max})$ where $u_{\min}$ and $u_{\max}$ control the censoring percentage. Appendix .1 provides more details on the data generation process. Below is a toy example of simulating competing risks data where $n = 500$, $\boldsymbol{\beta}_1 = (0.40, -0.40, 0, -0.50, 0, 0.60, 0.75, 0, 0, -0.80)$, $\boldsymbol{\beta}_2 = -\boldsymbol{\beta}_1$, $u_{\min} = 0$, $u_{\max} = 1$, $\pi = 0.5$, and where $\mathbf{Z}$ is simulated from a multivariate standard normal distribution with unit variance. This simulated dataset will be used to illustrate the use of the different modeling functions within **fastcmprsk**. The purpose of the simulated dataset is to demonstrate the use of the **fastcmprsk** package and its comparative estimation performance to currently-used packages for unpenalized and penalized Fine-Gray regression. Runtime comparisons between the different packages are reported in Section 2.5.

```
R> #### Need the following packages to run the examples in the paper
R> install.packages("cmprsk")
R> install.packages("crrp")
R> install.packages("doParallel")
R> install.packages("fastcmprsk")
R> ###

R> library(fastcmprsk)
R> set.seed(2019)
R> N  <- 500 # Set number of observations

R> # Create coefficient vector for event of interest and competing event
```

```
R> beta1 <- c(0.40, -0.40,  0, -0.50,  0,  0.60,  0.75,  0,  0, -0.80)
R> beta2 <- -beta1

R> # Simulate design matrix
R> Z     <- matrix(rnorm(nobs * length(beta1)), nrow = N)

R> # Generate data
R> dat   <- simulateTwoCauseFineGrayModel(N, beta1, beta2,
+        Z, u.min = 0, u.max = 1, p = 0.5)

R> # Event counts (0 = censored; 1 = event of interest; 2 = competing event)
R> table(dat$fstatus)

  0   1   2
241 118 141

R> # First 6 observed survival times
R> head(dat$ftime)

[1] 0.098345608 0.008722629 0.208321175 0.017656904 0.495185038 0.222799124
```

fastCrr: Unpenalized parameter estimation and inference

We first illustrate the coefficient estimation from (1) using the Fine-Gray log-pseudo likelihood. The `fastCrr` function returns an object of class "`fcrr`" that estimates these parameters using our forward-backward scan algorithm and is syntactically similar to the `coxph` function in **survival**. The `formula` argument requires an outcome of class "`Crisk`". The `Crisk` function produces this object by calling the `Surv` function in **survival**, modifying it to allow for more than one event, and requires four arguments: a vector of observed event times (`ftime`), a vector of corresponding event/censoring indicators (`fstatus`), the value of `fstatus` that denotes a right-censored observation (`cencode`) and the value of `fstatus` that denotes the event of interest (`failcode`). By default, `Crisk` assumes that `cencode = 0` and `failcode = 1`. The `variance` passed into `fastCrr` is a logical argument that specifies whether or not the variance should be calculated with parameter estimation.

```
# cmprsk package
R> library(cmprsk)
R> fit1 <- crr(dat$ftime, dat$fstatus, Z, failcode = 1, cencode = 0,
+                    variance = FALSE)

# fastcmprsk package
R> fit2 <- fastCrr(Crisk(dat$ftime, dat$fstatus, cencode = 0, failcode = 1) ~ Z,
+                          variance = FALSE)

R> max(abs(fit1$coef - fit2$coef)) # Compare the coefficient estimates for both methods

[1] 8.534242e-08
```

As expected, the `fastCrr` function calculates nearly identical parameter estimates to the `crr` function. The slight difference in numerical accuracy can be explained by the different methods of optimization and convergence thresholds used for parameter estimation. Convergence within the cyclic coordinate descent algorithm used in `fastCrr` is determined by the relative change of the coefficient estimates. We allow users to modify the maximum relative change and maximum number of iterations used for optimization within `fastCrr` through the `eps` and `iter` arguments, respectively. By default, we set `eps = 1E-6` and `iter = 1000` in both our unpenalized and penalized optimization methods.

We now show how to obtain the variance-covariance matrix for the parameter estimates. The variance-covariance matrix for $\hat{\boldsymbol{\beta}}$ via (3) can not be directly estimated using the `fastCrr` function. First, the asymptotic expression requires estimating both Ω and Σ, which can not be trivially calculated in $O(pn)$ operations. Second, for large-scale data where both n and p can be large, matrix calculations, storage, and inversion can be computationally prohibitive. Instead, we propose to estimate the variance-covariance matrix using the bootstrap (Efron, 1979). Let $\tilde{\boldsymbol{\beta}}^{(1)}, \dots \tilde{\boldsymbol{\beta}}^{(B)}$ be bootstrapped parameter estimates obtained by resampling subjects with replacement from the original data B times. Unless otherwise noted, the size of each resample is the same as the original data. For $j = 1, \dots, p$ and

$k = 1, \ldots, p$, we can estimate the covariance between $\hat{\beta}_j$ and $\hat{\beta}_k$ by

$$\widehat{Cov}(\hat{\beta}_j, \hat{\beta}_k) = \frac{1}{B-1} \sum_{b=1}^{B} (\tilde{\beta}_j^{(b)} - \bar{\beta}_j)(\tilde{\beta}_k^{(b)} - \bar{\beta}_k), \tag{7}$$

where $\bar{\boldsymbol{\beta}}_j = \frac{1}{B} \sum_{b=1}^{B} \tilde{\boldsymbol{\beta}}_j^{(b)}$. Therefore, with $\hat{\sigma}_j^2 = \widehat{Cov}(\hat{\beta}_j, \hat{\beta}_j)$, a $(1-\alpha) \times 100\%$ confidence interval for $\boldsymbol{\beta}_j$ is given by

$$\hat{\beta}_j \pm z_{1-\alpha/2} \hat{\sigma}_j, \tag{8}$$

where $z_{1-\alpha/2}$ is the $(1-\alpha) \times 100th$ percentile of the standard normal distribution. Since parameter estimation for the Fine-Gray model is done in linear time using our forward-backward scan algorithm, the collection of parameter estimates obtained by bootstrapping can also be obtained linearly. The `varianceControl` function controls the parameters used for bootstrapping, that one then passes into the `var.control` argument in `fastCrr`. These arguments include `B`, the number of bootstrap samples to be used, and `seed`, a non-negative numeric integer to set the seed for resampling.

```
R> # Estimate variance via 100 bootstrap samples using seed 2019.
R> vc   <- varianceControl(B = 100, seed = 2019)
R> fit3 <- fastcmprsk::fastCrr(Crisk(dat$ftime, dat$fstatus) ~ Z, variance = TRUE,
+                               var.control = vc,
+                               returnDataFrame = TRUE)
# returnDataFrame = TRUE is necessary for CIF estimation (next section)

R> round(sqrt(diag(fit3$var)), 3) # Standard error estimates rounded to 3rd decimal place

[1] 0.108 0.123 0.085 0.104 0.106 0.126 0.097 0.097 0.104 0.129
```

The accuracy of the bootstrap variance-covariance matrix compared to the asymptotic expression depends on several factors including the sample size and number of bootstrap samples B. Our empirical evidence in Section 2.5.1 show that $B = 100$ bootstrap samples provided a sufficient estimate of the variance-covariance matrix for large enough n in our scenarios. In practice, we urge users to increase the number of bootstrap samples until the variance is stable if they can computationally afford to. Although this may hinder the computational performance of `fastCrr` for small sample sizes, we find this to be a more efficient approach for large-scale competing risks data.

We adopt several S3 methods that work seamlessly with the `"fcrr"` object that is outputted from `fastCrr`. The `coef` method returns the estimated regression coefficient estimates $\hat{\boldsymbol{\beta}}$:

```
R> coef(fit3) # Coefficient estimates

[1] 0.192275755 -0.386400287  0.018161906 -0.397687129  0.105709092  0.574938015
[7] 0.778842652 -0.006105756 -0.065707434 -0.996867883
```

The model pseudo log-likelihood can also be extracted via the `logLik` function:

```
R> logLik(fit3) # Model log-pseudo likelihood
[1] -590.3842
```

Related quantities to the log-pseudo likelihood are information criteria, measures of the quality of a statistical model that are used to compare alternative models on the same data. These criterion are computed using the following formula: $-2l(\hat{\boldsymbol{\beta}}) + k \times |\hat{\boldsymbol{\beta}}|_0$, where k is a penalty factor for model complexity and $|\hat{\boldsymbol{\beta}}|_0$ corresponds to the number of parameters in the model. Information criteria can be computed for a `fcrr` object using `AIC` and users specify the penalty factor using the `k` argument. By default `k = 2` and corresponds to the Akaike information criteria (Akaike, 1974).

```
R> AIC(fit3, k = 2) # Akaike's Information Criterion
[1] 1200.768

R> # Alternative expression of the AIC
R> -2 * logLik(fit3) + 2 * length(coef(fit3))
[1] 1200.768
```

If the `variance` is set to `TRUE` for the `fastCrr` model fit, we can extract the bootstrap variance-covariance matrix using `vcov`. Additionally, `conf.int` will display confidence intervals, on the scale of $\hat{\boldsymbol{\beta}}$, and the `level` argument can be used to specify the confidence level. By default `level = 0.95` and corresponds to 95% confidence limits.

```
R> vcov(fit3)[1:3, 1:3] # Variance-covariance matrix for the first three estimates

             [,1]         [,2]         [,3]
[1,] 0.0116785745 0.0031154634 0.0007890851
[2,] 0.0031154634 0.0150597898 0.0004681825
[3,] 0.0007890851 0.0004681825 0.0072888011


R> confint(fit3, level = 0.95) # 95 % Confidence intervals

           2.5%      97.5%
x1  -0.01953256  0.4040841
x2  -0.62692381 -0.1458768
x3  -0.14916899  0.1854928
x4  -0.60197206 -0.1934022
x5  -0.10199838  0.3134166
x6   0.32827237  0.8216037
x7   0.58798896  0.9696963
x8  -0.19610773  0.1838962
x9  -0.26995659  0.1385417
x10 -1.24897861 -0.7447572
```

Lastly, summary will return an ANOVA table for the fitted model. The table presents the log-subdistribution hazard ratio (coef), the subdistribution hazard ratio (exp(coef)), the standard error of the log-subdistribution hazards ratio (se(coef)) if `variance = TRUE` in `fastCrr`, the corresponding z-score (z value), and two-sided p-value (Pr(|z|)). When setting `conf.int = TRUE`, the summary function will also print out the 95% confidence intervals (if `variance = TRUE` when running `fastCrr`). Additionally the pseudo log-likelihood for the estimated model and the null pseudo log-likelihood (when $\hat{\boldsymbol{\beta}} = \mathbf{0}$) are also reported below the ANOVA table.

```
R> # ANOVA table for fastCrr
R> summary(fit3, conf.int = TRUE) # conf.int = TRUE allows for 95% CIs to be presented

Fine-Gray Regression via fastcmprsk package.

fastCrr converged in 24 iterations.

Call:
fastcmprsk::fastCrr(Crisk(dat$ftime, dat$fstatus) ~ Z, variance = TRUE,
    var.control = vc, returnDataFrame = TRUE)

        coef exp(coef) se(coef) z value Pr(>|z|)
x1   0.19228     1.212   0.1081   1.779  7.5e-02
x2  -0.38640     0.679   0.1227  -3.149  1.6e-03
x3   0.01816     1.018   0.0854   0.213  8.3e-01
x4  -0.39769     0.672   0.1042  -3.816  1.4e-04
x5   0.10571     1.111   0.1060   0.997  3.2e-01
x6   0.57494     1.777   0.1259   4.568  4.9e-06
x7   0.77884     2.179   0.0974   7.998  1.3e-15
x8  -0.00611     0.994   0.0969  -0.063  9.5e-01
x9  -0.06571     0.936   0.1042  -0.631  5.3e-01
x10 -0.99687     0.369   0.1286  -7.750  9.1e-15

    exp(coef) exp(-coef)  2.5% 97.5%
x1      1.212      0.825 0.981 1.498
x2      0.679      1.472 0.534 0.864
x3      1.018      0.982 0.861 1.204
x4      0.672      1.488 0.548 0.824
x5      1.111      0.900 0.903 1.368
x6      1.777      0.563 1.389 2.274
x7      2.179      0.459 1.800 2.637
x8      0.994      1.006 0.822 1.202
x9      0.936      1.068 0.763 1.149
x10     0.369      2.710 0.287 0.475
Pseudo Log-likelihood = -590
```

```
Null Pseudo Log-likelihood = -675
Pseudo likelihood ratio test = 170 on 10 df.
```

Since standard error estimation is performed via bootstrap and resampling, it is easy to use multiple cores to speed up computation. Parallelization is seamlessly implemented using the **doParallel** package (Calaway et al., 2019). Enabling usage of multiple cores is done through the `useMultipleCores` argument within the `varianceControl` function. To avoid interference with other processes, we allow users to set up the cluster on their own. We provide an example below.

```
R> library(doParallel)

R> n.cores <- 2 # No. of cores
R> myClust <- makeCluster(n.cores)

R> # Set useMultipleCores = TRUE to enable parallelization
R> vc = varianceControl(B = 1000, useMultipleCores = TRUE)

R> registerDoParallel(myClust)
R> fit3 <- fastCrr(Crisk(dat$ftime, dat$fstatus) ~ Z, variance = TRUE,
+                             var.control = vc)
R> stopCluster(myClust)
```

Cumulative incidence function and interval/band estimation

The CIF is also available in linear time in the **fastcmprsk** package. Fine and Gray (1999) propose a Monte Carlo simulation method for interval and band estimation. We implement a slightly different approach using bootstrapping for interval and band estimation in our package. Let $\tilde{F}_1^{(1)}(t; \mathbf{z}_0), \ldots, \tilde{F}_1^{(B)}(t; \mathbf{z}_0)$ be the bootstrapped predicted CIF obtained by resampling subjects with replacement from the original data B times and let $m(\cdot)$ be a known, monotone, and continuous transformation. In our current implementation we let $m(x) = \log\{-\log(x)\}$; however, we plan on incorporating other transformations in our future implementation. We first estimate the variance function $\sigma^2(t; \mathbf{z}_0)$ of the transformed CIF through

$$\hat{\sigma}^2(t; \mathbf{z}_0) = \frac{1}{B} \sum_{b=1}^{B} \left[m\{\tilde{F}_1^{(b)}(t; \mathbf{z}_0)\} - \bar{m}\{\tilde{F}_1(t; \mathbf{z}_0)\} \right]^2, \tag{9}$$

where $\bar{m}\{\tilde{F}_1(t; \mathbf{z}_0)\} = \frac{1}{B} \sum_{b=1}^{B} m\{\tilde{F}_1^{(b)}(t; \mathbf{z}_0)\}$. Using the functional delta method, we can now construct $(1 - \alpha) \times 100\%$ confidence intervals for $F_1(t; \mathbf{z}_0)$ by

$$m^{-1} \left[m\{\hat{F}_1(t; \mathbf{z}_0)\} \pm z_{1-\alpha/2} \hat{\sigma}(t; \mathbf{z}_0) \right]. \tag{10}$$

Next we propose a symmetric global confidence band for the estimated CIF $\hat{F}_1(t; \mathbf{z}_0)$, $t \in [t_L, t_U]$ via bootstrap. We first determine a critical region $C_{1-\alpha}(\mathbf{z}_0)$ such that

$$\Pr\left\{ \sup_{t \in [t_L, t_U]} \frac{|m\{\hat{F}_1(t; \mathbf{z}_0)\} - m\{F_1(t; \mathbf{z}_0)\}|}{\sqrt{\widehat{Var}[m\{\hat{F}_1(t; \mathbf{z}_0)\}]}} \leq C_{1-\alpha}(\mathbf{z}_0) \right\} = 1 - \alpha. \tag{11}$$

While Equation (9) estimates $\widehat{Var}[m\{\hat{F}_1(t; \mathbf{z}_0)\}]$ we still need to find $C_{1-\alpha}(\mathbf{z}_0)$ by the bootstrap $(1 - \alpha)^{th}$ percentile of the distribution of the supremum in the equation above. The algorithm is as follows:

1. Resample subjects with replacement from the original data B times and estimate $\tilde{F}_1^{(b)}(t; \mathbf{z}_0)$ for $b = 1, \ldots, B$ and $\hat{\sigma}^2(t; \mathbf{z}_0)$ using (9).
2. For the b^{th} bootstrap sample , $b \in \{1, \ldots, B\}$, calculate

$$C^{(b)} = \sup_{t \in [t_L, t_U]} \frac{|m\{\tilde{F}_1^{(b)}(t; \mathbf{z}_0)\} - m\{\hat{F}_1(t; \mathbf{z}_0)\}|}{\hat{\sigma}(t; \mathbf{z}_0)}.$$

3. Estimate $C_{1-\alpha}(\mathbf{z}_0)$ from the sample $(1 - \alpha)^{th}$ percentile of the B values of $C^{(b)}$, denoted by $\hat{C}_{1-\alpha}(\mathbf{z}_0)$.

Finally, the $(1 - \alpha) \times 100\%$ confidence band for $F_1(t; \mathbf{z}_0)$, $t \in [t_L, t_U]$ is given by

$$m^{-1} \left[m\{\hat{F}_1(t; \mathbf{z}_0)\} \pm \hat{C}_{1-\alpha}(\mathbf{z}_0) \hat{\sigma}(t; \mathbf{z}_0) \right]. \tag{12}$$

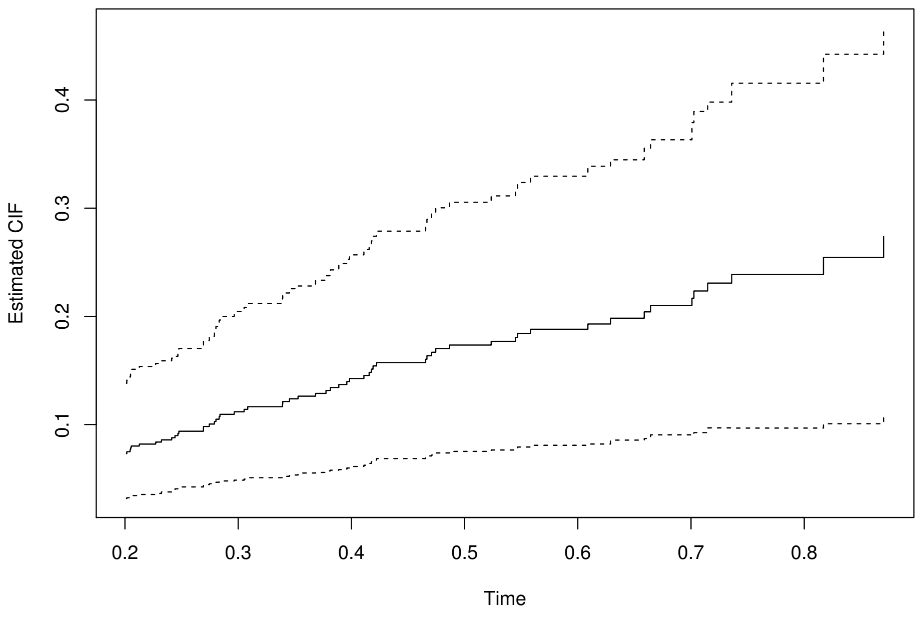


Figure 1: Estimated CIF (solid line) and corresponding 95% confidence intervals (dotted lines) between $t_L = 0.2$ and $t_U = 0.9$ given a covariate vector `z0` using the coefficient and baseline estimates from our toy example.

Similar to estimating the variance-covariance matrix for the coefficient estimates $\hat{\boldsymbol{\beta}}$, specifying the number of bootstrap samples, seed for reputability, and multicore functionality for estimating the variance of the CIF can be done through the `varianceControl` function. One can perform CIF estimation and interval/band estimation using the `predict` function by specifying a vector $\mathbf{z}_0$ in the `newdata` argument and the fitted model from `fastCrr`. To calculate the CIF, both the Breslow estimator of the cumulative subdistribution hazard and the (ordered) model data frame need to be returned values within the fitted object. This can be achieved by setting both the `getBreslowJumps` and `returnDataFrame` arguments within `fastCrr` to `TRUE`. Additionally, for confidence band estimation one must specify a time interval $[t_L, t_U]$. The user can specify the interval range using the `tL` and `tU` arguments in `predict`. Figure 1 illustrates the estimated CIF and corresponding 95% confidence interval, obtained using 100 bootstrap samples, over the range $[0.2, 0.9]$ given covariate entries `z0` simulated from a standard random normal distribution.

```
R> set.seed(2019)
R> # Make sure getBreslowJumps and returnDataFrame are set to TRUE
R> fit4 <- fastCrr(Crisk(dat$ftime, dat$fstatus, cencode = 0, failcode = 1) ~ Z,
+                           variance = FALSE,
+                           getBreslowJumps = TRUE, # Default = TRUE
+                           returnDataFrame = TRUE) # Default is FALSE for storage purposes

R> z0 <- rnorm(10) # New covariate entries to predict
R> cif.point <- predict(fit4, newdata = z0, getBootstrapVariance = TRUE,
+                      type = "interval", tL = 0.2, tU = 0.9,
+                      var.control = varianceControl(B = 100, seed = 2019))

R> plot(cif.point) # Figure 1 (Plot of CIF and 95% C.I.)
```

fastCrrp: Penalized Fine-Gray regression in linear time

We extend our forward-backward scan approach for for penalized Fine-Gray regression as described in Section 2.2.4. The `fastCrrp` function performs LASSO, SCAD, MCP, and ridge (Hoerl and Kennard, 1970) penalization. Users specify the penalization technique through the `penalty` argument. The advantage of implementing this algorithm for penalized Fine-Gray regression is two fold. Since the

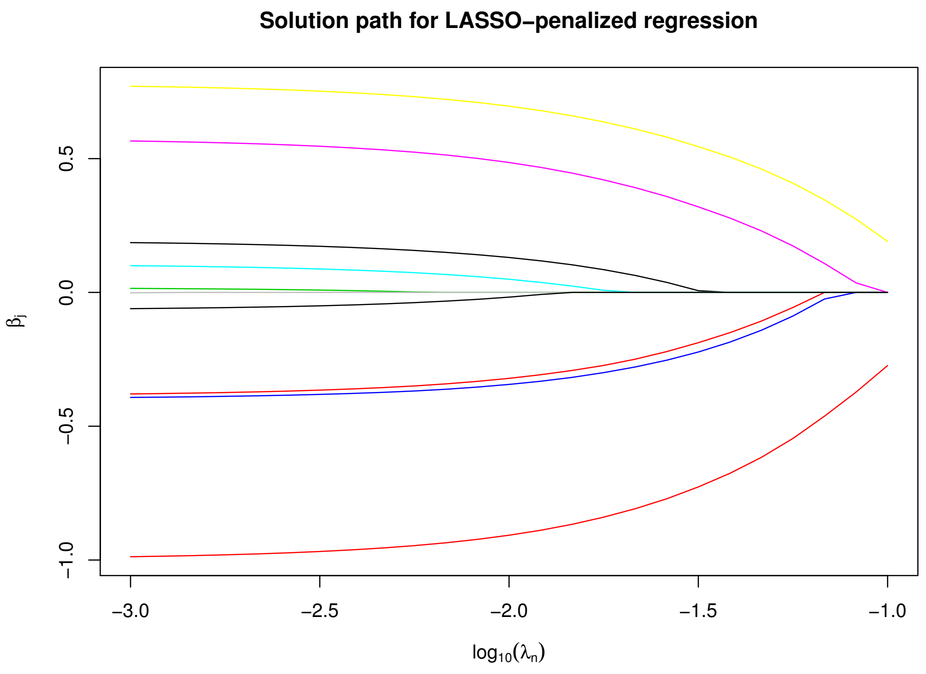


Figure 2: Path plot for LASSO-penalized Fine-Gray regression using our toy example. The tuning parameter λ varies between the log-spaced interval $[0.001, 0.1]$. The y-axis corresponds to the estimated value for $\hat{\beta}_j$ and the x-axis corresponds to λ (on the $\log_{10}$ scale).

cyclic coordinate descent algorithm used in the `crrp` function calculates the gradient and Hessian diagonals in $O(pn^2)$ time, as opposed to $O(pn)$ using our approach, we expect to see drastic differences in runtime for large sample sizes. Second, as mentioned earlier, researchers generally tune the strength of regularization through multiple model fits over a grid of candidate tuning parameter values. Thus the difference in runtime between both methods grows larger as the number of candidate values increases. Below we provide an example of performing LASSO-penalized Fine-Gray regression using a prespecified grid of 25 candidate values for λ that we input into the `lambda` argument of `fastCrrp`. If left untouched (i.e. `lambda = NULL`), a log-spaced interval of λ will be computed such that the largest value of λ will correspond to a null model. Figure 2 illustrates the solution path for the LASSO-penalized regression, a utility not directly implemented within the **crrp** package. The syntax for `fastCrrp` is nearly identical to the syntax for `crrp`.

```
R> library(crrp)
R> lam.path <- 10^seq(log10(0.1), log10(0.001), length = 25)

R> # crrp package
R> fit.crrp <- crrp(dat$ftime, dat$fstatus, Z, penalty = "LASSO",
+                        lambda = lam.path, eps = 1E-6)

R> # fastcmprsk package
R> fit.fcrrp <- fastCrrp(Crisk(dat$ftime, dat$fstatus) ~ Z, penalty = "LASSO",
+                                   lambda = lam.path)


R> # Check to see the two methods produce the same estimates.
R> max(abs(fit.fcrrp$coef - fit.crrp$beta))

[1] 1.110223e-15

R> plot(fit.fcrrp) # Figure 2 (Solution path)
```

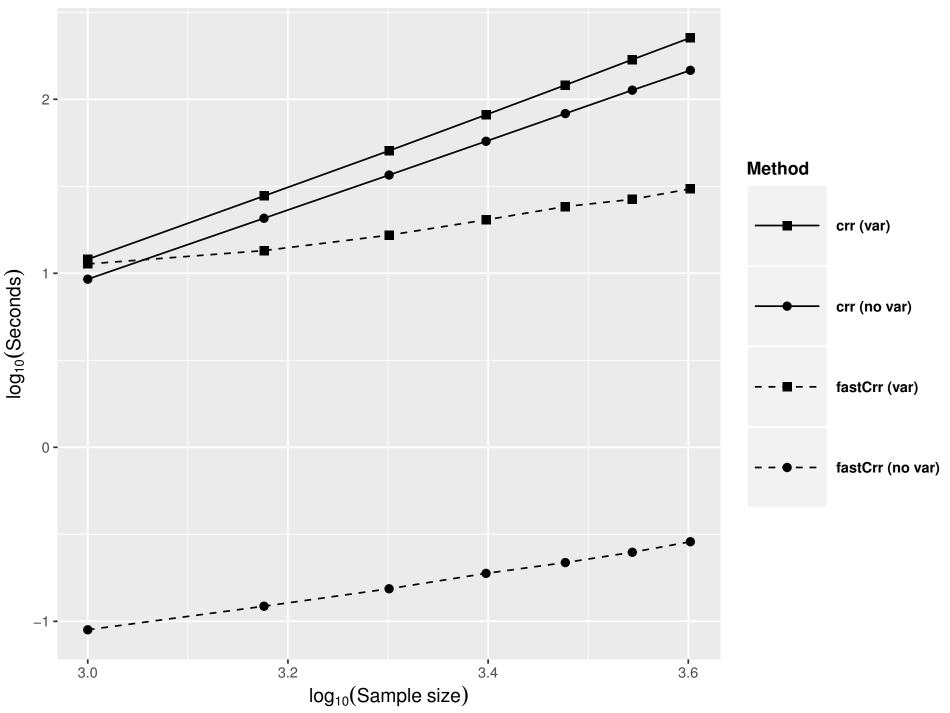


Figure 3: Runtime comparison between `fastCrr` and `crr` with and without variance estimation. Axes are on the $\log_{10}$ scale. Solid and dashed lines represent the **crrp** and **fastcmprsk** implementation, respectively. Square, and circle symbols denote variance and without variance calculation, respectively. Variance estimation for `crr` is performed using the asymptotic expression of the variance-covariance estimator. Variance estimation for `fastCrr` is performed using 100 bootstrap samples. Reported runtime are averaged over 100 Monte Carlo runs.

Simulation studies

This section provides a more comprehensive illustration of the computational performance of the **fastcmprsk** package over two popular competing packages **cmprsk** and **crrp**. We simulate datasets under various sample sizes and fix the number of covariates $p = 100$. We generate the design matrix, $\mathbf{Z}$ from a p-dimensional standard normal distribution with mean zero, unit variance, and pairwise correlation $\text{corr}(z_i, z_j) = \rho^{|i-j|}$, where $\rho = 0.5$ simulates moderate correlation. For Section 2.5.1, the vector of regression parameters for cause 1, the cause of interest, is $\boldsymbol{\beta}_1 = (\boldsymbol{\beta}^*, \boldsymbol{\beta}^*, \ldots, \boldsymbol{\beta}^*)$, where $\boldsymbol{\beta}^* = (0.40, -0.40, 0, -0.50, 0, 0.60, 0.75, 0, 0, -0.80)$. For Section 2.5.2, $\boldsymbol{\beta}_1 = (\boldsymbol{\beta}^*, \mathbf{0}_{p-10})$. We let $\boldsymbol{\beta}_2 = -\boldsymbol{\beta}_1$. We set $\pi = 0.5$, which corresponds to a cause 1 event rate of approximately 41%. The average censoring percentage for our simulations varies between $30 - 35\%$. We use `simulateTwoCauseFineGrayModel` to simulate these data and average runtime results over 100 Monte Carlo replicates. We report timing on a system with an Intel Core i5 2.9 GHz processor and 16GB of memory.

Comparison to the crr package

In this section, we compare the runtime and estimation performance of the `fastCrr` function to `crr`. We vary n from $1,000$ to $500,000$ and run `fastCrr` and `crr` both with and without variance estimation. We take 100 bootstrap samples, without parallelization, to obtain the bootstrap standard errors with `fastCrr`. As shown later in the section (Tables 3 and 4), 100 bootstrap samples suffices to produce a good standard error estimate with close-to-nominal coverage for large enough sample sizes in our scenarios. In practice, we recommend users to increase the number of bootstrap samples until the variance estimate becomes stable, when computationally feasible.

Figure 3 depicts how fast the computational complexities of `fastCrr` (dashed lines) and `crr` (solid lines) increase as n increases as measured by runtime (in seconds). It shows clearly that the computational complexity of `crr` increases quadratically (solid line slopes ≈ 2) while that of `fastCrr` is linear (dashed line slopes ≈ 1). This implies that the computational gains of `fastCrr` over `crr` are expected to grow exponentially as the sample size increases.

We further demonstrates the computational advantages of `fastCrr` over `crr` for large sample size

Table 2: Runtime comparison of `crr` versus `fasrCrr` for large n scenarios. The dashes ("–") indicate that runtime could not be completed within 72 hours. Variance estimation for `fastCrr` is calculated using $B = 100$ bootstrap samples.

	Sample size n		
	50,000	100,000	500,000
`crr` without variance	6 hours	24 hours	–
`crr` with variance	54 hours	–	–
`fastCrr` without variance	5 seconds	12 seconds	50 seconds
`fastCrr` with variance	7 minutes	14 minutes	69 minutes

Table 3: Standard error estimates for various different values of β_{1j} ($j = 1, 2, 3$). Empirical: Standard deviation of the 100 Monte Carlo estimates of $\hat{\beta}_{1j}$; Bootstrap: The average of the 100 Monte Carlo estimates of the bootstrap standard error for $\hat{\beta}_{1j}$ using $B = 100$ bootstrap samples; Asymptotic: The average of 100 Monte Carlo estimates of the standard error estimate for $\hat{\beta}_{1j}$ using the asymptotic variance-covariance matrix defined in (3).

	Std. Err. Est.	$n = 1000$	2000	3000	4000
$\beta_{11} = 0.4$	Empirical	0.06	0.05	0.04	0.03
	Bootstrap	0.10	0.05	0.04	0.03
	Asymptotic	0.07	0.04	0.03	0.03
$\beta_{12} = -0.4$	Empirical	0.10	0.05	0.04	0.03
	Bootstrap	0.11	0.06	0.04	0.04
	Asymptotic	0.08	0.05	0.04	0.03
$\beta_{13} = 0$	Empirical	0.09	0.06	0.04	0.03
	Bootstrap	0.11	0.06	0.04	0.04
	Asymptotic	0.07	0.05	0.04	0.03

data in Table 2 by comparing their runtime on a single simulated data with n varying from $50,000$ to $500,000$ using a system with an Intel Xeon 2.40GHz processor and 256GB of memory. It is seen that `fastCrr` scales well to large sample size data, whereas `crr` eventually grinds to a halt as n grows large. For example, for $n = 500,000$, it only takes less than 1 minute for `fastCrr` to finish, while `crr` did not finish in 3 days. Because the forward-backward scan allows us to efficiently compute variance estimates through bootstrapping, we have also observed massive computational gains in variance estimation with large sample size data (7 minutes for `fastCrr` versus 54 hours for `crr`). Furthermore, since parallelization of the bootstrap procedure was not implemented in these timing reports, we expect multicore usage to further decrease the runtime of the variance estimation for `fastCrr`

We also performed a simulation to compare the bootstrap procedure for variance estimation to the estimate of the asymptotic variance provided in (3) used in `crr`. First, we compare the two standard error estimates with the empirical standard error of $\hat{\boldsymbol{\beta}}_1$. For the j^{th} coefficient, the empirical standard error is calculated as the standard deviation of $\hat{\beta}_{1j}$ from the 100 Monte Carlo runs. For the standard error provided by both the bootstrap and the asymptotic variance-covariance matrix, we take the average standard error of $\hat{\beta}_{1j}$ over the 100 Monte Carlo runs. Table 3 compares the standard errors for $\hat{\beta}_{1j}$ for $j = 1, 2, 3$. When $n = 1000$, the average standard error using the bootstrap is slightly larger than the empirical standard error; whereas, the standard error from the asymptotic expression is slightly smaller. These differences diminish and all three estimates are comparable when $n \geq 2000$. This provides evidence that both the bootstrap and asymptotic expression are adequate estimators of the variance-covariance expression for large datasets.

Additionally, we present in Table 4 the coverage probability (and standard errors) of the 95% confidence intervals for $\beta_{11} = 0.4$ using the bootstrap (`fastCrr`) and asymptotic (`crr`) variance estimate. The confidence intervals are wider for the bootstrap approach when compared to confidence intervals produced using the asymptotic variance estimator, especially when $n = 1000$. However, both methods are close to the nominal 95% level as n increases. We observe similar trends across the other coefficient estimates.

Table 4: Coverage probability (and standard errors) of 95% confidence intervals for $\beta_{11} = 0.4$. Confidence intervals for `crr` are calculated using the asymptotic expression of the variance-covariance estimator. Confidence intervals for `fasrCrr` are calculated using the bootstrap variance-covariance estimator using 100 bootstrap samples.

	$n = 1000$	2000	3000	4000
`crr`	0.93 (0.03)	0.90 (0.03)	0.93 (0.03)	0.95 (0.02)
`fastCrr`	1.00 (0.00)	0.98 (0.02)	0.95 (0.02)	0.95 (0.02)

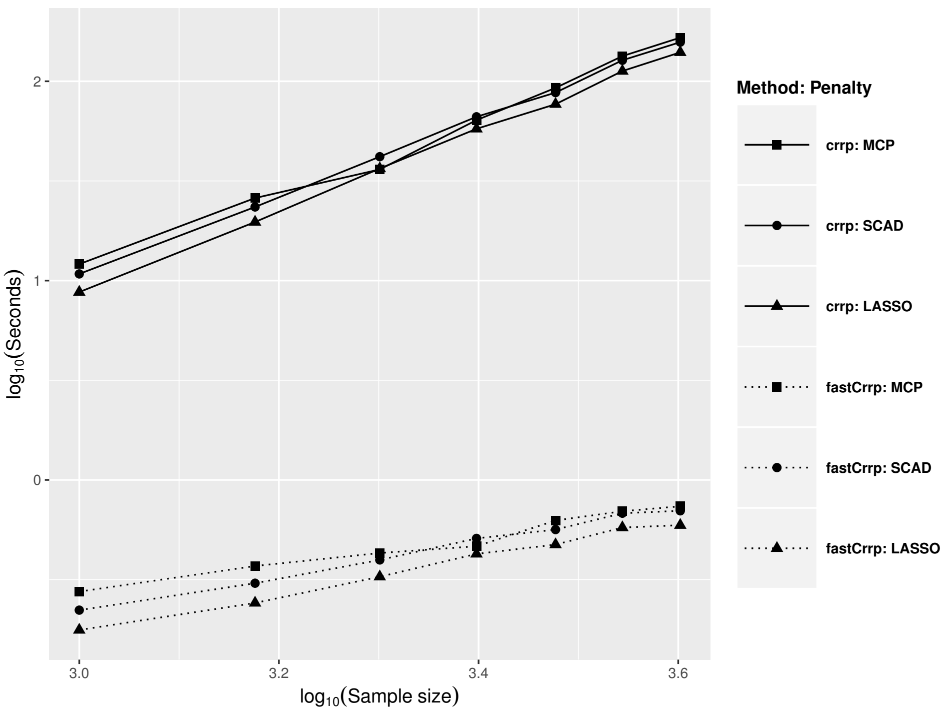


Figure 4: Runtime comparison between the **crrp** and **fastcmprsk** implementations of LASSO, SCAD, and MCP penalization. Solid and dashed lines represent the **crrp** and **fastcmprsk** implementation, respectively. Square, circle, and triangle symbols denote the penalties MCP, SCAD, and LASSO, respectively. Axes are on the $\log_{10}$ scale. Reported runtime are averaged over 100 Monte Carlo runs.

Comparison to the crrp package

As mentioned in Section 2.2.4, Fu et al. (2017) provide an R package **crrp** for performing penalized Fine-Gray regression using the LASSO, SCAD, and MCP penalties. We compare the runtime between `fastCrrp` with the implementation in the **crrp** package. To level comparisons, we modify the source code in `crrp` so that the function only calculates the coefficient estimates and BIC score. We vary $n = 1000, 1500, \ldots, 4000$, fix $p = 100$, and employ a 25-value grid search for the tuning parameter. Figure 4 illustrates the computational advantage the `fastCrrp` function has over `crrp`.

Similar to the unpenalized scenario, the computational performance of `crrp` (solid lines) increases quadratically while `fasrCrrp` (dashed lines) increases linearly, resulting in a 200 to 300-fold speed up in runtime when $n = 4000$. This, along with the previous section and a real data analysis conclusion in the following section, strongly suggests that for large-scale competing risks datasets (e.g. EHR databases), where the sample size can easily exceed tens to hundreds of thousands, analyses that may take several hours or days to perform using currently-implemented methods are available within seconds or minutes using the **fastcmprsk** package.

Discussion

The **fastcmprsk** package provides a set of scalable tools for the analysis of large-scale competing risks data by developing an approach to linearize the computational complexity required to estimate the parameters of the Fine-Gray proportional subdistribution hazards model. Multicore use is also implemented to further speed up methods that require bootstrapping and resampling. Our simulation results show that our implementation results in a up to 7200-fold decrease in runtime for large sample size data. We also note that in a real-world application, Kawaguchi et al. (2020) record a drastic decrease in runtime ($\approx$ 24 hours vs. $\approx$ 30 seconds) when comparing the proposed implementation of LASSO, SCAD, and MCP to the methods available in **crrp** on a subset of the United States Renal Data Systems (USRDS) where $n = 125,000$. The package implements both penalized and unpenalized Fine-Gray regression and we can conveniently extend our forward-backward algorithm to other applications such as stratified and clustered Fine-Gray regression.

Lastly, our current implementation assumes that covariates are densely observed across subjects. This is problematic in the sparse high-dimensional massive sample size (sHDMSS) domain (Mittal et al., 2014) where the number of subjects and sparsely-represented covariates easily exceed tens of thousands. These sort of data are typical in large comparative effectiveness and drug safety studies using massive administrative claims and EHR databases and typically contain millions to hundreds of millions of patient records with tens of thousands patient attributes, which such settings are particularly useful for drug safety studies of a rare event such as unexpected adverse events (Schuemie et al., 2018) to protect public health. We are currently extending our algorithm to this domain in a sequel paper.

Acknowledgements

We thank the referees and the editor for their helpful comments that improved the presentation of the article. Marc A. Suchard's work is partially supported through the National Institutes of Health grant U19 AI 135995. Jenny I. Shen's work is partly supported through the National Institutes of Health grant K23DK103972. The research of Gang Li was partly supported by National Institutes of Health Grants P30 CA-16042, UL1TR000124-02, and P50 CA211015.

Data generation scheme

We describe the data generation process for the `simulateTwoCauseFineGrayModel` function. Let n, p, $\mathbf{Z}_{n\times p}$, $\boldsymbol{\beta}_1$, $\boldsymbol{\beta}_2$, $u_{\min}$, $u_{\max}$ and π be specified. We first generate independent Bernoulli random variables to simulate the cause indicator ϵ for each subject. That is, $\epsilon_i \sim 1 + Bern\{(1-\pi)^{\exp(\mathbf{z}_i'\boldsymbol{\beta}_1)}\}$ for $i = 1, \ldots, n$. Then, conditional on the cause, event times are simulated from

$$\Pr(T_i \le t | \epsilon_i = 1, \mathbf{z}_i) = \frac{1 - [1 - \pi\{1 - \exp(-t)\}]^{\exp(\mathbf{z}_i'\boldsymbol{\beta}_1)}}{1 - (1-\pi)^{\exp(\mathbf{z}_i'\boldsymbol{\beta}_1)}}$$
$$\Pr(T_i \le t | \epsilon_i = 2, \mathbf{z}_i) = 1 - \exp\{-t\exp(\mathbf{z}_i'\boldsymbol{\beta}_2)\},$$

and $C_i \sim U(u_{\min}, u_{\max})$. Therefore, for $i = 1, \ldots, n$, we can obtain the following quadruplet $\{(X_i, \delta_i, \delta_i\epsilon_i, \mathbf{z}_i)\}$ where $X_i = \min(T_i, C_i)$, and $\delta_i = I(X_i \le C_i)$. Below is an excerpt of the code used in `simulateTwoCauseFineGrayModel` to simulate the observed event times, cause and censoring indicators.

```
#START CODE
...
...
...
# nobs, Z, p = pi, u.min, u.max, beta1 and beta2 are already defined.
# Simulate cause indicators here using a Bernoulli random variable
c.ind <- 1 + rbinom(nobs, 1, prob = (1 - p)^exp(Z %*% beta1))

ftime <- numeric(nobs)
eta1 <- Z[c.ind == 1, ] %*% beta1 #linear predictor for cause on interest
eta2 <- Z[c.ind == 2, ] %*% beta2 #linear predictor for competing risk

# Conditional on cause indicators, we simulate the model.
u1 <- runif(length(eta1))
t1 <- -log(1 - (1 - (1 - u1 * (1 - (1 - p)^exp(eta1)))^(1 / exp(eta1))) / p)
t2 <- rexp(length(eta2), rate = exp(eta2))
ci <- runif(nobs, min = u.min, max = u.max) # simulate censoring times

ftime[c.ind == 1] <- t1
ftime[c.ind == 2] <- t2
ftime <- pmin(ftime, ci) # X = min(T, C)
fstatus <- ifelse(ftime == ci, 0, 1) # 0 if censored, 1 if event
fstatus <- fstatus * c.ind  # 1 if cause 1, 2 if cause 2
...
...
...
```

Bibliography

H. Akaike. A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19(6):716–723, 1974. [p49]

P. Breheny and J. Huang. Coordinate descent algorithms for nonconvex penalized regression, with applications to biological feature selection. *The Annals of Applied Statistics*, 5(1):232, 2011. [p43]

N. Breslow. Covariance analysis of censored survival data. *Biometrics*, 30(1):89–99, 1974. doi: 10.2307/2529620. [p45]

R. Calaway, S. Weston, and D. Tenenbaum. **doParallel**: *Foreach Parallel Adaptor for the 'parallel' Package*, 2019. URL `https://CRAN.R-project.org/package=doParallel`. R package version 1.0.14. [p51]

D. R. Cox. Regression models and life-tables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 34(2):187–220, 1972. doi: 10.1007/978-1-4612-4380-9_37. [p43]

P. Craven and G. Wahba. Smoothing noisy data with spline functions. *Numerische Mathematik*, 31(4): 377–403, Dec 1978. ISSN 0945-3245. doi: 10.1007/BF01404567. [p45]

B. Efron. Bootstrap methods: Another look at the jackknife. *The Annals of Statistics*, 7(1):1–26, 1979. doi: 10.1214/aos/1176344552. [p48]

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. doi: 10.1198/016214501753382273. [p45]

Y. Fan and C. Y. Tang. Tuning parameter selection in high dimensional penalized likelihood. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 75(3):531–552, 2013. doi: 10.1111/rssb.12001. [p45]

J. P. Fine and R. J. Gray. A proportional hazards model for the subdistribution of a competing risk. *Journal of the American Statistical Association*, 94(446):496–509, 1999. doi: 10.1080/01621459.1999.10474144. [p43, 44, 45, 46, 51]

Z. Fu. **crrp**: *Penalized Variable Selection in Competing Risks Regression*, 2016. URL https://CRAN.R-project.org/package=crrp. R package version 1.0. [p45]

Z. Fu, C. R. Parikh, and B. Zhou. Penalized variable selection in competing risks regression. *Lifetime Data Analysis*, 23(3):353–376, 2017. doi: 10.1007/s10985-016-9362-3. [p43, 45, 47, 56]

T. A. Gerds, T. H. Scheike, and P. K. Andersen. Absolute risk regression for competing risks: interpretation, link functions, and prediction. *Statistics in Medicine*, 31(29):3921–3930, 2012. doi: 10.1002/sim.5459. [p43]

R. J. Gray. A class of k-sample tests for comparing the cumulative incidence of a competing risk. *The Annals of Statistics*, 16(3):1141–1154, 1988. doi: 10.1214/aos/1176350951. [p44]

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. [p52]

R. Ihaka and R. Gentleman. R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, 5(3):299–314, 1996. URL https://doi.org/10.1080/10618600.1996.10474713. [p43]

E. S. Kawaguchi, J. I. Shen, M. A. Suchard, and G. Li. Scalable algorithms for large competing risks data. *Journal of Computational and Graphical Statistics, accepted pending a minor revision*, 2020. [p43, 44, 46, 57]

D. Kuk and R. Varadhan. Model selection in competing risks regression. *Statistics in Medicine*, 32(18): 3077–3088, 2013. doi: 10.1002/sim.5762. [p43]

S. Mittal, D. Madigan, R. S. Burd, and M. A. Suchard. High-dimensional, massive sample-size cox proportional hazards regression for survival analysis. *Biostatistics*, 15(2):207–221, 2014. doi: 10.1093/biostatistics/kxt043. [p43, 57]

A. Ni and J. Cai. Tuning parameter selection in cox proportional hazards model with a diverging number of parameters. *Scandinavian Journal of Statistics*, 45(3):557–570, 2018. doi: 10.1111/sjos.12313. [p45]

J. M. Robins and A. Rotnitzky. Recovery of information and adjustment for dependent censoring using surrogate markers. In *AIDS epidemiology*, pages 297–331. Springer, 1992. doi: 10.1007/978-1-4757-1229-2_14. [p44]

T. H. Scheike and M.-J. Zhang. Analyzing competing risk data using the r timereg package. *Journal of Statistical Software*, 38(2), 2011. doi: 10.18637/jss.v038.i02. [p43]

M. J. Schuemie, P. B. Ryan, G. Hripcsak, D. Madigan, and M. A. Suchard. Improving reproducibility by using high-throughput observational studies with empirical calibration. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 376(2128):20170356, 2018. doi: 10.1098/rsta.2017.0356. [p57]

G. Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6(2):461–464, 1978. doi: 10.1214/aos/1176344136. [p45]

N. Simon, J. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for cox's proportional hazards model via coordinate descent. *Journal of Statistical Software*, 39(5):1, 2011. [p43]

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: 10.1.1.35.7574. [p45]

H. Wang, R. Li, and C.-L. Tsai. Tuning parameter selectors for the smoothly clipped absolute deviation method. *Biometrika*, 94(3):553–568, 2007. doi: 10.1093/biomet/asm053. [p45]

T. Wang and L. Zhu. Consistent tuning parameter selection in high dimensional sparse linear regression. *Journal of Multivariate Analysis*, 102(7):1141–1151, 2011. doi: 10.1016/j.jmva.2011.03.007. [p45]

C.-H. Zhang. Nearly unbiased variable selection under minimax concave penalty. *The Annals of Statistics*, 38(2):894–942, 2010. doi: 10.1214/09-aos729. [p45]

Y. Zhang, R. Li, and C.-L. Tsai. Regularization parameter selections via generalized information criterion. *Journal of the American Statistical Association*, 105(489):312–323, 2010. doi: 10.1198/jasa.2009.tm08013. [p45]

B. Zhou, A. Latouche, V. Rocha, and J. Fine. Competing risks regression for stratified data. *Biometrics*, 67(2):661–670, 2011. doi: 10.1111/j.1541-0420.2010.01493.x. [p43]

B. Zhou, J. Fine, A. Latouche, and M. Labopin. Competing risks regression for clustered data. *Biostatistics*, 13(3):371–383, 2012. doi: 10.1093/biostatistics/kxr032. [p43]

H. Zou. The adaptive lasso and its oracle properties. *Journal of the American Statistical Association*, 101 (476):1418–1429, 2006. doi: 10.1198/016214506000000735. [p45]

Eric S. Kawaguchi
University of Southern California
Department of Preventive Medicine
2001 N. Soto St. Los Angeles, CA 90032, USA
eric.kawaguchi@med.usc.edu

Jenny I. Shen
The Lundquist Institute at Harbor-UCLA Medical Center
Division of Nephrology and Hypertension
1124 W. Carson St.
Torrance, CA 90502, USA
jshen@lundquist.org

Gang Li
University of California, Los Angeles
Departments of Biostatistics and Computational Medicine
Los Angeles, CA 90095, USA
vli@ucla.edu

Marc A. Suchard
University of California, Los Angeles
Departments of Biostatistics, Computational Medicine, and Human Genetics
Los Angeles, CA 90095, USA
msuchard@ucla.edu

ordinalClust: An R Package to Analyze Ordinal Data

by Margot Selosse, Julien Jacques and Christophe Biernacki

Abstract Ordinal data are used in many domains, especially when measurements are collected from people through observations, tests, or questionnaires. **ordinalClust** is an innovative R package dedicated to ordinal data that provides tools for modeling, clustering, co-clustering and classifying such data. Ordinal data are modeled using the BOS distribution, which is a model with two meaningful parameters referred to as "position" and "precision". The former indicates the mode of the distribution and the latter describes how scattered the data are around the mode: the user is able to easily interpret the distribution of their data when given these two parameters. The package is based on the co-clustering framework (when rows and columns are simultaneously clustered). The co-clustering approach uses the Latent Block Model (LBM) and the SEM-Gibbs algorithm for parameter inference. On the other hand, the clustering and the classification methods follow on from simplified versions of the SEM-Gibbs algorithm. For the classification process, two approaches are proposed. In the first one, the BOS parameters are estimated from the training dataset in the conventional way. In the second approach, parsimony is introduced by estimating the parameters and column-clusters from the training dataset. We empirically show that this approach can yield better results. For the clustering and co-clustering processes, the ICL-BIC criterion is used for model selection purposes. An overview of these methods is given, and the way to use them with the **ordinalClust** package is described using real datasets. The latest stable package version is available on the Comprehensive R Archive Network (CRAN).

Introduction

Ordinal data is a specific kind of categorical data occurring when the levels are ordered (Agresti, 2012). Some common contexts for the collection of ordinal data include satisfaction surveys, aptitude and personality tests and psychological questionnaires. In the present work, an ordinal variable is represented by x and it is considered to have m levels that are written $(1, ..., m)$.

Thus far, ordinal data have received more attention from a supervised point of view. For example: a marketing firm investigating which factors influence the size of a soda (small, medium, large or extra large) that people order at a fast-food chain. These factors may include which type of sandwich is ordered (burger or chicken), whether or not fries are also ordered, and the consumer's age. In this case, an observation consists in factors of different types and the variable to predict is an ordinal variable. Several software can analyze ordinal data in a regression framework. The cumulative link model (CLM) assumes that:

$$\text{logit}\left(p\left(x \leq \mu\right)\right) = \log \frac{p(x \leq \mu)}{1-p(x \leq \mu)} = \beta_0\left(\mu\right) + \boldsymbol{\beta}^t \boldsymbol{t},$$

where x is the ordinal variable, μ is one of its levels, $\boldsymbol{t}$ are the covariates, and $\beta_0(1) \leq \beta_0(2) \leq \ldots \leq \beta_0(m)$. In the absence of covariates, it is equivalent to a multinomial model. CLMs are a powerful model class for ordinal data since observations are handled as categorical, their ordered nature is exploited and the regression framework enables interpretable analyses. In R, several packages implement this kind of models. The package **MASS** (Venables and Ripley, 2002) implements the CLM with standard link functions, while **VGAM** (Yee, 2010), **rms** (Jr, 2019), **brms** (Bürkner, 2017) and **ordinal** (Christensen, 2015) bring additional functions and features. Other contributions implement algorithms for ordinal data classification. For instance, the **ordinalForest** package (Hornung, 2019a,b) uses ordinal forests and **monmlp** (Cannon, 2017) uses neural networks, both to predict ordinal response variables. Finally, the **ocapis** package (Heredia-Gómez et al., 2019) implements several methods (such as CMLs, Support Machine, Weighted k-Nearest-Neighbor) to classify and preprocess ordinal data.

However, the focus of these techniques differs from ours in two ways. Firstly, they work in a supervised framework (classification). Secondly, they work with datasets for which the variables to predict are ordinal responses: the other variables are of various types. Our goal is to provide a tool for unsupervised and supervised tasks, and for datasets comprised only of ordinal variables only (in the classification context, the response is categorical). From an unsupervised point a view, the Latent Gold Software J. Vermunt (2005) is – to our knowledge – the only software that uses the CMLs to cluster the data. Nevertheless, the implementation of this method is known to be computationally expensive. In addition, it is not provided through a user-friendly R package.

Other contributions have defined clustering algorithms with ordinal variables. In McParland and Gormley (2013), the authors propose a model-based technique by considering the probability distribution of ordinal data as a discretization of an underlying continuous variable. This approach is implemented in the **clustMD** package (McParland and Gormley, 2017), which is generally more for heterogeneous data. In Ranalli and Rocci (2016), the categorical variables are seen as a discretization of an underlying finite mixture of Gaussians. In other works, the authors use the multinomial distribution to model the data. For instance in the case of Giordan and Diana (2011), the multinomial distribution and a cluster tree are used, whereas Jollois and Nadif (2009) apply a constrained multinomial distribution. However, these contributions do not provide a way to co-cluster and classify ordinal data. Furthermore, they are not always available as an R package (except in the case of McParland and Gormley (2013)). More recently, Corneli et al. (2020) proposed a method to co-cluster ordinal data modeled via latent Gaussian random variables. Their package **ordinalLBM** (Corneli et al., 2019) is available on CRAN.

Finally, the CUB (Combination of a discrete Uniform and a shifted Binomial random variable) model (D'Elia and Piccolo, 2005) is widely used to analyze ordinal datasets. For instance, Corduas (2008) proposes a clustering algorithm based on a mixture of CUB models. In the CUB model, an answer is interpreted as the result of a cognitive process where the decision is intrinsically continuous but is expressed on a discrete scale of m levels. This approach interprets the choice of the respondent as a weighted combination of two components. The first component reflects a personal feeling and is expressed by a shifted binomial random variable. The second component reflects an intrinsic uncertainty and is expressed by a uniform random variable. Many extensions for the CUB model have been defined and the **CUB** package (Maria Iannario, 2018) implements the associated statistical methods.

More recently, Biernacki and Jacques (2016) proposed the so-called Binary Ordinal Search model, referred to as the "BOS" model. It is a probability distribution specific to ordinal data that is parameterized with meaningful parameters (μ, π), linked to a position and precision role, respectively. This work also describes how the BOS distribution can be used to perform clustering on multivariate ordinal data. Jacques and Biernacki (2018) then employed this distribution coupled to the Latent Block Model (Govaert and Nadif, 2003) in order to carry out a co-clustering on ordinal data. The co-clustering task consists of simultaneously clustering the rows and the columns of the data matrix. It is a useful way of clustering the data while introducing parsimony, and providing more interpretable partitions. The authors in Jacques and Biernacki (2018) showed that their algorithm can easily deal with missing values. However, this model could not take ordinal data with different numbers of levels into account. Selosse et al. (2019) used an extension of the Latent Block Model to overcome this issue. These works have proved their proficiency and also provide efficient techniques to perform clustering and co-clustering of ordinal data. The purpose of the **ordinalClust** package is to offer a complete tool for analyzing ordinal data by implementing these methods. Furthermore, it presents a novel approach for classifying ordinal datasets with categorical responses. The present document gives an overview of the underlying methods and illustrates usage of **ordinalClust** through concrete examples. The paper is organized as follows. In the section "Statistical methods", the notation and models are described. The section "Application to the patients quality of life analysis in oncology" presents the functions of **ordinalClust** and details a use case for psychological survey datasets. The section "Conclusion" discusses the limits of **ordinalClust** and future work for the package.

Statistical methods

Data Notation

A dataset of ordinal data will be written as $\boldsymbol{x} = \left(x_{ij}\right)_{i,j}$, with $1 \leq i \leq N$ and $1 \leq j \leq J$, N and J denoting the number of individuals and the number of variables, respectively. Furthermore, a dataset can contain missing data. While dealing with this aspect, the dataset will be expressed by $\boldsymbol{x} = (\check{\boldsymbol{x}}, \hat{\boldsymbol{x}})$, with $\check{\boldsymbol{x}}$ being the observed data and $\hat{\boldsymbol{x}}$ being the missing data. Consequently an element of $\boldsymbol{x}$ will be annotated as follows: $\check{x}_{ij}$, whether x_{ij} is observed, $\hat{x}_{ij}$ otherwise.

The BOS model

The BOS model (Biernacki and Jacques, 2016) is a probability distribution for ordinal data parameterized by a position parameter $\mu \in \{1, ..., m\}$ and a precision parameter $\pi \in [0, 1]$. It was built on the assumption that an ordinal variable is the result of a stochastic binary search algorithm within the ordered table $(1, ..., m)$. This distribution rises from a uniform distribution when $\pi = 0$ to a more peaked distribution around the mode μ when π grows, and reaches a Dirac distribution at the mode μ

when $\pi = 1$. Figure 1 illustrates the shape of the BOS distribution with different values of μ and π. In Biernacki and Jacques (2016) it is shown that the BOS distribution is a polynomial function of π with degree $m - 1$ whose coefficients depend on the position parameter μ. For a univariate ordinal variable, the path in a stochastic binary search can be seen as a latent variable. Therefore, an efficient way to perform the maximum likelihood estimation is through the EM algorithm (Dempster et al., 1977).

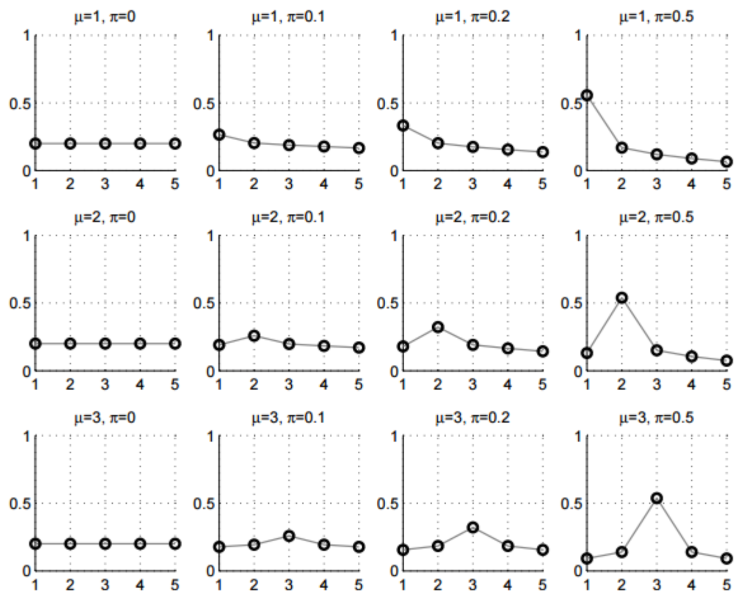


Figure 1: BOS distribution $p(x;\mu,\pi)$: shapes for $m = 5$ and for different values of μ and π.

The co-clustering model

Notation With this being in a co-clustering context, it is assumed that there are G row-clusters and H column-clusters inherent to the $\boldsymbol{x}$ matrix. It is therefore useful to introduce g (or h) which represents the g^{th} (or h^{th}) row-cluster (or column-cluster), with $1 \le g \le G$ (or $1 \le h \le H$). In addition, the sums and the products related to rows, columns, row-clusters and column-clusters will be subscripted using the letters i, j, g and h respectively. Therefore, the sums and products will be written as $\sum_i$, $\sum_j$, $\sum_g$ and $\sum_h$, and $\prod_i$, $\prod_j$, $\prod_g$ and $\prod_h$.

Latent Block Model Let us consider the data matrix $\boldsymbol{x} = \left(x_{ij}\right)_{i,j}$. It is assumed that there are G row-clusters and H column-clusters that correspond to a partition $\boldsymbol{v} = \left(v_{ig}\right)_{i,g}$ and a partition $\boldsymbol{w} = \left(w_{jh}\right)_{j,h}$ respectively, with $1 \le g \le G$ and $1 \le h \le H$. We have noted that $v_{ig} = 1$ if i belongs to cluster g, whereas $v_{ig} = 0$ otherwise, and $w_{jh} = 1$ when j belongs to cluster h, but $w_{jh} = 0$ otherwise. Each element x_{ij} is considered to be generated under a parameterized probability density function $p\left(x_{ij};\alpha_{gh}\right)$. Here, g denotes the cluster of row i, and h denotes the cluster of column j, while α_{gh} represents the parameters of a probability density function of block (g,h), a block being the crossing of both a row-cluster and a column-cluster. Figure 2 is an example of co-clustering performed on an ordinal data matrix.

The univariate random variables x_{ij} are assumed to be conditionally independent given the row and column partitions $\boldsymbol{v}$ and $\boldsymbol{w}$. Therefore, the conditional probability density function of $\boldsymbol{x}$ given $\boldsymbol{v}$ and $\boldsymbol{w}$ can be written as:

$$p\left(\boldsymbol{x}|\boldsymbol{v},\boldsymbol{w};\boldsymbol{\alpha}\right) = \prod_{i,j,g,h} p\left(x_{ij};\alpha_{gh}\right)^{v_{ig}w_{jh}},$$

where $\boldsymbol{\alpha} = \left(\alpha_{gh}\right)_{g,h}$ is the distribution's parameters of block (g,h). Any univariate distribution can be used with respect to the kind of data (e.g: Gaussian, Bernoulli, Poisson...). In the **ordinalClust**

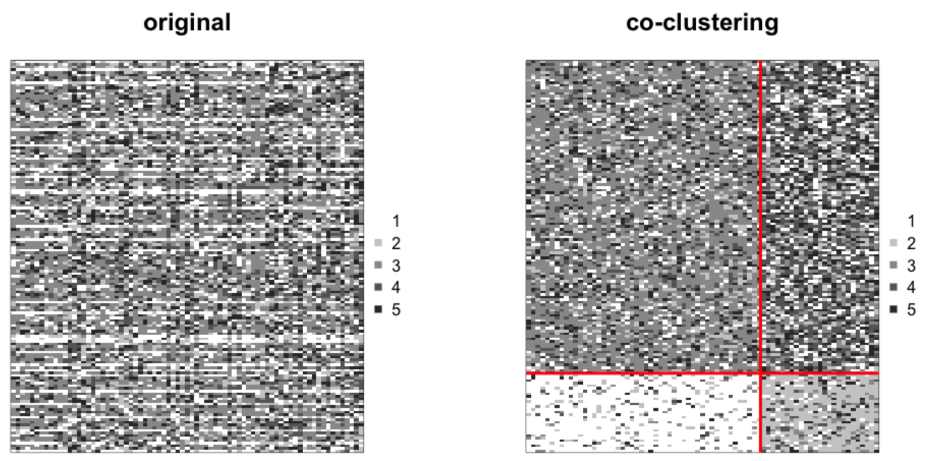


Figure 2: Left: original dataset made of ordinal data with $m = 5$. Right: a co-clustering is performed with $G = H = 2$, the rows and columns are sorted by row-clusters and column clusters, which emphasizes a structure in the dataset.

package, the BOS distribution is employed, thus $\alpha_{gh} = \left(\mu_{gh}, \pi_{gh}\right)$. For convenience, the label of row i is also denoted by $\boldsymbol{v}_i = (v_{i1}, ..., v_{iG}) \in \{0,1\}^G$. Similarly, the label of column j is denoted by $\boldsymbol{w}_j = \left(w_{j1}, ..., w_{iH}\right) \in \{0,1\}^H$. These latent variables $\boldsymbol{v}$ and $\boldsymbol{w}$ are assumed to be independent so $p\left(\boldsymbol{v}, \boldsymbol{w}; \boldsymbol{\gamma}, \boldsymbol{\rho}\right) = p\left(\boldsymbol{v}; \boldsymbol{\gamma}\right) p\left(\boldsymbol{w}; \boldsymbol{\rho}\right)$ with:

$$p\left(\boldsymbol{v}; \boldsymbol{\gamma}\right) = \prod_{i,g} \gamma_g^{v_{ig}} \text{ and } p\left(\boldsymbol{w}; \boldsymbol{\rho}\right) = \prod_{j,h} \rho_h^{w_{jh}},$$

with the knowledge that $\gamma_g = p\left(v_{ig} = 1\right)$ with $g \in \{1, ..., G\}$ and $\rho_h = p\left(w_{jh} = 1\right)$ with $h \in \{1, ..., H\}$. This implies that, for all i, the distribution of $\boldsymbol{v}_i$ is the multinomial distribution $\mathcal{M}\left(\gamma_1, ..., \gamma_G\right)$ and does not depend on i. In a similar way, for all j, the distribution of $\boldsymbol{w}_j$ is the multinomial distribution $\mathcal{M}\left(\rho_1, ..., \rho_H\right)$ and does not depend on j. From these considerations, the parameters of the latent block model are defined as $\boldsymbol{\theta} = (\boldsymbol{\gamma}, \boldsymbol{\rho}, \boldsymbol{\mu}, \boldsymbol{\pi})$, with $\boldsymbol{\gamma} = (\gamma_1, ..., \gamma_G)$ and $\boldsymbol{\rho} = (\rho_1, ..., \rho_H)$ as the mixing proportions of the rows and columns; $\boldsymbol{\mu} = \left(\mu_{gh}\right)_{g,h}$ and $\boldsymbol{\pi} = \left(\pi_{gh}\right)_{g,h}$ are the distribution parameters of the blocks. Therefore, if V and W are the sets of all possible labels $\boldsymbol{v}$ and $\boldsymbol{w}$, the probability density function $p\left(\boldsymbol{x}; \boldsymbol{\theta}\right)$ of $\boldsymbol{x}$ can be written as:

$$p\left(\boldsymbol{x}; \boldsymbol{\theta}\right) = \sum_{(\boldsymbol{v}, \boldsymbol{w}) \in V \times W} \prod_{i,g} \gamma_g^{v_{ig}} \prod_{j,h} \rho_h^{w_{jh}} \prod_{i,j,g,h} p\left(x_{ij}; \alpha_{gh}\right)^{v_{ig} w_{jh}}. \quad (1)$$

Model Inference In the co-clustering context, tha im of the inference is to maximize the observed log-likelihood $l\left(\boldsymbol{\theta}; \check{\boldsymbol{x}}\right) = \sum_{\hat{\boldsymbol{x}}} \log p\left(\boldsymbol{x}; \boldsymbol{\theta}\right)$. The EM-algorithm (Dempster et al., 1977) is a very well known technique for maximizing parameters with latent variables. However, with respect to the co-clustering case, it is not computationally tractable. Indeed, this method requires computation of the expectation of the complete data log-likelihood. Nevertheless, this expression contains the probability $p\left(v_{ig} = 1, w_{jh} = 1 | \boldsymbol{x}, \boldsymbol{\theta}\right)$, which needs to take into account all the possible values for $v_{i'}$ and $\boldsymbol{w}_{j'}$ with $i' \neq i$ and $j' \neq j$. The E-step would require calculation of $G^N \times H^J$ terms for each value of the data matrix. Using the values from the section "Application to the patients quality of life analysis in oncology", i.e., $G = 3$, $H = 3$, $N = 117$ and $J = 28$, it would result in computation of $3^{117} \times 3^{28} \approx 1 \times 10^{69}$ terms. There are different alternatives to the EM algorithm, such as the variational EM algorithm, the SEM-Gibbs algorithm or other algorithms linked to a Bayesian inference. The SEM-Gibbs version is used because it is known to avoid spurious solutions (Keribin et al., 2010). Furthermore, it easily handles missing values $\hat{\boldsymbol{x}}$ in $\boldsymbol{x}$, which is an important advantage, particularly with real datasets. The SEM-algorithm is made of two iteratively repeated steps that are detailed in

Algorithm 1.

```
Data: $\boldsymbol{x}$, $G$, $H$
Result: A sequence $(\boldsymbol{v}, \boldsymbol{w}, \boldsymbol{\theta}, \hat{\boldsymbol{x}})^{(q)}$ for $q \in \{1, ..., nbSEM\}$
Initialization of $\hat{\boldsymbol{x}}$, $\boldsymbol{v}$, $\boldsymbol{w}$ and $\boldsymbol{\theta}$ by $\hat{\boldsymbol{x}}^{(0)}$, $\boldsymbol{v}^{(0)}$, $\boldsymbol{w}^{(0)}$ and $\boldsymbol{\theta}^{(0)}$, respectively;
for $q$ in 1:nbSEM do
    1. SE-step.
        1.1 Sample the row partitions for all $1 \leq i \leq N, 1 \leq g \leq G$:
            $p\left(v_{ig} = 1 | \boldsymbol{x}^{(q-1)}, \boldsymbol{w}^{(q-1)}; \boldsymbol{\theta}^{(q-1)}\right) \propto \gamma_g^{(q-1)} \prod_{j,h} p\left(x_{ij}; \mu_{gh}^{(q-1)}, \pi_{gh}^{(q-1)}\right)^{w_{jh}^{(q-1)}}.$
        1.2 Sample the column partitions for all $1 \leq j \leq J, 1 \leq h \leq H$:
            $p\left(w_{jh} = 1 | \boldsymbol{x}, \boldsymbol{v}^{(q)}; \boldsymbol{\theta}^{(q-1)}\right) \propto \rho_h^{(q-1)} \prod_{i,g} p\left(x_{ij}; \mu_{gh}^{(q-1)}, \pi_{gh}^{(q-1)}\right)^{v_{ig}^{(q)}}.$
        1.3 Generate the missing data:
            $p\left(\hat{x}_{ij}^{(q)} | \check{\boldsymbol{x}}, \boldsymbol{v}^{(q)}, \boldsymbol{w}^{(q)}; \boldsymbol{\theta}^{(q-1)}\right) = \prod_{g,h} p\left(\hat{x}_{ij}; {\mu_{gh}}^{(q-1)}, {\pi_{gh}}^{(q-1)}\right)^{v_{ig}^{(q)} {w_{gh}}^{(q)}}.$
    2. M-step.
        2.1 Update the mixing proportions:
            $\rho_h^{(q)} = \frac{1}{J} \sum_j w_{jh}^{(q)}$ and $\gamma_h^{(q)} = \frac{1}{N} \sum_i v_{ig}^{(q)}.$
        2.2 Update the parameters $\boldsymbol{\mu}^{(q)}$ and $\boldsymbol{\pi}^{(q)}$ (see Biernacki and Jacques (2016)).
end
```

Algorithm 1: SEM-Gibbs for co-clustering on ordinal data.

Initializations The **ordinalClust** package provides three modes for value initialization. It is set through the argument `init`, which can take values `'random'`, `'kmeans'` or `'randomBurnin'`. The first value randomly initializes $\boldsymbol{v}^{(0)}$ and $\boldsymbol{w}^{(0)}$ with the multinomial distribution $\mathcal{M}(1/G, \ldots, 1/G)$ and $\mathcal{M}(1/H, \ldots, 1/H)$, respectively. The second argument (by default) value consists of performing a Kmeans algorithm (Hartigan and Wong, 1979) on the rows and on the columns.

The third one, `'randomBurnin'` is a bit more complex and requires additional arguments for the algorithm. It aims at avoiding a degeneracy of the algorithm that leads to empty clusters, knowing that the degeneracy event arises more often at the early stage of the algorithm (thus during the burn-in period. This starts with a first random initialization. However, for the first `nbSEMburn` iterations (`nbSEMburn` < `nbSEM`), whenever a row-cluster gets empty, a percentage `percentRandomB` of the row partitions are resampled from the multinomial distribution $\mathcal{M}(1/G, \ldots, 1/G)$. Similarly when a column-cluster gets empty, a percentage of the column partitions are resampled from the multinomial distribution $\mathcal{M}(1/H, \ldots, 1/H)$.

Estimation of model parameters and partitions The first iterations of the SEM-Gibbs are called the burn-in period, which means that the parameters are not yet stable. Consequently, only the iterations that occur after this burn-in period are taken into account and are referred to as the "sampling distribution" hereafter. While the final estimation of the position parameters $\hat{\boldsymbol{\mu}}$ are the mode of the sampling distributions, the final estimations of the continuous parameters $(\hat{\boldsymbol{\pi}}, \hat{\boldsymbol{\gamma}}, \hat{\boldsymbol{\rho}})$ are the mean of the sample distribution. This leads to a final estimation of $\boldsymbol{\theta}$ that is called $\hat{\boldsymbol{\theta}}$. Then, a sample of $(\hat{\boldsymbol{x}}, \boldsymbol{v}, \boldsymbol{w})$ is generated through several SE-steps (step **1.** from Algorithm 1) with $\boldsymbol{\theta}$ fixed to $\hat{\boldsymbol{\theta}}$. The final partitions $(\hat{\boldsymbol{v}}, \hat{\boldsymbol{w}})$ and the missing observations $\hat{\boldsymbol{x}}$ are estimated by the mode of their sample distribution.

Model Selection To determine how many row-clusters and how many column-clusters are necessary, an adaptation of the ICL criterion (Biernacki et al., 2000), called ICL-BIC, is proposed in Jacques and Biernacki (2018). In practice, the algorithm must be executed with all the (G, H) to test, and the highest ICL-BIC is retained.

The clustering model

The clustering model described in this section is a particular case of the co-clustering model, in which each feature is in its own cluster ($H = J$). Consequently, $\boldsymbol{w}$ is no longer a latent variable since each feature represents a cluster of size 1. Let us define a multivariate ordinal variable $\boldsymbol{x}_i = \left(x_{ij}\right)_j$ with $1 \leq j \leq J$. Conditionally to cluster g, the distribution of $\boldsymbol{x}_i$ is assumed to be:

$$p\left(\boldsymbol{x}_i | v_{ig} = 1; \boldsymbol{\mu}_g, \boldsymbol{\pi}_g\right) = \prod_j p\left(x_{ij}; \mu_{gj}, \pi_{gj}\right),$$

where $\boldsymbol{\mu}_g = \left(\mu_{gj}\right)_j$ and $\boldsymbol{\pi}_g = \left(\pi_{gj}\right)_j$ with $1 \leq j \leq J$. This conditional independence hypothesis assumes that conditional to belonging to row-cluster g, the J ordinal responses of an individual are independently drawn from J univariate BOS models of parameters $\left(\mu_{gj}, \pi_{gj}\right)_{j \in \{1,...,J\}}$. Furthermore, as in the co-clustering case, the distribution of $\boldsymbol{v}_i$ is assumed to be a multinomial distribution $\mathcal{M}\left(\gamma_1, ..., \gamma_G\right)$ and not dependent on i. In this configuration, the parameters of the clustering model are defined as $\boldsymbol{\theta} = (\boldsymbol{\gamma}, \boldsymbol{\alpha})$, with $\alpha_{gj} = \left(\mu_{gj}, \pi_{gj}\right)$ being the position and precision BOS parameters of the row-cluster g and ordinal variable j. Consequently, with a matrix $\boldsymbol{x} = \left(x_{ij}\right)_{i,j}$ of ordinal data, the probability density function $p\left(\boldsymbol{x}; \boldsymbol{\theta}\right)$ of $\boldsymbol{x}$ is written as:

$$p\left(\boldsymbol{x}; \boldsymbol{\theta}\right) = \sum_{v \in V} \prod_{i,g} \gamma_g^{v_{ig}} \prod_{i,j,g} p\left(x_{ij}; \mu_{gj}, \pi_{gj}\right)^{v_{ig}}. \quad (2)$$

To infer the parameters of this model, the SEM-Gibbs Algorithm 1 is used with the part in **1.2** removed from the SE-step. The part in **1.3** relating to missing value imputation also remains. It is noted here that clustering can also be achieved by using the co-clustering model in section "The co-clustering model", and by considering the resulting $\boldsymbol{v}$ partition as the outcome. As a matter of fact, in this case, the co-clustering is a parsimonious version of the clustering procedure.

The classification model

By considering a classification task with a categorical variable to predict from ordinal data, the configuration encountered is a particular case where $\boldsymbol{v}$ is known for all $i \in \{1, ..., N\}$ and for all $g \in \{1, ..., G\}$. In **ordinalClust**, two classification models are provided.

Multivariate BOS model This first model is similar to the clustering model: each variable represents a column-cluster of size 1, thus $\boldsymbol{w}$ is not a latent variable. This model assumes that, conditional on the class of the observations, the J variables are independent. Since the row classes are observed, the algorithm only needs to estimate the parameter $\boldsymbol{\theta}$ that maximizes the log-likelihood $l\left(\boldsymbol{\theta}; \check{\boldsymbol{x}}\right)$. The probability density function $p\left(\boldsymbol{x}, \boldsymbol{v}; \boldsymbol{\theta}\right)$ is therefore expressed as below:

$$p\left(\boldsymbol{x}, \boldsymbol{v}; \boldsymbol{\theta}\right) = \prod_{i,g} \gamma_g^{v_{ig}} \prod_{i,j,g} p\left(x_{ij}; \alpha_{gj}\right)^{v_{ig}}. \quad (3)$$

The inference of this model's parameters only requires the M-step of Algorithm 1. However, if there are missing data, the SE-step made of the part in **1.3** only is also required.

Parsimonious BOS model This model is a parsimonious version of the first model. Parsimony is introduced by grouping the features into H clusters (as in the co-clustering model). The main hypothesis is that given the row-cluster partitions and the column-cluster partitions, the realization of x_{ij} is independent from the other ones. In practice the number H of column-clusters is chosen with a training dataset and a validation dataset. Consequently, the probability density function $p\left(\boldsymbol{x}, \boldsymbol{v}; \boldsymbol{\theta}\right)$ is

$$\boldsymbol{x} = \left[\left[\quad \boldsymbol{x}^1 \quad \right] \ldots \left[\quad \boldsymbol{x}^D \quad \right] \right], \text{ with } \boldsymbol{x}^d = \left(x_{ij}^d \right)_{i=1,\ldots,N;\, j=1,\ldots,J_d}.$$

Figure 3: Data set matrix $\boldsymbol{x}$ when the ordinal data has D numbers of levels.

annotated:

$$p\left(\boldsymbol{x}, v; \boldsymbol{\theta}\right) \;=\; \sum_{w \in W} \prod_{i,g} \gamma_g^{v_{ig}} \prod_{j,h} \rho_h^{w_{jh}} \prod_{i,j,g,h} p\left(x_{ij}; \alpha_{gh}\right)^{v_{ig} w_{jh}}. \tag{4}$$

To infer this model's parameters, Algorithm 1 is used with an SE-step only containing the part in **1.2**, and the entire M-step. Again, if there are missing data, the SE-step made of the part in **1.3** is also required.

Handling ordinal data with several numbers of levels

The Latent Block Model as described before cannot take variables with different numbers of levels m into account. Indeed, the distributions of variables with different numbers of levels are not defined on the same support. This implies that it is impossible to gather two variables with different m within a same block.

In Selosse et al. (2019), a constrained Latent Block Model is provided. Although it does not allow ordinal features with different numbers of levels to be gathered in a same column-cluster, it is able to take into account the fact that there are several numbers of levels and to perform a co-clustering on more diverse datasets. The matrix $\boldsymbol{x}$ is considered to contain D different numbers of levels. Its representation is seen as D matrices placed side by side, such that the d^{th} table is a $N \times J_d$ matrix written as $\boldsymbol{x}^d$ and composed of ordinal data with numbers of levels m_d (see Figure 3).

The model relies on the following hypothesis:

$$p\left(\boldsymbol{x}^1, \ldots \boldsymbol{x}^D | \boldsymbol{v}, \boldsymbol{w}^1, \ldots, \boldsymbol{w}^D\right) = p\left(\boldsymbol{x}^1 | \boldsymbol{v}, \boldsymbol{w}^1\right) \times \ldots \times p\left(\boldsymbol{x}^D | \boldsymbol{v}, \boldsymbol{w}^D\right),$$

with $\boldsymbol{w}^d$ the column partition of $\boldsymbol{x}^d$. This means that there is independence between the D blocks, knowing their row and column partitions: the realization of the univariate random variable x_{ij}^d will not depend on the column partitions of the other blocks than d.

In this case, the SEM-Gibbs algorithm is slightly changed: in the SE-step, a sampling step is appended for every additional $\boldsymbol{x}^d$. For further details on this adapted SEM-Gibbs algorithm, see Selosse et al. (2019).

Application to the patients quality of life analysis in oncology

This section explains how to use the implementation of the methods described before through the **ordinalClust** package. Users should be aware that the code provided was run with R 3.5.3, and that the results could be different with another version. If users wish to use a version of R $\geq$ 3.6.0 and reproduce the same results as in the paper, they should run the command `RNGkind(sample.kind='Rounding')` before running the code.

Data sets

The datasets included were part of the **QoLR** package (Anota et al., 2017). They contain responses to the well known "EORTC QLQ-C30" (European Organization for Research and Treatment of Cancer (EORTC) Quality of Life Questionnaire (QLQ-C30)), provided to patients affected by breast cancer. Furthermore, for all questions, the most positive answer is given by a level "1". For example, for question: *"During the past week, did you feel irritable?"* with possible responses: *"Not at all." "A little." "Quite a bit." "Very much."*, the following level numbers are assigned to the replies: 1 *"Not at all."*, 2 *"A little."*, 3 *"Quite a bit."*, 4 *"Very much."*, because it is perceived as more negative to have felt irritable. Two datasets are available:

- `dataqol` is a dataframe with 117 lines such that each line represents a patient and the columns contain information about the patient:
  - `Id`: patient Id,
  - `q1-q28`: responses to 28 questions with the number of levels equals to 4,
  - `q29-q30`: responses to 2 questions with the number of levels equals to 7.
- `dataqol.classif` is a dataframe with 40 lines such that a line represents a patient, and the columns contain information about the patient:
  - `Id`: patient Id,
  - `q1-q28`: responses to 28 questions with the number of levels equals to 4,
  - `q29-q30`: responses to 2 questions with the number of levels equals to 7,
  - `death`: if the patient passed away (2) or not (1).

The datasets contain missing values, coded as `NA`: in `dataqol`, 1.1% are missing values and 3.6% in `dataqol.classif`. To load the package and its datasets, the following commands must be executed:

```
library(ordinalClust)
data("dataqol")
data("dataqol.classif")
```

Then, a seed is set so that users can obtain results identical to this document:

```
set.seed(1)
```

Users must define how many SEM-Gibbs iterations (`nbSEM`) and how many burn-in iterations (`nbSEMburn`) are needed for Algorithm 1. The section "Setting the SEMburn and nbSEMburn arguments" provides an empirical way of checking correctness of these values. Moreover, the `nbindmini` argument must be defined: it indicates the minimum number of elements that must be present in a block. Finally, the `init` argument indicates how to initialize the algorithm. It can be set to `"kmeans"`, `"random"` or `"randomBurnin"`.

```
nbSEM <- 150
nbSEMburn <- 100
nbindmini <- 1
init <- "randomBurnin"
percentRandomB <- c(50, 50)
```

Here, `percentRandom` is a vector because it defines two percentages: the percentage of rows that will be resampled if a row-cluster is emptied, and the percentage of columns that will be resampled if a column-cluster is emptied.

Performing classification

In this section, the `dataqol.classif` dataset is used. The aim is to predict the `death` variable from the ordinal data that corresponds to the patients answers. The following commands show how to setup the classification configuration. First, the x ordinal data matrix (the responses to the questionnaires) is defined, as well as the v vector, which is the variable `death` to predict.

```
x <- as.matrix(dataqol.classif[,2:29])
v <- dataqol.classif$death
```

ordinalClust provides two classification models. The first model (chosen by the option `kc=0`) is a multivariate BOS model with the assumption that, conditional on the class of the observations, the features are independent as in Equation 3. The second model introduces parsimony by grouping the features into clusters and assuming that the features of a cluster have a common distribution, as in Equation 4. This latter is a novel approach for classification. The number H of clusters of features is defined with the argument `kc = H`. H is selected using a training dataset and a validation dataset:

```
# sampling datasets for training and to predict
nb.sample <- ceiling(nrow(x)*7/10)
sample.train <- sample(1:nrow(x), nb.sample, replace=FALSE)

x.train <- x[sample.train,]
x.validation <- x[-sample.train,]

v.train <- v[sample.train]
v.validation <- v[-sample.train]
```

We also indicate how many classes there are, and how many levels the ordinal data have:

```
# classes
kr <- 2
# levels
m <- 4
```

The training can be performed using the function bosclassif. In the code below, several kc parameters are tested. When kc = 0, the multivariate model is used: all variables are considered to be independent. When kc >0, the parsimonious model is used: the variables are grouped into kc groups. To classify new observations, the predict function is used: it takes as arguments the result from bosclassif and the observations to classify. In the following example, we store in the preds matrix the predictions resulting from the classifications performed with different kc.

```
kcol <- c(0, 1, 2, 3, 4)
preds <- matrix(0, nrow = length(kcol), ncol = nrow(x.validation))

for( kc in 1:length(kcol) ){
  classif <- bosclassif(x = x.train, y = v.train, kr = kr, kc = kcol[kc],
               m = m, nbSEM = nbSEM, nbSEMburn = nbSEMburn,
               nbindmini = nbindmini, init = init,
               percentRandomB = percentRandomB)
  new.prediction <- predict(classif, x.validation)
  if(!is.character(new.prediction)){
       preds[kc,] <- new.prediction@zr_topredict
  }
}
```

Then the preds matrix can be formatted to a dataframe:

```
preds <- as.data.frame(preds)
row.names <- paste0("kc = ", kcol)
rownames(preds) <- row.names

preds
v.validation

> preds
     V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11 V12
kc=0  2  1  2  2  2  2  1  1  1   2   1   2
kc=1  2  1  2  1  2  2  1  2  1   1   2   2
kc=2  2  1  2  2  2  2  1  2  1   2   2   2
kc=3  2  1  2  1  2  2  1  2  1   2   1   2
kc=4  1  1  2  1  1  1  1  2  1   2   1   2
> v.validation
 [1] 2 1 1 1 1 1 1 2 1 1 1 2
```

Table 1 shows the sensitivity and specificity for each different kc. The code to get these values is available in the Appendix "Specificity and sensitivity". First of all, the results are globally satisfying since the sensitivities and specificities are quite high. We observe that the parsimonious models (when kc = 1,2,3,4) have better results than the multivariate model (kc = 0). The two parsimonious models kc = 1 and kc = 3 obtain the best results. This illustrates the interest of introducing parsimonious models in a supervised context. However, users should be aware that the dataset is small, and the number of observations used here is too low to draw definitive conclusions.

Performing clustering

Clustering setting. This section uses the dataqol dataset, plotted in Figure 4.

The purpose of clustering is to emphasize information regarding the rows of a data matrix. First, the $\boldsymbol{x}$ ordinal matrix is loaded, which corresponds to the patients' responses:

```
set.seed(1)
x <- as.matrix(dataqol[,2:29])
```

The clustering is obtained using the bosclust function:

Table 1: Sensitivity and specificity for different kc.

	sensitivity	specificity
kc = 0	0.67	0.44
kc = 1	**1.00**	**0.56**
kc = 2	1.00	0.33
kc = 3	**1.00**	**0.56**
kc = 4	0.78	0.67

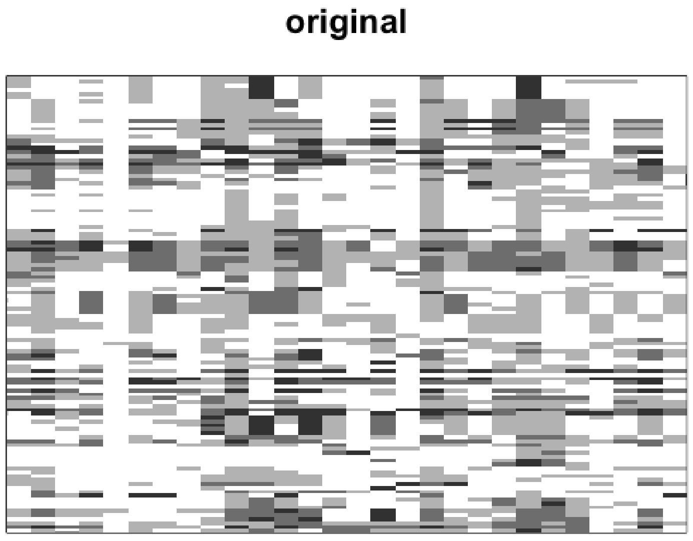


Figure 4: Plot of the dataqol dataset. The rows represent the patients, the columns represent the questions they answered to. A cell is the response of a patient to a question. The more black the cell is, the more negative the answer.

```
clust <- bosclust(x = x, kr = 3, m = 4,
            nbSEM = nbSEM, nbSEMburn = nbSEMburn,
            nbindmini = nbindmini, init = init)
```

The outcome can be plotted using the `plot` function:

```
plot(clust)
```

Figure 5 represents the clustering result. We count the clusters from the bottom to the top. Among the 3 row-clusters, the first one (at the bottom) stands out as the lightest. This means that the patients from this cluster globally chose levels close to 1, which is the most positive answer. In contrast, the third row-cluster (at the top) is darker, which implies the patients from this group answered in a more negative way.

Clusters interpretation. The parameters are obtained with the command `clust@params`:

```
> clust@params
[[1]]
[[1]]$mus
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14]
[1,]    1    1    1    1    1    1    1    1    1     2     1     2     1     1
[2,]    2    2    1    1    1    2    1    1    2     2     1     3     2     1
[3,]    3    4    3    3    1    4    3    2    3     4     2     4     3     4
```

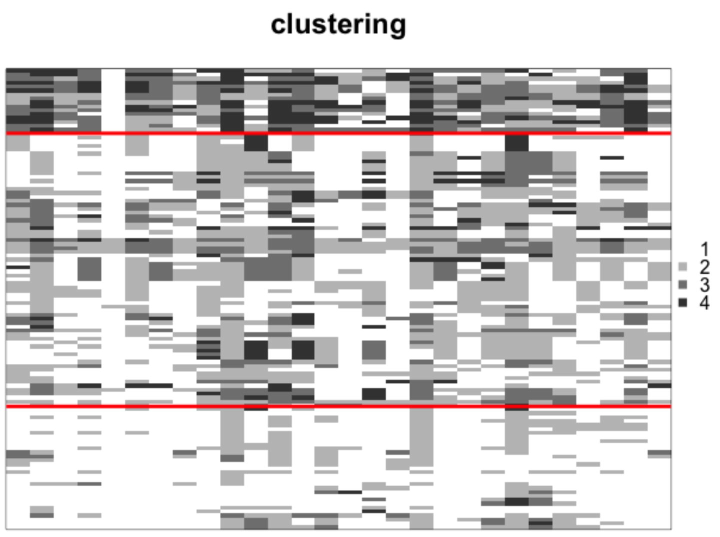


Figure 5: Clustering obtained when following the example provided.

```
     [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24] [,25] [,26]
[1,]     1     1     1     2     1     1     1     2     1     1     1     1
[2,]     1     1     1     2     2     1     2     2     1     2     1     1
[3,]     1     1     1     4     3     3     3     3     2     2     2     3
     [,27] [,28]
[1,]     1     1
[2,]     1     1
[3,]     4     1

[[1]]$pis
          [,1]      [,2]     [,3]      [,4] [,5]      [,6]      [,7]      [,8]
[1,] 0.8079608 0.6673682 0.961979 0.7770536    1 0.9619790 1.0000000 0.8852379
[2,] 0.3946294 0.3736864 0.722322 0.4690402    1 0.3567357 0.5546162 0.6402318
[3,] 0.4319502 0.5928978 0.347433 0.4930463    1 0.2718517 0.5888644 0.3310052
          [,9]     [,10]     [,11]     [,12]     [,13]     [,14]     [,15]
[1,] 0.9246885 0.5903583 0.6951631 0.5438752 0.9226941 0.4932884 0.8825371
[2,] 0.4767814 0.6937982 0.1481492 0.1859040 0.1176366 0.6624020 0.7916167
[3,] 0.3220447 0.7079570 0.4084469 0.5779180 0.5745136 0.1691940 0.3161048
         [,16]     [,17]     [,18]     [,19]     [,20]     [,21]     [,22]
[1,] 0.8036703 0.7364791 0.6643935 1.0000000 0.9619790 0.6951631 0.5681893
[2,] 0.3054584 0.8394348 0.5440131 0.3395749 0.4757433 0.4142450 0.3805989
[3,] 0.1255990 0.4281432 0.5470879 0.4280508 0.2300193 0.5776385 0.2632960
         [,23]     [,24]     [,25]     [,26]     [,27]     [,28]
[1,] 0.4905033 0.5510665 0.8167944 0.7477762 0.8521366 0.9226941
[2,] 0.3870155 0.4064222 0.6484691 0.4666815 0.3530825 0.6599010
[3,] 0.4183768 0.4709545 0.1959082 0.5465595 0.6419857 0.4174326
```

`clust@params` is a list: when the data have D numbers of levels as in Figure 3, the list is $D-$long. Here the data has only one number of levels, so `clust@params` has one element. Each element of the list has two attributes, `pis` and `mus`. They indicate the π and μ values for each row-cluster and each column. Here, we see that, as observed with Figure 5, the first row-cluster has globally lower parameters μ, which means that people from this cluster globally answered in a more positive way to the questions. We also note that the π parameters for the fifth variable (the fifth question) are all equal to 1. This means that the dispersion around the position μ is null. When observing the μ parameters for the fifth variables, they are also all equal to 1. This means that everybody answered in a positive way to this question. The fifth question of the EORTC QLQ-C30 questionnaire is "Do you need help with eating, dressing, washing yourself or using the toilet?". Therefore, we know that none of the participants had problems getting ready and eating the week before they answered the questionnaire.

Choosing G**.** In the example above, the choice for G was made by performing several clustering with $G = (2,3,4)$. Using the command `clust@icl`, we can find out which result has the highest ICL value. The G with the highest ICL-BIC was retained, that is to say $G = 3$. The code to perform these clusterings is available in the Appendix "ICL search for clustering".

Performing co-clustering

Co-clustering setting. Once again, this section uses the `dataqol` dataset. The co-clustering is performed using the `boscoclust` function:

```
set.seed(1)
coclust <- boscoclust(x = x, kr = 3, kc = 3, m = 4,
                nbSEM = nbSEM, nbSEMburn = nbSEMburn,
                nbindmini = nbindmini, init = init)
```

As in the clustering context, the result can be plotted with the command below, as in Figure 6.

```
plot(coclust)
```

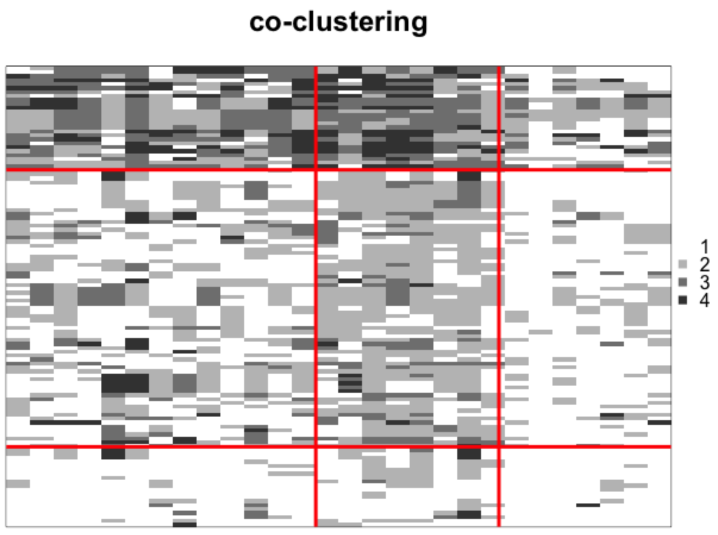


Figure 6: Co-clustering obtained when following the example provided.

In this case, the algorithm highlights a structure amid the rows, as for the clustering Figure 5. In addition, it also reveals a structure inherent to the columns: for example, the third column-cluster is lighter than the others, consequently, these questions were globally responded to in a more positive way.

Co-clusters interpretation. Once again, the parameters of the co-clustering are available through the command `coclust@params`:

```
> coclust@params
[[1]]
[[1]]$mus
     [,1] [,2] [,3]
[1,]    1    1    1
[2,]    1    2    1
[3,]    3    3    1

[[1]]$pis
          [,1]      [,2]      [,3]
[1,] 0.8496224 0.6266097 0.9426305
[2,] 0.4876194 0.5340329 0.7722278
[3,] 0.2638594 0.3044552 0.3623779
```

In order to find out which questions belong to the third column-cluster (the one whose corresponding blocks are lighter), we need the command `coclust@zc`, which indicates the column-cluster of each column. `coclust@zc` is also a list of length D (when we have different numbers of levels). Here, $D = 1$ so we need `coclust@zc[[1]]`:

```
which(coclust@zc[[1]] == 3)
[1]  3  5  8 15 17 25 28
```

We know that questions 3, 5, 8, 15, 17, 25 and 28 are globally the ones that were answered the more positively. Here is the list of these questions in the EORTC QLQ C30:

- 3. Do you have any trouble taking a short walk outside of the house?
- 5. Do you need help with eating, dressing, washing yourself or using the toilet?
- 8. During the past week, were you short of breath?
- 15. During the past week, have you vomited??
- 17. During the past week, have you had diarrhea?
- 25. During the past week, have you had difficulty remembering things?
- 28. During the past week, has your physical condition or medical treatment caused you financial difficulties?

Choosing G and H. In the examples above, the choice for G and H were made by performing several co-clusterings with $G = (2, 3, 4)$ and $H = (2, 3, 4)$. In both cases, the couple (G, H) with the highest ICL-BIC value was retained, i.e., for $(G, H) = (3, 3)$. The code to search the highest ICL value is given in the Appendix "ICL search for co-clustering"[1].

Missing values.

In this section we use the `dataqol` dataset. It has 1.1% missing values (40 elements are missing in the matrix). The SEM-algorithm can handle these values since at each Expectation step (see Algorithm 1, which computes the expectation of the missing values. The following code obtains the index of the missing values and prints their values imputed by the clustering (or co-clustering) algorithm in the Section "Performing co-clustering" (or the Section "Performing clustering").

```
missing <- which(is.na(x))
missing

values.imputed.clust <- clust@xhat[[1]][missing]
values.imputed.clust

values.imputed.coclust <- coclust@xhat[[1]][missing]
values.imputed.coclust

> missing
 [1]  148  177  278  352  380  440  450  559  996 1058 1496 1513 1611 1883 1981
[16] 2046 2047 2050 2085 2285 2402 2450 2514 2517 2518 2663 2754 2785 2900 2902
[31] 2982 2986 3060 3152 3366 3367 3368 3520 3572 3602
> values.imputed.clust
 [1] 4 4 4 1 1 1 4 4 1 4 4 4 4 1 4 1 1 1 4 4 4 4 4 4 4 4 4 4 4 1 1 1 4 1 1 1
> values.imputed.coclust
 [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

We see that the co-clustering and the clustering algorithm had different values imputed for the missing data.

Comparison of clustering and co-clustering.

Co-clustering as parsimonious clustering. Co-clustering can be seen as a parsimonious way of performing clustering, which is why these two techniques are compared here. For example, the interpretation of row-clusters is more precise with the co-clustering. Indeed, in Figure 5, the row-clusters can be seen as a group of people who globally replied positively, a group of people who

[1]In case of several numbers of levels, testing all the possible values for $(G, H_1, ..., H_D)$ can be tedious. In such cases, users are invited to implement a specific heuristic strategy as in Selosse et al. (2019).

replied negatively, and a third group that replied in between. On the other hand, in Figure 6, an inherent structure of the data is better highlighted and adds more information: for each row-cluster, it is also easy to detect the questions that were replied to negatively. Co-clustering can therefore be seen as a more efficient way of performing clustering. Furthermore the interpretation of the parameters was easier with the co-clustering result because it only had 18 parameters: $kr \times kc$ for $\boldsymbol{\pi}$ and $kr \times kc$ for $\boldsymbol{\mu}$. The clustering result had 168 parameters ($kr \times J$ for $\boldsymbol{\pi}$ and $kr \times J$ for $\boldsymbol{\mu}$), which is a lot to process for the user.

ARI values on row partitions The Adjusted Rand Index (Rand, 1971) was computed on row partitions of co-clustering and clustering results, using the package **mclust** Scrucca et al. (2016).

```
mclust::adjustedRandIndex(coclust@zr, clust@zr)
```

The value obtained is `0.41`, meaning that co-clustering creates a row partition related to that created by the clustering, without being identical.

Setting the SEMburn and nbSEMburn arguments

The SEM-algorithm can be slow at reaching its stationary state, depending on the dataset. After having chosen arbitrary `nbSEM` and `nbSEMburn` arguments (in practice at least higher than 50), the stability of the algorithm has to be verified. For this reason, all the functions of the **ordinalClust** package also return parameters estimations at each iteration of the SEM-algorithm. Indeed, the `pichain`, `rhochain` and `paramschain` slots represent the γ, $\boldsymbol{\rho}$ and $\boldsymbol{\alpha}$ values, respectively, for each iteration. As a result, the evolution of the parameters can be analyzed and users can be confident that the returned parameters are well estimated. In the co-clustering case, for example, the evolution of the parameters can be visualized through a plot:

```
par(mfrow=c(3,3))
for(kr in 1:3){
    for(kc in 1:3){
        toplot <- rep(0, nbSEM)
        for(i in 1:nbSEM){
            toadd <- coclust@paramschain[[1]]$pis[kr,kc,i]
            toplot <- c(toplot, toadd)
        }
        plot.default(toplot, type = "l",ylim = c(0,1),
                 col = "hotpink3", main = "pi",
                 ylab = paste0("pi_", kr, kc, "values"),
                 xlab = "SEM-Gibbs iterations")
    }
}
```

In Figure 7, we observe that the parameters reach their stationary state before the 100^{th} iteration. In this case, a burn-in period of 100 iterations (corresponding to `nbSEMburn=100`) is therefore enough. The total number of iterations corresponds to the argument `nbSEM=150`, so 50 iterations are used to approximate the parameters.

Handling data with different numbers of levels

If users wish to execute one of the functions described previously on variables with different m, then they should use the same function with some changes to the arguments definitions. Let us assume that the data is made of D different numbers of levels. First of all, the columns of matrix matrix `x` must be grouped by same number of levels `m[d]`. The additional changes for the arguments to pass are listed below:

- `m` must be a vector of length D. The d^{th} element indicates the number of levels for the d^{th} group of variables.
- `kc` must be a vector of length D. The d^{th} element indicates the number of column-clusters for the d^{th} group of variables.
- `idx_list` is a new vector argument of length D. The d^{th} item of the vector indicates the index of the first column that have the number of levels `m[d]`.

An example on the `dataqol` dataset is available in the Appendix "Handling different numbers of levels".

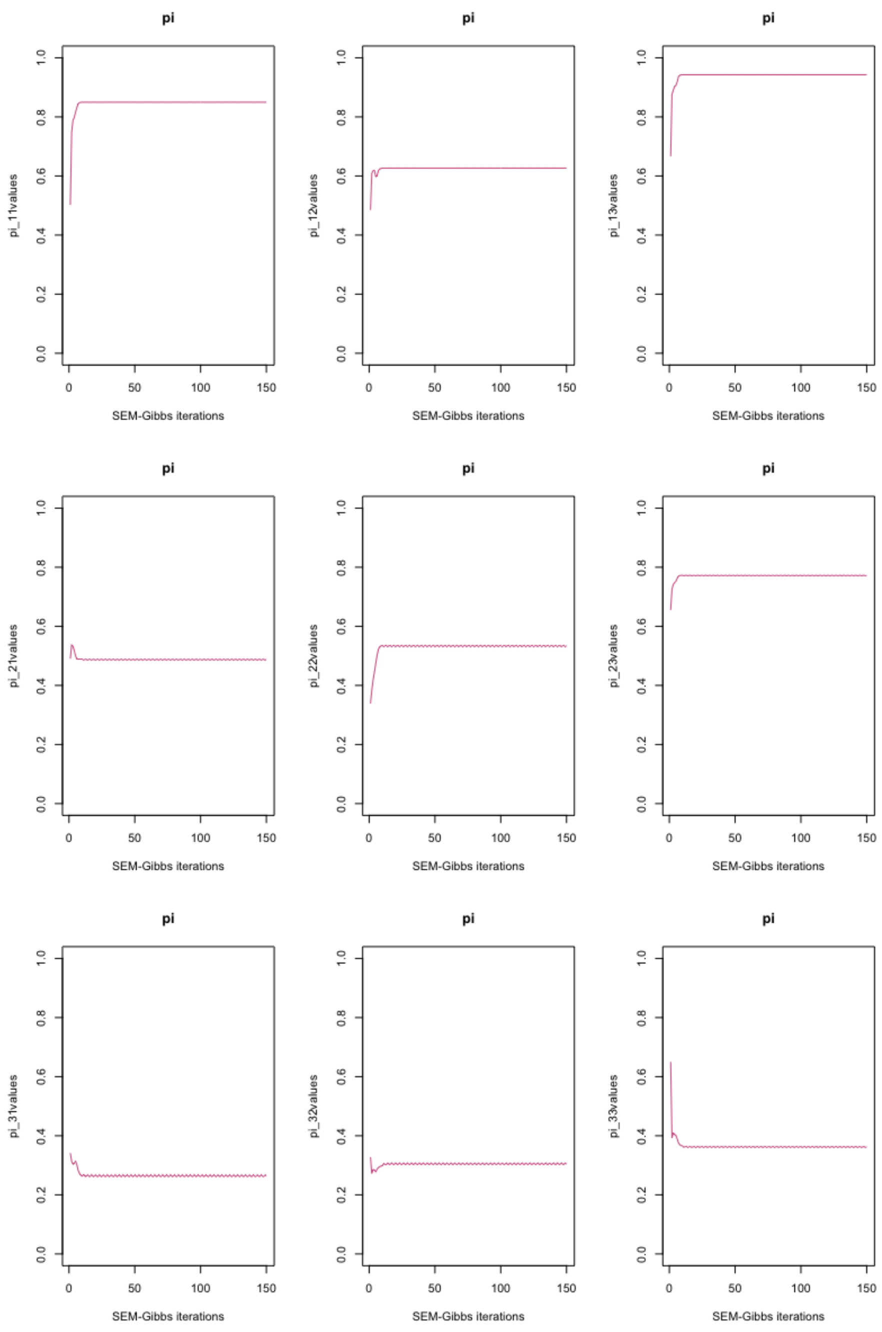


Figure 7: Evolution of π parameters through SEM-Gibbs iterations, in the clustering example. We observe that the parameters have reached a stationary state with time.

Conclusion

The **ordinalClust** package presented in this paper implements several methods for analyzing ordinal data. First, it implements a clustering and co-clustering framework based on the Latent Block Model, coupled with a SEM-Gibbs algorithm and the BOS distribution. Moreover, it defines a novel approach to classify ordinal data. For the classification method, two models are proposed, so that users can introduce parsimony in their analyses. Similarly, it has been shown that the co-clustering method provides a parsimonious way of performing clustering. The framework is able to handle missing values which is notably relevant in the case of real datasets. Finally, these techniques are also implemented in the case of dataset with ordinal data with several numbers of levels. The package **ordinalClust** is available on the Comprehensive R Archive Network (CRAN), and is still under active development. A future work will implement the method defined in Gelman and Rubin (1992), to automatically define the number of iterations of the SEM-Gibbs algorithm.

Bibliography

A. Agresti. *Analysis of ordinal categorical data. Wiley Series in Probability and Statistics*, pages 397–405. John Wiley & Sons, Inc., 2012. [p61]

A. Anota, M. Savina, C. Bascoul-Mollevi, and F. Bonnetain. Qolr: An r package for the longitudinal analysis of health-related quality of life in oncology. *Journal of Statistical Software, Articles*, 77(12): 1–30, 2017. [p67]

C. Biernacki and J. Jacques. Model-Based Clustering of Multivariate Ordinal Data Relying on a Stochastic Binary Search Algorithm. *Statistics and Computing*, 26(5):929–943, 2016. [p62, 63, 65]

C. Biernacki, G. Celeux, and G. Govaert. Assessing a mixture model for clustering with the integrated completed likelihood. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(7):719–725, 2000. [p66]

P.-C. Bürkner. brms: An r package for bayesian multilevel models using stan. *Journal of Statistical Software, Articles*, 80(1):1–28, 2017. ISSN 1548-7660. [p61]

A. J. Cannon. *monmlp: Multi-Layer Perceptron Neural Network with Optional Monotonicity Constraints*, 2017. R package version 1.1.5. [p61]

R. H. B. Christensen. ordinal—regression models for ordinal data, 2015. R package version 2015.6-28. [p61]

M. Corduas. A statistical procedure for clustering ordinal data. *Quaderni di statistica*, 10:177–189, 2008. [p62]

M. Corneli, C. Bouveyron, and P. Latouche. *ordinalLBM: Co-Clustering of Ordinal Data via Latent Continuous Random Variables*, 2019. [p62]

M. Corneli, C. Bouveyron, and P. Latouche. Co-clustering of ordinal data via latent continuous random variables and not missing at random entries. *Journal of Computational and Graphical Statistics*, 0(ja): 1–39, 2020. [p62]

A. D'Elia and D. Piccolo. A mixture model for preferences data analysis. *Computational Statistics & Data Analysis*, 49(3):917–934, June 2005. [p62]

A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of he Royal Statistical Society, series B*, 39(1):1–38, 1977. [p63, 64]

A. Gelman and D. Rubin. Inference from iterative simulation using multiple sequences. *Statistical Science*, 7(4):457–472, 1992. ISSN 08834237. [p75]

M. Giordan and G. Diana. A clustering method for categorical ordinal data. *Communications in Statistics - Theory and Methods*, 40(7):1315–1334, 2011. [p62]

G. Govaert and M. Nadif. Clustering with block mixture models. *Pattern Recognition*, 36:463–473, 2003. [p62]

J. A. Hartigan and M. A. Wong. A k-means clustering algorithm. *JSTOR: Applied Statistics*, 28(1): 100–108, 1979. [p65]

M. C. Heredia-Gómez, S. García, P. A. Gutiérrez, and F. Herrera. Ocapis: R package for ordinal classification and preprocessing in scala. *Progress in Artificial Intelligence*, 2019. ISSN 2192-6360. [p61]

R. Hornung. *ordinalForest: Ordinal Forests: Prediction and Variable Ranking with Ordinal Target Variables*, 2019a. R package version 2.3-1. [p61]

R. Hornung. Ordinal forests. *Journal of Classification*, pages 1–14, 2019b. [p61]

J. M. J. Vermunt. *Technical Guide for Latent GOLD 4.0: Basic and Advanced. Statistical Innovations Inc.* Belmont, Massachussetts, 2005. [p61]

J. Jacques and C. Biernacki. Model-based co-clustering for ordinal data. *Computational Statistics & Data Analysis*, 123:101 – 115, 2018. ISSN 0167-9473. [p62, 66]

F.-X. Jollois and M. Nadif. Classification de données ordinales : modèles et algorithmes. In *41èmes Journées de Statistique, SFdS, Bordeaux*, Bordeaux, France, 2009. [p62]

F. E. H. Jr. *rms: Regression Modeling Strategies*, 2019. R package version 5.1-3.1. [p61]

C. Keribin, G. Govaert, and G. Celeux. Estimation d'un modèle à blocs latents par l'algorithme SEM. In *42èmes Journées de Statistique*, Marseille, France, 2010. [p64]

R. S. Maria Iannario, Domenico Piccolo. *CUB: A Class of Mixture Models for Ordinal Data*, 2018. R package version 1.1.3. [p62]

D. McParland and I. C. Gormley. Clustering ordinal data via latent variable models. In B. Lausen, D. Van den Poel, and A. Ultsch, editors, *Algorithms from and for Nature and Life: Classification and Data Analysis*, pages 127–135. Springer International Publishing, Switzerland, 2013. [p62]

D. McParland and I. C. Gormley. *clustMD: Model Based Clustering for Mixed Data*, 2017. R package version 1.2.1. [p62]

M. Ranalli and R. Rocci. Mixture models for ordinal data: A pairwise likelihood approach. *Statistics and Computing*, 26(1-2):529–547, Jan. 2016. ISSN 0960-3174. [p62]

W. M. Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 66(336):846–850, 1971. [p74]

L. Scrucca, M. Fop, T. B. Murphy, and A. E. Raftery. mclust 5: clustering, classification and density estimation using Gaussian finite mixture models. *The R Journal*, 8(1):205–233, 2016. [p74]

M. Selosse, J. Jacques, C. Biernacki, and F. Cousson-Gélie. Analysing a quality-of-life survey by using a coclustering model for ordinal data and some dynamic implications. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 68(5):1327–1349, 2019. [p62, 67, 73]

W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S*. Springer, New York, fourth edition, 2002. ISBN 0-387-95457-0. [p61]

T. W. Yee. The vgam package for categorical data analysis. *J Stat Softw*, 2010. [p61]

Appendix

Specificity and sensitivity

The following code computes the specificities, and sensitivities obtained with the different kc in the section "Performing classification":

```
library(caret)

actual <- v.validation - 1

specificities <- rep(0,length(kcol))
sensitivities <- rep(0,length(kcol))

for(i in 1:length(kcol)){
  prediction <- unlist(as.vector(preds[i,])) - 1
  u <- union(prediction, actual)
  conf_matrix <- table(factor(prediction, u),factor(actual, u))
  sensitivities[i] <- recall(conf_matrix)
  specificities[i] <- specificity(conf_matrix)
}

sensitivities
specificities
```

```
> sensitivities
[1] 0.6666667 1.0000000 1.0000000 1.0000000 0.7777778
> specificities
[1] 0.4444444 0.5555556 0.3333333 0.5555556 0.6666667
```

ICL search for clustering

```
set.seed(1)

library(ordinalClust)
data("dataqol")
M <- as.matrix(dataqol[,2:29])

nbSEM <- 150
nbSEMburn <- 100
nbindmini <- 2
init <- "randomBurnin"
percentRandomB <- c(50)
icl <- rep(0,3)

for(kr in 2:4){
    object <- bosclust(x = M, kr = kr, m = 4, nbSEM = nbSEM,
                nbSEMburn = nbSEMburn, nbindmini = nbindmini,
                percentRandomB = percentRandomB, init = init)

    if(length(object@icl)) icl[kr-1] <- object@icl
}
icl
```

```
> icl
[1] -3713.311 -3192.351 0
```

We see that the clustering algorithm could not find a solution without an empty cluster for `kr = 4`. The highest icl is for `kr = 3`.

ICL search for co-clustering

```
set.seed(1)
library(ordinalClust)
```

```
data("dataqol")
M <- as.matrix(dataqol[,2:29])

nbSEM <- 150
nbSEMburn <- 100
nbindmini <- 2
init <-  "randomBurnin"
percentRandomB <- c(50, 50)
icl <- matrix(0, nrow = 3, ncol = 3)

for(kr in 2:4){
    for(kc in 2:4){
        object <- boscoclust(x = M,kr = kr, kc = kc, m = 4, nbSEM = nbSEM,
                        nbSEMburn = nbSEMburn, nbindmini = nbindmini,
                        percentRandomB = percentRandomB, init = init)
         if(length(object@zr)){
        icl[kr-1, kc-1] <- object@icl
       }
    }

}

icl
```

```
> icl
          [,1]      [,2]      [,3]
[1,] -3529.423     0.000 -3503.235
[2,]     0.000 -3373.573     0.000
[3,]     0.000 -3361.628 -3299.497
```

We note that the co-clustering algorithm could not find a solution without an empty cluster for `(kr,kc) = (2,3),(3,2),(3,4),(4,2)`. The highest ICL-BIC is obtained when `(kr,kc) = (3,3)`.

Handling different numbers of levels

The following code shows how to handle different numbers of levels in a co-clustering context. It may take several minutes due to the high number of levels of the two last columns.

```
set.seed(1)

library(ordinalClust)

# loading the real dataset
data("dataqol")

# loading the ordinal data
x <- as.matrix(dataqol[,2:31])


# defining different number of categories:
m <- c(4,7)


# defining number of row and column clusters
krow <- 3
kcol <- c(3,1)

# configuration for the inference
nbSEM <- 20
nbSEMburn <- 15
nbindmini <- 2
init <- 'random'
```

```
d.list <- c(1,29)

# Co-clustering execution
object <- boscoclust(x = x,kr = krow, kc = kcol, m = m,
                idx_list = d.list, nbSEM = nbSEM,
                nbSEMburn = nbSEMburn, nbindmini = nbindmini,
                init = init)
```

Margot Selosse
Université de Lyon, Lyon 2, ERIC EA 3083.
5 Avenue Pierre Mendés France, 69500 Bron
France
margot.selosse@gmail.com

Julien Jacques
Université de Lyon, Lyon 2, ERIC EA 3083.
5 Avenue Pierre Mendés France, 69500 Bron
France
julien.jacques@univ-lyon2.fr

Christophe Biernacki
Inria, Université de Lille, CNRS Université Lille - UFR de Mathématiques - Cité Scientifique - 59655 Villeneuve d'Ascq Cedex
France
christophe.biernacki@inria.fr

KSPM: A Package For Kernel Semi-Parametric Models

by Catherine Schramm, Sébastien Jacquemont, Karim Oualkacha, Aurélie Labbe and Celia M.T. Greenwood

Abstract Kernel semi-parametric models and their equivalence with linear mixed models provide analysts with the flexibility of machine learning methods and a foundation for inference and tests of hypothesis. These models are not impacted by the number of predictor variables, since the kernel trick transforms them to a kernel matrix whose size only depends on the number of subjects. Hence, methods based on this model are appealing and numerous, however only a few R programs are available and none includes a complete set of features. Here, we present the **KSPM** package to fit the kernel semi-parametric model and its extensions in a unified framework. **KSPM** allows multiple kernels and unlimited interactions in the same model. It also includes predictions, statistical tests, variable selection procedure and graphical tools for diagnostics and interpretation of variable effects. Currently **KSPM** is implemented for continuous dependent variables but could be extended to binary or survival outcomes.

Introduction

In the last decades, the popularity and accessibility of machine learning has increased as a result of both the availability of big data and technical progress in computer science. The flexibility of machine learning methods enables avoidance of assumptions about functional relationships, such as strong linear or additive hypotheses, that are often involved in classical statistical models. However, this flexibility and adaptability also limits the capacity to interpret results or make inference. When understanding and inference are required, simpler statistical models are often preferred, as they are easy to understand and implement. Methods from the machine learning field might better model complex relationships, and yet would be more attractive if the results could be made interpretable. With this goal in mind, Liu et al. (2007) clearly demonstrated the under-appreciated equivalence between a machine learning tool and a classical statistic model through the link between kernel semi-parametric models – developed first in the machine learning field – and more classical linear mixed models. This equivalence allows analysts to take advantage of knowledge advances in both the machine learning domain and the traditional statistical inference domain, including hypotheses testing, when using kernel semi-parametric models (Table 1).

Features of kernel models	
from traditional statistical models	from machine learning models
• Inference, confidence/prediction intervals • Tests, p values • Information criteria (variable selection) • Interpretation via estimation of functional form of variable effects on outcome	• No need to explicitly define outcome-predictors relationship • May deal with a large amount of data (big data and fat data) • Potential for reinforcement learning

Table 1: Features of kernel models combine features from traditional statistical models and features from machine learning models.

The kernel semi-parametric model assumes that the outcome is related to the set of variables through an unknown smooth function, which is simply approximated by computing the similarity matrix between subjects according to the set of variables and a chosen kernel function (i.e., the kernel trick). Matrix size depends only on the number of subjects, making kernel models particularly suited to datasets where the number of features is very large ($p >> n$). The equivalence between kernel semi-parametric models and linear mixed models motivates a score test, which is simple to implement, for the simultaneous effect of all variables on the outcome.

Methods based on the kernel semi-parametric model and its extensions are appealing and numerous, but only a few programs are available, and none includes a complete set of the features that correspond to recent developments. Table 2 gives some examples of existing R packages according to their features of interest.

	Adjustment	*User's own kernel*	*Single kernel test*	*Test of interaction*	*Predictions*	*Interpretation plot*	*Diagnostic plots*	*Variable selection*
coxme	✓	✓						
SKAT		✓	✓					
SPA3G				✓				
KRLS					✓	✓		
e1071					✓			
KSPM	✓	✓	✓	✓	✓	✓	✓	✓

Table 2: Features incorporated in **KSPM** (Schramm, 2020), as well as in several other R packages for kernel nonparametric or semiparametric models: **coxme** (Therneau, 2018), **SKAT** (Lee et al., 2017), **KRLS** (Hainmueller and Hazlett, 2017), **e1071** (Meyer et al., 2018) and **SPA3G** (Li and Cui, 2012a). **Adjustment** refers to models including a kernel for adjusting the model on correlation structure similarly to a random factor. **User's own kernel** refers to a kernel function explicitly defined by the user of the package, in contrast to traditional kernel functions that are already implemented in the package. **Single kernel test** refers to the test of the joint effect of a set of variables on the outcome. **Test of interaction** refers to the interaction between two sets of variables and its effect on the outcome. **Predictions** refers to the possibility of displaying predictions with confidence and prediction intervals. **Interpretation plot** refers to graphical tools for interpretation of individual effects of each variable in the kernel on the outcome. **Diagnostic plots** refers to graphical tools based on residuals and leverage measures to check the validity conditions of a model and identify outlier samples. **Variable selection** refers to the implementation of a variable selection procedure.

Several packages in Table 2 were developed in the genetics field where interest often lies in testing the contribution of a group of variables (variants) simultaneously, notably through the sequence kernel association test (SKAT) for single nucleotide polymorphisms (Wu et al., 2011; Chen et al., 2016). Extensions from single to multiple kernel model were motivated by (i) interaction tests (Li and Cui, 2012b; Wang et al., 2017; Ge et al., 2015; Marceau et al., 2015) and (ii) estimating the conditional effect of one set of variables after adjusting for another set of variables, or after adjusting for population structure (Oualkacha et al., 2013). The latter goal may be achieved easily using the `lmekin()` function from the **coxme** package under mixed model theory where kinship matrix corresponds to the kernel matrix measuring similarity between subjects, and defines the covariance matrix of the random effect term.

In the machine learning field, packages like **e1071** have been developed to perform accurate predictions and they focus on a single kernel function. Traditionally, model interpretation has remained an outstanding challenge of the machine learning field. However it has been recently demonstrated that the kernel semi-parametric models can be used to interpret effects of variables on the outcome through a graphical tool based on derivatives (Hainmueller and Hazlett, 2013) and this is implemented in the **KRLS** package. When the kernel function and the corresponding approximated smooth function are differentiable with respect to the variable of interest, pointwise derivatives capture the effect of this variable on the outcome.

Since researchers may be interested in all of these features, we have consolidated them in the R package **KSPM**. The package and a vignette including detailed examples are available from the comprehensive R archive network (CRAN) at https://CRAN.R-project.org/package=KSPM. Our package, currently designed for continuous outcomes and normal errors, fits kernel semi-parametric models and their extensions in a unified framework incorporating all the previously described model fitting features and tests. It allows multiple kernels and unlimited interactions in the same model. Although most popular kernel functions are available in the package, the user also has the option of designing and using his/her own kernel functions. Furthermore, whenever interest lies in prediction or making inference from the model, diagnostic assessments of the model may be performed through graphical tools to detect data points with large residuals or high leverage that may greatly influence the results. Finally, we have also included a variable selection procedure based on Akaike's and Bayesian information criteria (AIC and BIC). These last two options are not included in other software packages.

The **KSPM** package is a new tool for semi-parametric regression. It is not competing with other R packages for semi- or non-parametric regression models since either our methods or our objectives are different. Indeed, the estimation method involved in **KSPM** is based on regularized least squares in the kernel Hilbert space and should not be confused with local kernel smoothing based on Nadaraya-Watson estimator (**np** package, Racine and Hayfield (2020)). Similarly, **KSPM** is also different from other smoothing methods connecting regression segments through knots among which we may cite splines (**mgcv** package, Wood (2020)).

However, since the kernel semi-parametric model is equivalent to a linear mixed effect model, our package could be compared to **lme4** (Bates et al., 2019) or **nlme** (Pinheiro et al., 2019) packages for linear mixed effect models. Although it is possible to obtain similar inference from any variables involved in a linear part of each model, and to obtain similar predictions from both overall models, loss function maximization are different. Moreover, **KSPM** has the advantage of being easier-to-use as the user does not need to compute kernel matrix nor matrix of interactions and provides interpretations for variable effects that cannot be obtained with traditional packages.

Kernel semi-parametric models

Single kernel semi-parametric model

Let $Y = (Y_1, ..., Y_n)^\top$ be a $n \times 1$ vector of continuous outcomes where Y_i is the value for subject i ($i = 1, ..., n$), and X and Z are $n \times p$ and $n \times q$ matrices of predictors, such that Y depends on X and $Z = (\mathbf{z_{.1}}, ..., \mathbf{z_{.q}})$ as follows:

$$Y = X\beta + h(Z) + e \tag{1}$$

where β is a $p \times 1$ vector of regression coefficients for the linear parametric part, $h(.)$ is an unknown smooth function and e is an $n \times 1$ vector of independent and homoscedastic errors such as $e_i \sim \mathcal{N}(0, \sigma^2)$. Throughout this article, $X\beta$ will be referred to as the linear part of the model and we assume that X contains an intercept. $h(Z)$ will be referred to as the kernel part. The function $h(.)$ need not be explicitly specified but it is assumed to lie in $\mathcal{H}_k$, the function space generated by a kernel function $k(.,.)$ which is, by definition, symmetric and positive definite. We note that K, the $n \times n$ Gram matrix of $k(.,.)$ such that $K_{ij} = k(\mathbf{z_{i.}}, \mathbf{z_{j.}})$, represents the similarity between subjects i and j according to Z. Of note, in the estimation process, $h(.)$ will be expressed as a linear combination of $k(.,.)$ (see Equation (7)). See "Fitting the kernel semi-parametric model with `kspm`" for straightforward coding.

The multiple kernel semi-parametric model

Now suppose there are L matrices $Z_1, ..., Z_L$ of dimension $n \times q_1$, ..., $n \times q_L$ respectively. Keeping a similar notation, Y can be assumed to depend on X and Z_1, ..., Z_L as follows:

$$Y = X\beta + \sum_{\ell=1}^{L} h_\ell(Z_\ell) + e \tag{2}$$

where $\forall \ell \in \{1, ..., L\}$, h_ℓ is an unknown smooth function from $\mathcal{H}_{k_\ell}$, the function space generated by a kernel function $k_\ell(.,.)$ whose Gram matrix is noted K_ℓ. We assume $\forall \ell \neq m$, $h_\ell(Z_\ell) \neq h_m(Z_m)$ either because $Z_\ell \neq Z_m$ or $k_\ell(.,.) \neq k_m(.,.)$ or both. Note that when $L = 1$, the model corresponds to Equation (1).

Suppose there is an interaction between two sets of variables represented by Z_1 and Z_2, $n \times q_1$ and $n \times q_2$ matrices, and Y depends on X, Z_1 and Z_2 as follows:

$$Y = X\beta + h_1(Z_1) + h_2(Z_2) + h_{12}(Z_1, Z_2) + e \tag{3}$$

where $h_1(.)$, $h_2(.)$ and $h_{12}(.,.)$ are unknown smooth functions from $\mathcal{H}_{k_1}$, $\mathcal{H}_{k_2}$ and $\mathcal{H}_{k_{12}}$ respectively. The $h_{12}(.,.)$ function represents the interaction term if its associated kernel function $k_{12}(.,.)$ is defined as the product of kernel functions $k_1(.,.)$ and $k_2(.,.)$ as follows:

$$k_{12}\left((Z_1, Z_2)_i, (Z_1, Z_2)_j\right) = k_1(\mathbf{z_{1i.}}, \mathbf{z_{1j.}}) \times k_2(\mathbf{z_{2i.}}, \mathbf{z_{2j.}}) \tag{4}$$

such that $K_{12} = K_1 \odot K_2$ where $\odot$ is the Hadamard product (Ge et al., 2015). Model (3) is a particular case of the multiple kernel semi-parametric model (2) with $L = 3$, $Z_3 = (Z_1, Z_2)$, $q_3 = q_1 + q_2$ and $h_3 = h_{12}$. Of notes, in **KSPM**, different kernel choices can be made for $k_1(.,.)$ and $k_2(.,.)$. Obviously, this 2-way interaction model could be generalized to higher order interaction terms in a similar manner.

Link with linear mixed models

As shown by Liu et al. (2007), model (2) is equivalent, without additional conditions, to the following linear mixed model:

$$Y = X\beta + \sum_{\ell=1}^{L} h_\ell + e \tag{5}$$

where $\forall \ell \in \{1, ..., L\}$, $h_\ell \sim \mathcal{N}(0, \tau_\ell K_\ell)$ with K_ℓ is a matrix of similarity between subjects as defined in model (2). Throughout the paper, we denote the variance parameters as $\theta = (\tau_1, ..., \tau_L, \sigma^2)$ and $\Sigma_\theta = \sum_{\ell=1}^{L} \tau_\ell K_\ell + \sigma^2 I$, the variance-covariance matrix of Y, where I is the $n \times n$ identity matrix.

Estimation of parameters

The parameter estimates can be obtained either by maximizing the penalized log-likelihood associated with the kernel semi-parametric model (2), or the log-likelihood derived from the corresponding linear mixed model (5) (Liu et al., 2007). Even though the latter option may be computationally less time-consuming, we implemented the **KSPM** package using a penalized log-likelihood associated with the kernel semi-parametric model, because this method leads to an estimation of h useful for prediction, and also leads to suitable approaches for interpretation through kernel derivatives. Estimation of parameters is obtained by maximizing the penalized log-likelihood function:

$$l(\beta, h) = -\frac{1}{2}\sum_{i=1}^{n}\left(Y_i - X_i^\top\beta - \sum_{\ell=1}^{L} h_\ell(Z_{\ell i})\right)^2 - \frac{1}{2}\sum_{\ell=1}^{L}\lambda_\ell \parallel h_\ell \parallel^2_{\mathcal{H}_{k_\ell}} \tag{6}$$

where λ_1, ..., λ_L are tuning parameters associated with each smooth function $h_1(.)$, ..., $h_L(.)$ and $\parallel . \parallel^2_{\mathcal{H}_{k_\ell}}$ defines the inner product on $\mathcal{H}_{k_\ell}$ space. According to the Representer theorem (Kimeldorf and Wahba, 1971), the functions h_1, ..., h_L satisfying Equation (6) can be expressed as:

$$\forall \ell \in \{1, ..., L\}, \quad h_\ell(.) = \sum_{i=1}^{n} \alpha_{\ell i} k(., Z_{\ell i}) \tag{7}$$

where $\forall \ell \in \{1, ..., L\}$, $\alpha_\ell = (\alpha_{\ell 1}, ..., \alpha_{\ell n})^\top$ is a $n \times 1$ vector of unknown parameters. Estimating the kernel semi-parametric model parameters consists in estimating $\alpha_1, ..., \alpha_L$ and β. Then, estimators of $h_1(.)$, ..., $h_L(.)$ are deduced from $\hat{\alpha}_1, ..., \hat{\alpha}_L$.

In **KSPM**, we estimate penalization parameters by minimizing the mean sum of squares of the leave-one-out errors (LOOE). An advantage of LOOE compared to other cross-validation methods is that we do not need to recompute new model(s), because its value may be derived directly from the primary model parameters:

$$\forall i \in \{1, ..., n\}, \qquad \text{LOOE}_i(\lambda_1, ..., \lambda_L) = \frac{Y_i - \hat{Y}_i}{1 - H_{ii}} \tag{8}$$

where H_{ii} is the i^{th} diagonal element of the Hat matrix H such as $\hat{Y} = HY$.

Penalization parameters and tuning parameters, if applicable, are estimated simultaneously during the optimization algorithm, by minimizing the LOOE. If only one parameter needs to be estimated, the convexity of the function to be minimized makes the problem easier and the convergence faster. In that case, **KSPM** uses the standard `optimize()` function from the basic R package, based on the golden section search algorithm (Brent, 2013). When several hyperparameters need to be estimated, the resulting function to be minimized may not be convex. Hence, more complex optimization algorithms should be envisaged, and **KSPM** uses the `DEoptim()` function from the **DEoptim** package (Ardia et al., 2020) based on the differential evolution algorithm (Mullen et al., 2009). Given the random nature of the algorithm, it would be safe to apply the algorithm several times to ensure convergence toward the global minimum.

Tests of hypotheses in KSPM

For either a single kernel or a multiple kernel semi-parametric model, a standard test of interest is $H_0 : h_\ell(.) = 0$, i.e., there is no effect, singly or jointly, of a set of variables on the outcome. Following Liu et al. (2007), this test is equivalent to $H_0 : \tau_\ell = 0$, a test of the variance component in the linear mixed model (5) using the REML-based score test. The corresponding test statistic Q_ℓ follows a mixture of independent chi-square distributions with one degree of freedom (Zhang and Lin, 2003). The **KSPM** package uses exact distribution of Q_ℓ in single kernel model but in multiple kernel model, we use Davies' method to approximate this distribution (Davies, 1980). Based on similar methodology, **KSPM** also provides the global test for multiple kernel semi-parametric models $H_0 : h_1(.) = ... = h_L(.) = 0$ i.e., $H_0 : \tau_1 = ... = \tau_L = 0$.

Interpretation of variable effects

A kernel represents similarity between subjects through combining a set of variables in a possibly complex way, that may include their possible interactions. Hence the general effect of a kernel on an outcome is hard to see and difficult to interpret. Nevertheless, the marginal effect of each variable in the kernel may be illustrated using a partial derivative function (Hainmueller and Hazlett, 2013). Indeed, the derivative measures how the change in a variable of interest impacts the outcome in an intuitively understandable way. When a variable's effect is linear, the interpretation is straightforward since the derivative corresponds to standard slope (β) coefficients. In kernel semi-parametric models, we are simultaneously modeling a set of variables. Therefore, when exploring the effect of any one variable of interest, the other variables in the kernel must be taken into account. Thus, plotting pointwise derivatives of the prediction for each subject against the value of the variable of interest may help in interpreting the effect of this variable on the outcome. Although a variable-level summary statistic may be obtained by averaging the pointwise derivatives across subjects (Hainmueller and Hazlett, 2013), we did not implement this option in **KSPM** because the average can mask relevant variability. For example, when positive and negative derivatives occur for the same variable, the average may be zero.

The choice of kernel functions

In any kernel semi-parametric model, the smooth unknown function $h_\ell(.)$ is approximated using basis functions from $\mathcal{H}_{k_\ell}$. Since the inner product of the basis functions corresponds to the kernel function $k_\ell(.,.)$, the choice of the kernel determines the function space used to approximate $h_\ell(.)$. The **KSPM** package includes the most popular kernel functions, described below. The linear kernel function $k(Z_i, Z_j) = Z_i^\top Z_j$ assumes that variables have a linear effect on outcome. It generates a linear function space so that the kernel semi-parametric model leads to a penalized multiple linear model using an L^2 norm (equivalent to a ridge regression). The polynomial kernel function $k(Z_i, Z_j) = (\rho\, Z_i^\top Z_j + \gamma)^d$ assumes that d^{th}-order products of the variables have a linear effect on the outcome, and is equivalent to a d^{th}-order interaction model. The Gaussian kernel function $k(Z_i, Z_j) = exp(- \parallel Z_i - Z_j \parallel^2 /\rho)$ generates the infinite function space of radial basis functions. The sigmoid kernel function $k(Z_i, Z_j) = tanh(\rho\, Z_i^\top Z_j + \gamma)$ and the inverse quadratic kernel $k(Z_i, Z_j) = (\parallel Z_i - Z_j \parallel^2 + \gamma)^{-1/2}$ are also often cited in the literature. Finally, we propose also the equality kernel $k(Z_i, Z_j) = 1$ if $Z_i = Z_j$ and 0 otherwise. Of note, users can define their own kernel function in **KSPM** by providing the corresponding kernel matrix. Some kernel functions, such as the linear or polynomial kernels, make assumptions about the shape of the effect of the variables on the continuous outcome, whereas other kernels like the gaussian may, in theory, handle all types of effects, regardless of their complexity. Indeed, in contrast to the linear and polynomial kernels, the gaussian kernel function captures a kernel space of infinite size leading to higher flexibility for approximation of $h(.)$. If true effects are linear, the linear kernel and gaussian kernel should converge toward similar results. However, in practice, if sample size is low or noise is large, the gaussian kernel will tend to retain a sinusoidal shape to the fit even when the truth is linear.

Kernel functions often involve tuning parameters; above, the parameters ρ and γ were used to indicate these kernel specific parameters. In practice, these tuning parameters are usually chosen by the user. However, if user does not provide a value for these parameters, the **KSPM** package estimates them at the same time as the penalization parameter(s), by minimizing the mean sum of squares of LOOE. The choice of tuning parameters may strongly impact the results and modify the smoothness of the $\hat{h}(.)$ function leading to overfitting or underfitting. For example, with a ρ value that is too large, the Gaussian kernel tends to lose its non-linear properties, whereas with ρ too small, it is very sensitive to noise.

In general, the choice of tuning parameters may strongly impact the results With that in mind, sensitivity analyses may include a comparison of results obtained with different values for these parameters. Also, comparing models based on information criteria such as AIC and BIC may help to choose the kernel function and its tuning parameter(s).

Package presentation

The **KSPM** package provides an R interface to perform kernel semi-parametric models and is available from the comprehensive R archive network (CRAN) at https://CRAN.R-project.org/package=KSPM. The package is called through the main function `kspm()` taking data and model hyperparameters as inputs, fitting the model, and returning a model fit object of class `"kspm"`.

Fitting the kernel semi parametric model with `kspm`

The main function of the package, `kspm()`, can fit the single or multiple kernel semi-parametric models described in the earlier, as detailed below:

```
kspm(response,linear,kernel,data,...)
```

The argument `response` indicates a continuous outcome variable. It can be specified as a string corresponding to the column name associated with the response in the dataset provided in the `data` argument, as a vector of length n, or as a $n \times 1$ matrix containing the continuous values of the outcomes. Of note, `kspm` does not deal with multivariate outcomes, and if an $n \times r$ (> 1) matrix is provided, only the first column is used. Argument `linear` specifies the linear part of the model and could be either a formula, a vector of length n if only one variable is included in the linear part, or an $n \times p$ design matrix containing the p variables included in the linear part. By default, an intercept is added. To remove the intercept term, the user should use the formula specification and add the term `-1`, as usual. `kernel` specifies the kernel part of the model. Its argument should be a formula of `Kernel` object(s), described below. For a multiple kernel semi-parametric model, `Kernel` objects are separated by the usual signs `"+"`, `"*"` and `":"` to specify addition and interaction between kernels.

The `Kernel` object regroups all information about a kernel including the choice of kernel function and its parameters. It is specified using the `Kernel` function as follows:

```
Kernel(x,kernel.function,scale,rho,gamma,d)
```

Argument `x` represents either the variables included in the kernel part of the model, or a kernel Gram matrix. In the latter case, the user should specify `kernel.function = "gram.matrix"` and all other arguments are not used. When `x` represents the variables included in the kernel part, it may be specified as a formula, a vector, or a matrix, and `kernel.function` indicates which kernel function should be used (e.g., `"gaussian"`, `"polynomial"`, ...). `scale` is a boolean indicating if variables should be scaled before computing the gram matrix. The need for other arguments depends on the choice of kernel function: `rho` and `gamma` are tuning parameters and `d` is the highest order in a polynomial kernel function; these parameters correspond to the ρ, γ and d introduced earlier. It is worth noting that in a multiple kernel model, **KSPM** allows kernel objects to follow different formats.

Different options were considered for the interface with respect to specification of the linear and kernel parts of the model. We decided to use an interface with separate formulae for the two parts. This structure makes it straightforward to manage variables coming from different sources or data structures within the package. For example, genetic data or high dimensional genomic data are often provided in a matrix format, whereas other variables and phenotypes are saved in vectors or data frames with meaningful variable names. This diversity of data source and format cannot be handled by a unique formula. If data elements are assembled from different sources, they should include the same individuals or observations ($i = 1, .., n$), with identical ordering; if not, `kspm` will return an error. It is worth noting that the **KSPM** package does tolerate observations containing missing values, although these observations will be removed prior to model fitting.

The function `kspm` returns an object of class `"kspm"` with `summary()` and `plot()` commands available, the first giving estimates and p-values and the second displaying diagnostic plots.

Methods applicable to objects of the class `"kspm"`

An object of class `"kspm"` results from a kernel semi-parametric model fit. It contains obviously estimated coefficients, fitted values and residuals, but also information about kernels such as $n \times n$ kernel matrices, hyperparameters, penalization parameters.

The `"kspm"` object can be summarized or viewed using commands and methods very similar to those implemented in `"lm"` or `"glm"`.

```
> methods(class = "kspm")
 [1] case.names  coef          confint    cooks.distance  deviance
 [6] extractAIC  fitted        logLik     nobs            plot
[11] predict     print         residuals  rstandard       sigma
[16] summary     variable.names
see '?methods' for accessing help and source code
```

Predictions

A `predict()` command has been implemented for the class `"kspm"`.

`predict.kspm(object,newdata.linear,newdata.kernel,interval,level)`

where `object` refers to a `"kspm"` object. If a new dataset is not specified, `predict.kspm` will return the fitted values from the original data. If `predict.kspm` is applied to a new dataset, all variables used in the original model should be provided in the `newdata.linear` and `newdata.kernel` arguments. `newdata.linear` should be a data frame, a vector or design matrix of variables used in the linear part. `newdata.kernel` is a list containing data frames, vectors and/or design matrices for each kernel. Formats depend on the ones previously used in model specification as shown in Table 3. For simplicity, users may follow the list returned by the `info.kspm()` function.

kernel specifications	new data specification
formula of q variables	a `data.frame` with columns names corresponding to variables included in the formula. Number of rows is $n^\star$, number of columns is q
vector	a vector of length $n^\star$
design matrix $n \times q$	a matrix $n^\star \times q$
kernel matrix $n \times n$	a matrix $n^\star \times n$ where cell i, j represents the similarity between new subject i and j^{th} subject included in the model.

Table 3: How new data should be specified in `predict.kspm`, depending on original model specifications for $n^\star$ new subjects. The first column refers to how the kernel is specified in the current model. The second column refers to how the new data should be specified in the `predict.kspm` function.

In `predict.kspm`, `interval` can be either `"none"`, `"confidence"`, or `"prediction"` according to whether the user wants a confidence or prediction interval. The level of confidence/prediction interval is specified using the `level` argument. By default, `level = 0.95` is used.

Variable selection procedures for the single kernel semi-parametric model

A variable selection procedure algorithm has been implemented for the class `"kspm"`. It is similar to the `step()` command existing for other regression packages.

`stepKSPM(object,linear.lower,linear.upper,kernel.lower,kernel.lower,k,direction)`

As before, `object` refers to a `"kspm"` object. However, in contrast to the generality allowed for fitting a single kernel semi-parametric model, here the `Kernel` object should not be specified with the Gram matrix option. The arguments of `linear.lower`, `linear.upper`, `kernel.lower` and `kernel.lower` are formulae corresponding to the desired boundaries for the smallest and largest numbers of variables to be included. As is standard in many R packages, all variables in the `linear.lower` and `kernel.lower` formulae cannot be removed from the model and variables that are not in the `linear.upper` and `kernel.upper` formulae cannot be added to the model. The procedure to select variables is based on AIC or BIC depending on the value of `k`. If `k` is set to 2, AIC is used, if `k` is set to $ln(n)$, BIC is used instead. The `direction` argument may be `"forward"` for a forward selection, `"backward"` for a backward selection and `"both"` for a stepwise selection. Our package was implemented so that variable selection for the linear part and the kernel part of the model may be done simultaneously. However, it is also possible to perform the variable selection procedure on each part separately by giving the appropriate formula to `linear.lower`, `linear.upper`, `kernel.lower` and `kernel.lower`.

Of note, as for the standard `stepAIC()` function, this procedure can only be used on complete observations. Thus missing data should be removed by the user prior to calling `stepKSPM()` so that the number of observations is identical at each step.

Graphical tools

The plot() method has been implemented for "kspm" and "derivatives.kspm" objects. The former gives usual diagnostic plots including residual distribution, leverage, Cook's distance,... The latter gives interpretation plot from pointwise derivatives.

Example 1: the Movie ratings data

This first example illustrates the functions provided in **KSPM** included the model fit, the diagnotic plot, the interpetation plot based on pointwise derivatives, the test of interaction as well as the variable selection procedure.

The conventional and social media movies (CSM) dataset is available on the UCI machine learning repository (https://archive.ics.uci.edu/ml/index.php) as well as in **KSPM** and is fully described in Ahmed et al. (2015). It contains data on 232 movies from 2014 and 2015; the movies are described in term of conventional features (sequel, budget in USD, gross income in USD, number of screens in USA) as well as social media features (aggregate actor followers on Twitter, number of views, likes, dislikes, comments of movie trailer on Youtube, sentiment score) in the aim of predicting ratings. In all subsequent analyses, we used only the 187 entries without missing data.

```
> data("csm")
> head(csm)

  Year Ratings     Gross  Budget Screens Sequel Sentiment   Views Likes
1 2014     6.3      9130 4.0e+06      45      1         0 3280543  4632
2 2014     7.1 192000000 5.0e+07    3306      2         2  583289  3465
3 2014     6.2  30700000 2.8e+07    2872      1         0  304861   328
4 2014     6.3 106000000 1.1e+08    3470      2         0  452917  2429
5 2014     4.7  17300000 3.5e+06    2310      2         0 3145573 12163
7 2014     6.1  42600000 4.0e+07    3158      1         0 3013011  9595
  Dislikes Comments Aggregate.Followers
1      425      636             1120000
2       61      186            12350000
3       34       47              483000
4      132      590              568000
5      610     1082             1923800
7      419     1020             8153000
```

Predict ratings using conventional features

In our first model, we assume a gaussian kernel function to fit the joint effect of the conventional features on ratings. Here we do not provide any value for the ρ parameter. It will be estimated at the same time as the penalization parameter by minimizing the LOOE. We also do not provide a linear argument, meaning that only an intercept will be included in the linear part of the model.

```
> csm.fit1 <- kspm(response = "Ratings", kernel = ~Kernel(~Gross + Budget +
+    Screens + Sequel, kernel.function = "gaussian"), data = csm)
> summary(csm.fit1)

Call:
kspm(response = "Ratings", kernel = ~Kernel(~Gross + Budget +
    Screens + Sequel, kernel.function = "gaussian"),
    data = csm)

Sample size:
n = 187

Residuals:
    Min      Q1  Median      Q3     Max
-3.0066 -0.4815  0.0109  0.5534  2.1228

Coefficients (linear part):
            Estimate Std. Error  t value     Pr(>|t|)
(Intercept) 6.297723   1.058707 5.948505 1.427565e-08
```

```
Score test for non-parametric kernel:
         lambda      tau      p-value
Ker1 0.04804093 16.13793 5.625602e-06

Residual standard error: 0.88 on 175.82 effective degrees of freedom
Multiple R-squared: 0.2643, Adjusted R-squared: 0.2217
```

The summary output gives information about the sample size used in the model, the residual distribution, the coefficient for the linear part (similar to other regression R packages), and the penalization parameter and score test associated with the kernel part. The kernel results indicate that the conventional features are strongly associated with ratings. In such a complex model, the number of free parameters – i.e., the standard definition of the degrees of freedom of a model – is undefined, and we use instead the effective degrees of freedom of the model, which is not necessarily a whole number. However, our definition for effective degrees of freedom is similar to the one used in linear regression models and depends on the trace of the hat matrix. The ρ parameter may be extracted using the following code:

```
> csm.fit1$kernel.info$Ker1$rho

 par1
61.22
```

This value alone may not provide much information about linearity of the kernel function. However interpretation is feasible when comparing gaussian kernel functions with different ρ parameter values, or when comparing the gaussian and linear kernel functions.

The `plot()` command may be applied to the model to display diagnostic plots. Figure 1 has been generated using the following code:

```
> par(mfrow = c(2,2), mar = c(5, 5, 5, 2))
> plot(csm.fit1, which = c(1, 3, 5), cex.lab = 1.5, cex = 1.3, id.n = 7)
> hist(csm$Ratings, cex.lab = 1.5, main = "Histogram of Ratings", xlab =
+    "Ratings")
```

Outlier points are annotated and we can see movie 134 (*Jurassic World*) has high leverage. Also of note, the lower tail of the residuals distribution is not as expected for a Normal distribution. The histogram of ratings shows that the deviation could be due to the left skewed distribution of the response variable.

The derivative function may help to interpret the effect of each variable individually on the outcome. Indeed, the sign of the derivatives captures variational changes of the effect of the variable on the outcome. To illustrate this feature, Figure 2 displays the derivatives for Gross income, Budget and Screens. It has been generated with the following code:

```
> par(mfrow = c(1,2), mar = c(5,5,5,2))
> plot(derivatives(csm.fit1), subset = "Gross", cex.lab = 1.5, cex = 1.3,
+    main = "Pointwise derivatives according to Gross Income")
> plot(derivatives(csm.fit1), subset = "Screens", col = csm$Sequel,
+    cex.lab = 1.5, cex = 1.3, pch = 16, main = "Pointwise derivatives
+    according to Number of Screens \n and Sequel")
> legend("topleft", fill = palette()[1:7], legend = 1:7, title = "Sequel",
+    horiz = TRUE)
```

By default, the X-axis label gives the name of the variable and, in brackets, the kernel in which it is included. When only one kernel is included in the kernel, it is named `Ker1`. Because genre and sequel variables, even if they are numeric, refer to categorical variables, it is possible to easily highlight some patterns of interaction between these variables and the others. Derivatives according to Gross income are mostly positive meaning that higher Gross income is associated with higher ratings. However the slope of this effect decreases as the gross income grows. It is difficult to interpret the derivatives for Gross income higher than 2.5e+08 since this category includes only a few movies. Based on the display in the right panel, whether a movie has sequels seems to interact with the effect of the number of screens on the ratings. Indeed when the movie is the first or second release (Sequel = 1 or 2), the derivatives are always negative meaning that as the number of screens on which the movie is released increases, the ratings tend to decrease. However, this relationship seems to be stronger for the first release than the second. No clear pattern can be observed for subsequent sequels. It is difficult to know if this reveals an absence of effect or whether there are simply too few movies past sequel 2 in these data.

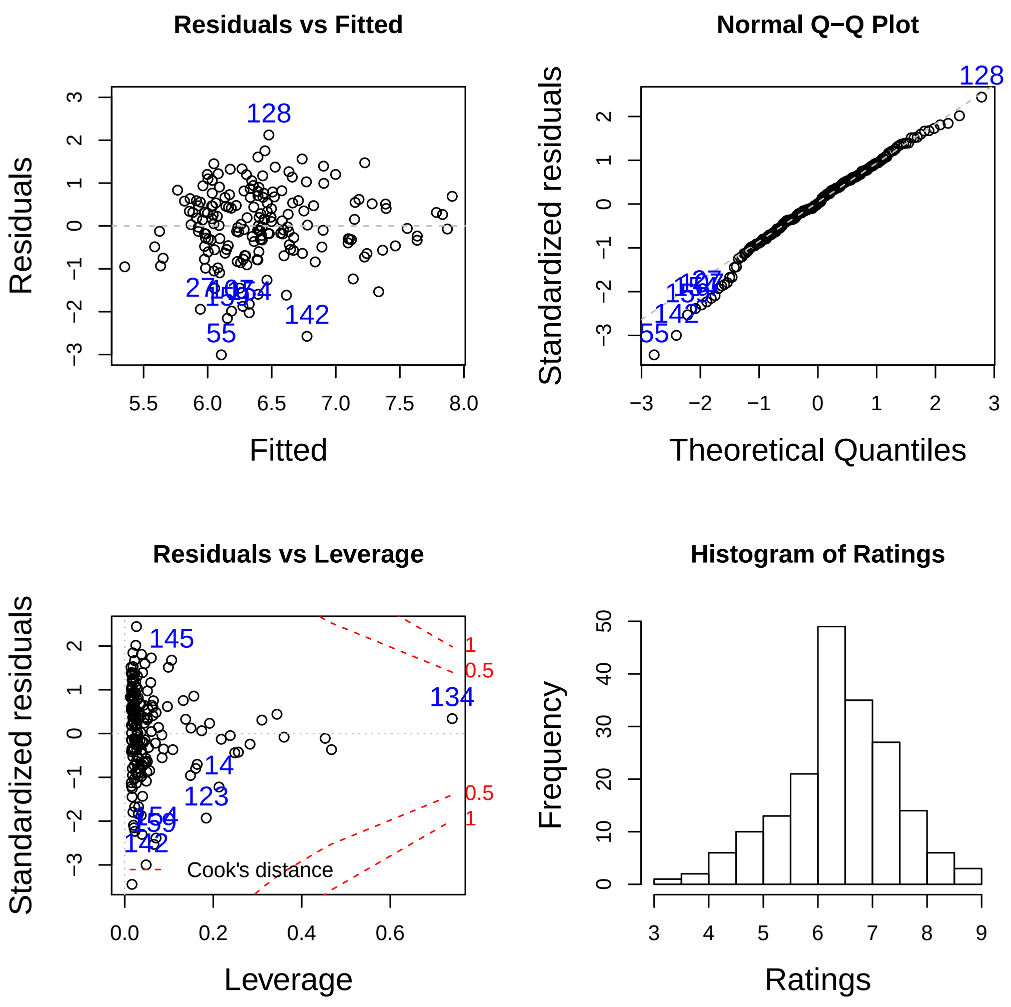


Figure 1: Diagnostic plots of CSM data. Plots at the top left, top right and bottom left were obtained with `plot.kspm`. They represent respectively residuals against fitted values, the normal Q-Q plot of residuals and residuals against leverage with the Cook's distance information. Plot at the bottom right represents the distribution of ratings in the database.

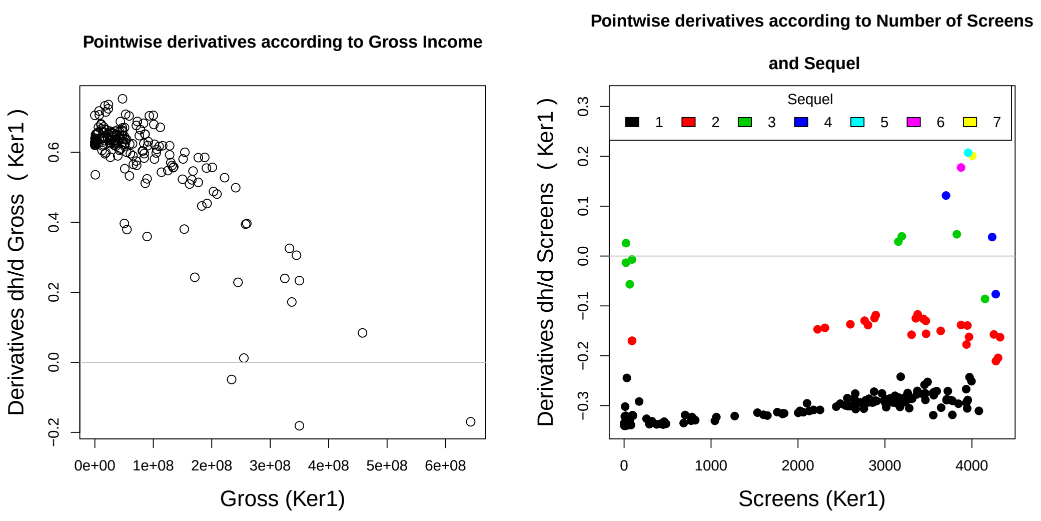


Figure 2: Derivative plots on CSM data obtained with `plot.derivatives`. Each point corresponds to an observation. Plot on the left represents the pointwise derivatives according to the Gross income variable. Plot on the right represents the pointwise derivatives according to the Screens variable and are colored according Sequel variable showing a probable interaction between Screens and Sequel.

To help in the choice of the kernel function, we may compare several models by using information criteria. As an example, we fit a second model assuming a polynomial kernel function with fixed tuning parameters ($\rho = 1, \gamma = 1$ and $d = 2$). This model can be compared to the previous one using information criteria such as the AIC or BIC. By default the `extractAIC()` command gives the AIC.

```
> csm.fit2 <- kspm(response = "Ratings", kernel = ~Kernel(~Gross + Budget +
+    Screens + Sequel, kernel.function = "polynomial", rho = 1, gamma = 1,
+    d = 2), data = csm, level = 0)
> extractAIC(csm.fit1)

[1] 941.4521

> extractAIC(csm.fit2)

[1] 944.4618
```

Here, we concluded that gaussian kernel function fits our data better than the polynomial kernel function, given the tuning parameters we considered.

Adding social media features to the model: a model with kernel interaction

Now, we assume a model with two kernel parts, one for conventional features and one for social media features, as well as their interaction. We propose to use the gaussian kernel function for each set of features, although different kernels could be used. The hyperparameters we chose are those obtained for each kernel separately.

```
> csm.fit3 <- kspm(response = "Ratings", linear = NULL, kernel = ~Kernel(~
+    Gross + Budget + Screens + Sequel, kernel.function = "gaussian",
+    rho = 61.22) * Kernel(~ Sentiment + Views + Likes + Dislikes + Comments +
+    Aggregate.Followers, kernel.function = "gaussian", rho = 1.562652),
+    data = csm)
```

While the model is running, R returns a summary of the kernel part(s) and interaction(s) included in the model.

```
------------------------------------------------------------------
The model includes the following kernels:
Ker1
Ker2
Ker1:Ker2
```

```
-----------------------------------------------------------------
Details:
Ker1: ~Kernel(~Genre + Gross + Budget + Screens + Sequel,
kernel.function = "gaussian", rho = 55.5897)
Ker2: ~Kernel(~Sentiment + Views + Likes + Dislikes + Comments +
Aggregate.Followers, kernel.function = "gaussian", rho = 1.562652)
-----------------------------------------------------------------
```

As defined by model (4), the `summary()` command will return the p value of tests $H_0 : h_1(.) = 0$, $H_0 : h_2(.) = 0$ and $H_0 : h_{12}(.) = 0$. By default all tests are performed. However, if our interest lies only in the test of interaction, the `kernel.test` option may be used to choose the test of interest and reduce the computation time. If interest lies in the global test $H_0 : h_1(.) = h_2(.) = h_{12}(.) = 0$, the `global.test` option should be set at `TRUE`.

```
> summary(csm.fit3, kernel.test = "Ker1:Ker2", global.test = TRUE)

Call:
kspm(response = "Ratings", linear = NULL, kernel = ~Kernel(~Gross +
    Budget + Screens + Sequel, kernel.function = "gaussian",
    rho = 61.22) * Kernel(~Sentiment + Views + Likes + Dislikes +
    Comments + Aggregate.Followers, kernel.function = "gaussian",
    rho = 1.562652), data = csm)

Sample size:
n = 187

Residuals:
    Min      Q1  Median      Q3     Max
-1.4185 -0.3396  0.0112  0.3291  1.3597

Coefficients (linear part):
            Estimate Std. Error  t value     Pr(>|t|)
(Intercept) 4.548524   1.256813 3.619095 0.0004324838

Score test for non-parametric kernel:
          lambda        tau   p-value
Ker1:Ker2    187 0.00208359 0.7376828

Global test: p-value = 6e-04

Residual standard error: 0.62 on 121.17 effective degrees of freedom
Multiple R-squared: 0.7452, Adjusted R-squared: 0.6089
```

Adding social media features to the model improved the predictions as indicated by the adjusted R^2. However the smooth function associated with the kernel interaction does not significantly differ from the null, leading to the conclusion that there is no interaction effect between conventional and social media features on the ratings.

Suppose now, we want to predict the ratings that will be attributed to the three artificial movies described in tables 4 and 5 below, according to the model `csm.fit3`.

	Gross	Budget	Screens	Sequel
Movie 1	5.0e+07	1.8e+08	3600	2
Movie 2	50000	5.2e+05	210	1
Movie 3	10000	1.3e+03	5050	1

Table 4: The conventional features of three artificial movies. Rows represent the new movies and columns represent the features.

```
> newdata.Ker1 <- data.frame(Gross = c(5.0e+07, 50000, 10000),
+    Budget = c(1.8e+08, 5.2e+05, 1.3e+03), Screens = c(3600, 210, 5050),
+    Sequel = c(2, 1, 1))
> newdata.Ker2 <- data.frame(Sentiment = c(1, 2, 10), Views = c(293021,
+    7206, 5692061), Likes = c(3698, 2047, 5025), Dislikes = c(768, 49,
```

	Sentiment	Views	Likes	Dislikes	Comments	Aggregate.Followers
Movie 1	1	293021	3698	768	336	4530000
Movie 2	2	7206	2047	49	70	350000
Movie 3	10	5692061	5025	305	150	960000

Table 5: The social media features of three artificial movies. Rows represent the new movies and columns represent the features.

```
+    305), Comments = c(336, 70, 150), Aggregate.Followers = c(4530000,
+    350000, 960000))
> predict(csm.fit3, newdata.kernel = list(Ker1 = newdata.Ker1, Ker2 =
+    newdata.Ker2), interval = "prediction")

       fit      lwr      upr
1 4.682560 3.147755 6.217365
2 6.401853 5.100309 7.703396
3 6.128641 4.395417 7.861864
```

The output of the `predict()` function gives the predicted values (`fit`) and the lower (`lwr`) and upper (`upr`) bounds of prediction intervals.

We may obtain the predictions for the original data directly from the model or from the `predict()` function. With the latter, confidence intervals may be additionally obtained.

```
> pred <- csm.fit3$fitted.value
> pred <- predict(csm.fit3, interval = "confidence")
> plot(csm$Ratings, pred$fit, xlim = c(2, 10), ylim = c(2, 10),
+    xlab = "Observed ratings", ylab = "Predicted ratings", cex.lab = 1.3)
> abline(a = 0, b = 1, col = "red", lty = 2)
```

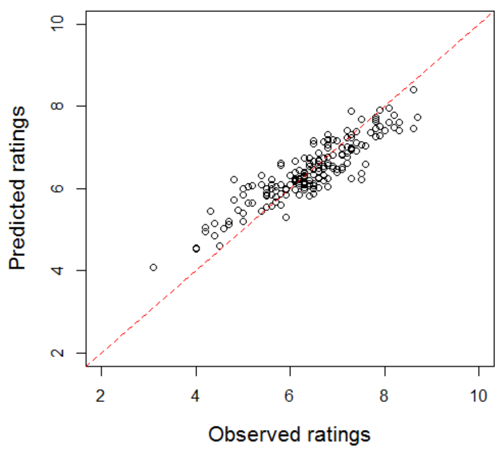


Figure 3: Predicted versus observed ratings in CSM dataset. Red dotted line represents a perfect concordance between predictions and observations.

Figure (3) shows that for smaller values, the model overestimates the outcome. This is again probably due to the left skewness of the outcome distribution.

An example of the variable selection procedure

Suppose we fit a single kernel semi-parametric model with a gaussian kernel to adjust the social media features in the CSM data. The kernel part contains the set of social media features. We want to select the relevant variables to be included in the kernel. We therefore can perform a stepwise variable selection procedure based on AIC, while letting ρ vary at each iteration. To do so, we first fit the full model including all features.

```
> csm.fit4 <- kspm(response = "Ratings", kernel = ~Kernel(~ Sentiment + Views
+    + Likes + Dislikes + Comments + Aggregate.Followers, kernel.function =
+    "gaussian"), data = csm)
```

Then, we apply the `stepKSPM()` command on the full model as follows.

```
> stepKSPM(csm.fit4, kernel.lower = ~1, kernel.upper = ~ Sentiment + Views
+    + Likes + Dislikes + Comments + Aggregate.Followers, direction = "both",
+    k = 2, kernel.param = "change", data = csm)
```

At each iteration, R returns the current model, and the list of variables that may be added or removed.

```
Start: AIC = 913.2
Linear part: ~ 1
Kernel part: ~ Sentiment + Views + Likes + Dislikes + Comments +
               Aggregate.Followers

                        Part      AIC
- Sentiment           kernel 910.8282
<none>                       913.1769
- Views               kernel 913.9753
- Likes               kernel 917.6532
- Comments            kernel 921.2163
- Aggregate.Followers kernel 925.4491
- Dislikes            kernel 969.4304


Step: AIC = 910.8
Linear part: ~ 1
Kernel part: ~ Views + Likes + Dislikes + Comments + Aggregate.Followers

                        Part      AIC
- Views               kernel 905.2908
<none>                       910.8282
+ Sentiment           kernel 913.1769
- Aggregate.Followers kernel 915.5481
- Likes               kernel 916.8125
- Comments            kernel 921.9627
- Dislikes            kernel 970.3804


Step: AIC = 905.3
Linear part: ~ 1
Kernel part: ~ Likes + Dislikes + Comments + Aggregate.Followers

                        Part      AIC
<none>                       905.2908
+ Views               kernel 910.8282
+ Sentiment           kernel 913.9753
- Aggregate.Followers kernel 916.0224
- Comments            kernel 917.8758
- Likes               kernel 925.0230
- Dislikes            kernel 968.2502
```

The final model includes the variables `Likes`, `Dislikes`, `Comments` and `Aggregate.Followers`.

Example 2: Consumption of energy data

Our second example illustrates how **KSPM** may be efficient with complex data as time series and show how the choice of tuning parameters impacts the results.

The energy data is a set of data on energy consumption each hour on Ouessant island (France) from September the 13th, 2015 to October the 4th, 2015, that were made publicly available by *Electricité de*

France at https://iles-ponant-edf-sei.opendatasoft.com. The data set also contains corresponding meteorologic data such as temperature (Celsius degrees), pressure (Pa) and humidity rate (g/m^3). These measures are collected by *Meteo France* every 3 hours and are publicly available on www.infoclimat.fr. We obtained hourly values by linear interpolation. In total the data set contains 504 measurements.

```
> data("energy")
> head(energy)

     power       date        T        P       HR hour hour.num
1 526.1667 2015-09-13 12.43333 1007.933 81.66667  01h        1
2 495.0000 2015-09-13 12.36667 1007.167 82.33333  02h        2
3 446.1667 2015-09-13 12.30000 1006.400 83.00000  03h        3
4 365.8333 2015-09-13 12.30000 1005.833 82.66667  04h        4
5 341.0000 2015-09-13 12.30000 1005.267 82.33333  05h        5
6 352.3333 2015-09-13 12.30000 1004.700 82.00000  06h        6
```

These data demonstrate strong periodicity depending on time of day (Figure 4). Of note, if data had been collected for a period longer than one year, a second periodicity would be visible corresponding to seasons.

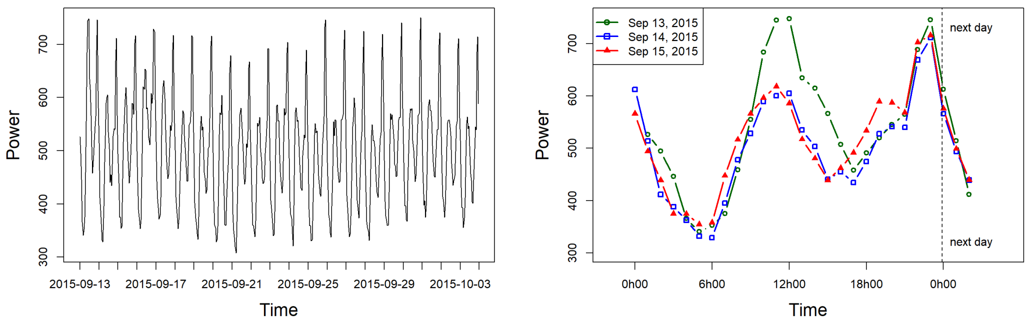


Figure 4: Pattern of energy consumption over the entire data set from Ouessant island (Left) and on the three first days (Right). The power is observed each hour and show a one day periodicity.

We will consider the 408 first measurements (17 days) as the training set. The others 4 days will be used as a test set, where we want to predict the energy consumption.

```
> energy_train_ <- energy[1:408, ]
> energy_test_ <- energy[409:504, ]
```

Modeling

We fit a single kernel semi-parametric model to the training data. We assume that energy depends linearly on temperature (T), and therefore this variable is included in the linear part of the model. The other meteorologic data, as well as hours in 24-hour numeric format, are included in the kernel part of the model. We used a gaussian kernel and we left the ρ parameter free to be estimated by the model.

```
> energy.fit1 <- kspm(response = "power", linear = ~T, kernel = ~Kernel(~
+    hour.num + P + HR, kernel.function = "gaussian") , data = energy_train_)
> energy.fit1$kernel.info$Ker1$rho

     par1
0.7028723
```

Impact of the ρ parameter on derivatives and predictions

We recomputed the model using other values for the ρ parameter to explore sensitivity of the results to this key kernel parameter. Values were chosen tenfold larger and smaller than the estimated value of 0.70 in energy.fit1.

```
> energy.fit2 <- kspm(response = "power", linear = ~T, kernel = ~Kernel(~
+    hour.num + P + HR, kernel.function = "gaussian", rho = 7) , data =
+    energy_train_)
> energy.fit3 <- kspm(response = "power", linear = ~T, kernel = ~Kernel(~
+    hour.num +  P + HR, kernel.function = "gaussian", rho = 0.07) , data =
+    energy_train_)
```

Figure 5 displays the predictions obtained on both the training and test data sets, as well as the derivatives, as a function of the hours variable, for the three models `energy.fit1`, `energy.fit2` and `energy.fit3` that differ only on the value of the tuning parameter ρ.

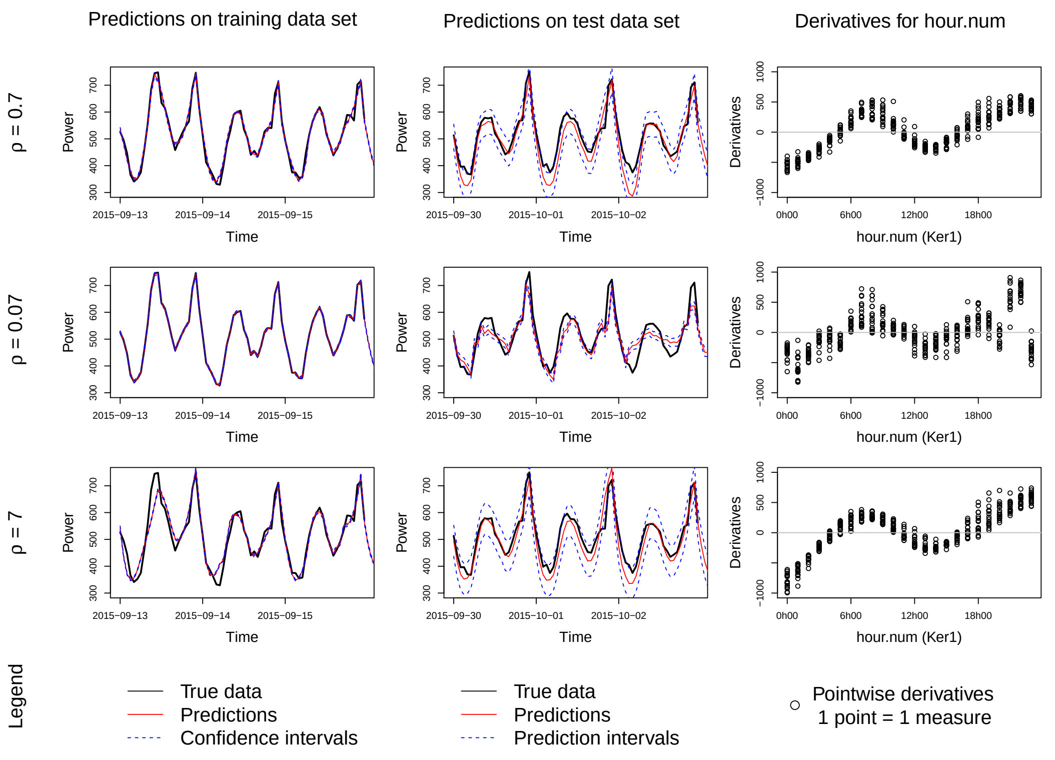


Figure 5: Predictions and derivatives obtained for different values of ρ in the energy dataset. Predictions on training data set are displayed with confidence intervals, whereas predictions on test data set are displayed with prediction intervals. The first row corresponds to the model `energy.fit1` with estimated $\rho = 0.7$. The second row corresponds to a tenfold smaller $\rho = 0.07$ (overfitting) and third row corresponds to a tenfold larger $\rho = 7$ (underfitting). The last column shows how the choice of ρ impacts the derivatives.

The first row of Figure 5 corresponds to the model with the value of ρ estimated by the model. Predictions fit well for both the training and the test sets. The derivative graph shows that between 6h00 and 12h00 and between 18h00 and 23h00 the derivatives are positive. This result is coherent with the increase of energy consumption during these times of the day (Figure 4-Right). Inversely between 0h00 and 6h00 and between 12h00 and 18h00, the energy consumption decreases as showed by negative values of derivatives. One might have expected that derivative values should be close between 0h00 and 23h00 but the peak consumption observed at 23h00 everyday (Figure 4) is probably associated with a sudden change in derivative values that is difficult to smooth accurately.

This example shows how a gaussian kernel function may handle complex functional data. However the choice of the ρ parameter may influence the results. A higher ρ leads to a smoother approximation of $h(.)$ and worsens the predictions. Indeed, a higher ρ may perfectly fit the training data set, but this corresponds to a case of model overfitting, which results in biased predictions in the test set. In contrast, a lower ρ may lead to underfitting of training and test data sets. The choice of the ρ parameter also impacts the derivatives; indeed a higher ρ induces more noise among the derivatives.

Example 3: Gene-gene interaction

This example shows how the interaction test may be applied to gene-gene interaction using standard linear kernel and SNPs involved in genes.

We simulated a data of 300 subjects and 9 SNPs, 6 belonging to gene A and 3 belonging to gene B, using the `glSim()` function into the **adegenet** R package (Jombart and Ahmed, 2011). SNPs are coded 0, 1 or 2 according to the number of alleles of each type carried by the subject. A continuous outcome was simulated as a linear combination of the interaction terms between the 6 SNPs of gene A and the three SNPs of gene B, where the weight coefficients where randomly and uniformly chosen in the interval $[-2; 0]$. We added an intercept of 100 and a normally distributed error of mean 0 and standard deviation 15. Code details are available in Supplement S1.

Let `y` be the vector of continuous outcomes, `geneA` be the 300 $\times$ 6 matrix of SNPs belonging to gene A and `geneB` be the 300 $\times$ 3 matrix of SNPs belonging to gene B.

We test the interaction between the genes using the code below. Of note, in this example, the data are specified using vector and design matrices instead of using formulae and data frame as in previous examples.

```
> gene.fit <- kspm(response = y, kernel = ~ Kernel(geneA, kernel = "linear")
+     * Kernel(geneB, kernel = "linear"))
> summary(gene.fit, kernel.test = "Ker1:Ker2")

Call:
kspm(response = y, kernel = ~Kernel(geneA, kernel = "linear") *
    Kernel(geneB, kernel = "linear"))

Sample size:
n = 300

Residuals:
     Min       Q1   Median       Q3      Max
-34.1111  -8.7300  -1.1659   9.7299  38.6887

Coefficients (linear part):
            Estimate Std. Error  t value      Pr(>|t|)
(Intercept) 88.80183  0.8178659 108.5775 2.531469e-231

Score test for non-parametric kernel:
            lambda       tau     p-value
Ker1:Ker2 199.4587 0.9862838 0.006646781

Residual standard error: 14.03 on 280.74 effective degrees of freedom
Multiple R-squared: 0.2387, Adjusted R-squared: 0.1892
```

The result suggests that genes A and B impact the continuous outcome y through their interaction.

Summary

This new **KSPM** package provides a flexible implementation of the kernel semi-parametric model and its extensions in a unified framework, using nomenclatures similar to other regression R packages. Thanks to the kernel trick, such a model is useful when interest lies in prediction and our `predict()` command makes this easy. Nevertheless, inference is also possible through the confidence intervals and tests provided by the package, such as through the `summary()` command. Moreover, we have provided many options for model diagnostics (residuals, leverage, ...), model interpretation (derivatives, ...) and model comparisons (AIC, stepwise, ...).

Model estimation for multiple kernel model is based on an iterative estimation of regression parameters we develop and prove in Supplement S2. Penalization and tuning parameter estimation involves `optimize()` and `DEoptim()` functions. The both integrate C code ensuring a faster convergence of the algorithms. However, the overall **KSPM** algorithm includes inversion of $n \times n$ matrices, thereby resulting in slower optimization as n increases. It would improve **KSPM**'s performance to combine matrix inversion code and optimization code in a single efficient set of C code and we are considering this for a future implementation. It is worth noting, however, that computation time is not severely impacted by the number of predictors even if $n << q$.

In the **KRLS** package, Hainmueller and Hazlett (2013) proposed a marginal test for the individual effect of each variable included in the kernel. We decided not to implement this test in our package since the marginal effect may mask interesting but symmetric effects of a variable on the continuous outcome. However, we have implemented a test of hypothesis based on variance components to test the joint effect of a set of variables on the continuous outcome. The distribution of this test statistic follows a mixture of χ^2 distributions that we approximated using Davies' method (Davies, 1980), available through the **CompQuadForm** package (de Micheaux, 2017; Duchesne and de Micheaux, 2010). The distribution can also be approximated by a scaled chi-squared distribution using Satterthwaite's method where parameters are computed by moment matching. Schifano et al. (2012) compared the two approximation methods on the type-I error rate in a single kernel semi-parametric model and showed that the rate is inflated with Satterthwaite's method when the α-level is low, and therefore we recommend the Davies method. To assess global testing of multiple kernels, we implemented a sum of the L single-kernel test statistics as an overall test statistic. In general, the computation of the p value of this overall test involves the joint distribution of (Q_1, ... Q_L). To derive analytical values for the overall p value, one can use techniques similar to those used in Sun et al. (2019), which relied on a copula-based model to approximate this joint distribution. This last option is not yet implemented in **KSPM**.

For all our tests, the model has to be re-estimated under the null hypothesis for the kernel of interest. If the resulting null model still contains one or more kernel part(s), we made the choice to recompute penalization parameter(s), since this choice ensures an optimized model under the null hypothesis. However, we have also decided to leave the kernel tuning parameters fixed when re-estimating under the null. Indeed, a change in tuning parameters induces a change in the choice of kernel functions - and thus in model assumptions - and hence, keeping the tuning parameters fixed ensures comparability of model assumptions under the null and the alternative hypotheses.

In our **KSPM** package, we have also included an algorithm for selection of variables based on these information criteria. Backward, forward or stepwise approaches can be chosen for single kernel semi-parametric models. Although these concepts may be easily extended to the case of multiple kernel models, such analyses require large computing times. Parallelization of such a process may greatly increase the appeal of this procedure. For now, we recommend investigating variables within a single kernel before starting to fit multiple kernel models.

In summary, the **KSPM** package is a flexible comprehensible option for studying the effects of groups of variables on a continuous outcome. These tools may be extended to the case of binary or survival outcomes in future work.

Acknowledgements

CS was supported by the Institute for Data Valorization (IVADO) fellowship. CG was supported by the Ludmer Centre for Neuroinformatics and Mental Health, the Canadian Institutes for Health Research, as well as the CIHR PJT 148620 grant. SJ is a recipient of a Canada Research Chair in neurodevelopmental disorders, and a chair from the Jeanne et Jean Louis Levesque Foundation. This research was enabled in part by support provided by Calcul Quebec (http://www.calculquebec.ca) and Compute Canada (www.computecanada.ca). This work is supported by a grant from the Brain Canada Multi-Investigator initiative and CIHR grant 159734 (SJ, CMTG). This work was also supported by an NIH award U01 MH119690 granted (SJ) and U01 MH119739.

Bibliography

M. Ahmed, M. Jahangir, H. Afzal, A. Majeed, and I. Siddiqi. Using crowd-source based features from social media and conventional features to predict the movies popularity. In *IEEE International Conference on Smart City/SocialCom/SustainCom (SmartCity)*, pages 273–278. IEEE, 2015. URL `https://doi.org/10.1109/SmartCity.2015.83`. [p89]

D. Ardia, K. Mullen, B. Peterson, and J. Ulrich. *DEoptim: Global Optimization by Differential Evolution*, 2020. URL `https://CRAN.R-project.org/package=DEoptim`. R package version 2.2-5. [p85]

D. Bates, M. Maechler, B. Bolker, and S. Walker. *lme4: Linear Mixed-Effects Models using 'Eigen' and S4*, 2019. URL `https://CRAN.R-project.org/package=lme4`. R package version 1.1-21. [p84]

R. P. Brent. *Algorithms for Minimization without Derivatives*. Prentice-Hall, Englewood Clifts, New Jersey, 2013. [p85]

J. Chen, W. Chen, N. Zhao, M. C. Wu, and D. J. Schaid. Small sample kernel association tests for human genetic and microbiome association studies. *Genetic epidemiology*, 40(1):5–19, 2016. URL `https://doi.org/10.1002/gepi.21934`. [p83]

R. B. Davies. Distribution of a linear combination of chi-square random variables: Algorithm as 155. *Applied Statistics*, 29(3):323–33, 1980. [p86, 99]

P. L. de Micheaux. *CompQuadForm: Distribution Function of Quadratic Forms in Normal Variables*, 2017. URL https://CRAN.R-project.org/package=CompQuadForm. R package version 1.4.3. [p99]

P. Duchesne and P. L. de Micheaux. Computing the distribution of quadratic forms: Further comparisons between the Liu-Tang-Zhang approximation and exact methods. *Computational Statistics and Data Analysis*, 54:858–862, 2010. URL https://doi.org/10.1016/j.csda.2009.11.025. [p99]

T. Ge, T. E. Nichols, D. Ghosh, E. C. Mormino, J. W. Smoller, M. R. Sabuncu, and the Alzheimer's Disease Neuroimaging Initiative. A kernel machine method for detecting effects of interaction between multidimensional variable sets: An imaging genetics application. *Neuroimage*, 109:505–514, 2015. URL https://doi.org/10.1016/j.neuroimage.2015.01.029. [p83, 85]

J. Hainmueller and C. Hazlett. Kernel regularized least squares: Reducing misspecification bias with a flexible and interpretable machine learning approach. *Political Analysis*, 22(2):143–168, 2013. [p83, 86, 98]

J. Hainmueller and C. Hazlett. *KRLS: Kernel-Based Regularized Least Squares*, 2017. URL https://CRAN.R-project.org/package=KRLS. R package version 1.0-0. [p83]

T. Jombart and I. Ahmed. Adegenet 1.3-1: New tools for the analysis of genome-wide SNP data. *Bioinformatics*, 2011. URL https://doi.org/10.1093/bioinformatics/btr521. [p98]

G. Kimeldorf and G. Wahba. Some results on Tchebycheffian spline functions. *Journal of mathematical analysis and applications*, 33(1):82–95, 1971. URL https://doi.org/10.1016/0022-247X(71)90184-3. [p85]

S. Lee, with contributions from Larisa Miropolsky, and M. Wu. *SKAT: SNP-Set (Sequence) Kernel Association Test*, 2017. URL https://CRAN.R-project.org/package=SKAT. R package version 1.3.2.1. [p83]

S. Li and Y. Cui. *SPA3G: R Package for the Method of Li and Cui (2012)*, 2012a. URL https://CRAN.R-project.org/package=SPA3G. R package version 1.0. [p83]

S. Li and Y. Cui. Gene-centric gene-gene interaction: A model-based kernel machine method. *The Annals of Applied Statistics*, 6(3):1134–1161, 2012b. URL https://doi.org/10.1214/12-AOAS545. [p83]

D. Liu, X. Lin, and D. Ghosh. Semiparametric regression of multidimensional genetic pathway data: Least-squares kernel machines and linear mixed models. *Biometrics*, 63(4):1079–1088, 2007. URL https://doi.org/10.1111/j.1541-0420.2007.00799.x. [p82, 85, 86, 105]

R. Marceau, W. Lu, S. Holloway, M. M. Sale, B. B. Worrall, S. R. Williams, F.-C. Hsu, and J.-Y. Tzeng. A fast multiple-kernel method with applications to detect gene-environment interaction. *Genetic epidemiology*, 39(6):456–468, 2015. URL https://doi.org/10.1002/gepi.21909. [p83]

D. Meyer, E. Dimitriadou, K. Hornik, A. Weingessel, and F. Leisch. *e1071: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien*, 2018. URL https://CRAN.R-project.org/package=e1071. R package version 1.7-0. [p83]

K. M. Mullen, D. Ardia, D. L. Gil, D. Windover, and J. Cline. Deoptim: An R package for global optimization by differential evolution. *Journal of Statistical Software*, 40(6):1–26, 2009. URL http://www.jstatsoft.org/v40/i06/. [p85]

K. Oualkacha, Z. Dastani, R. Li, P. E. Cingolani, T. D. Spector, C. J. Hammond, J. B. Richards, A. Ciampi, and C. M. Greenwood. Adjusted sequence kernel association test for rare variants controlling for cryptic and family relatedness. *Genetic epidemiology*, 37(4):366–376, 2013. URL https://doi.org/10.1002/gepi.21725. [p83]

J. Pinheiro, D. Bates, and R-core. *nlme: Linear and Nonlinear Mixed Effects Models*, 2019. URL https://CRAN.R-project.org/package=nlme. R package version 3.1-140. [p84]

J. S. Racine and T. Hayfield. *np: Nonparametric Kernel Smoothing Methods for Mixed Data Types*, 2020. URL https://CRAN.R-project.org/package=np. R package version 0.60-10. [p84]

E. D. Schifano, M. P. Epstein, L. F. Bielak, M. A. Jhun, S. L. Kardia, P. A. Peyser, and X. Lin. SNP set association analysis for familial data. *Genetic epidemiology*, 36(8):797–810, 2012. URL https://doi.org/10.1002/gepi.21676. [p99]

C. Schramm. *KSPM: Kernel Semi-Parametric Models*, 2020. URL https://CRAN.R-project.org/package=KSPM. R package version 0.2.1. [p83]

J. Sun, K. Oualkacha, C. M. Greenwood, and L. Lakhal-Chaieb. Multivariate association test for rare variants controlling for cryptic and family relatedness. *Canadian Journal of Statistics*, 47(1):90–107, 2019. URL https://doi.org/10.1002/cjs.11475. [p99]

T. M. Therneau. *coxme: Mixed Effects Cox Models*, 2018. URL https://CRAN.R-project.org/package=coxme. R package version 2.2-10. [p83]

C. Wang, J. Sun, B. Guillaume, T. Ge, D. P. Hibar, C. M. Greenwood, A. Qiu, and the Alzheimer's Disease Neuroimaging Initiative. A set-based mixed effect model for gene-environment interaction and its application to neuroimaging phenotypes. *Frontiers in neuroscience*, 11:191, 2017. URL https://doi.org/10.3389/fnins.2017.00191. [p83]

S. Wood. *mgcv: Mixed GAM Computation Vehicle with Automatic Smoothness Estimation*, 2020. URL https://CRAN.R-project.org/package=mgcv. R package version 1.8-33. [p84]

M. C. Wu, S. Lee, T. Cai, Y. Li, M. Boehnke, and X. Lin. Rare-variant association testing for sequencing data with the sequence kernel association test. *The American Journal of Human Genetics*, 89(1):82–93, 2011. URL https://doi.org/10.1016/j.ajhg.2011.05.029. [p83]

D. Zhang and X. Lin. Hypothesis testing in semiparametric additive mixed models. *Biostatistics*, 4(1): 57–74, 2003. URL https://doi.org/10.1093/biostatistics/4.1.57. [p86]

SUPPLEMENT

Supplement S1

We simulated a 300×9 matrix representing the number of minor allele (0, 1 or 2) of 300 subjects for 9 SNPs, 6 belonging to gene A and 3 belonging to gene B, using the `glSim()` function into the **adegenet** R package.

```
> library(adegenet)
> set.seed(78)
> SNPdata <- as.matrix(glSim(n.ind = 300, n.snp.nonstruc = 9, n.snp.struc = 0, k = 1,
  LD = FALSE, ploidy = 2))
```

We may deduce two matrices, one with data belonging to gene A and one with data belonging to gene B.

```
> geneA <- SNPdata[,1:6]
> geneB <- SNPdata[,7:9]
```

Then we computed the matrix of interactions between SNPs from gene A and SNPs from geneB.

```
> SNPdataAB <- matrix(NA, nrow = 300, ncol = 6*3)
> k <- 1
> for (i in 1:6) {
    for (j in 1:3) {
      SNPdataAB[,k] <- SNPdataA[,i] * SNPdataB[,j]
      k <- k+1
    }
  }
```

A continuous outcome was simulated as a linear combination of the interaction terms between the 6 SNPs of gene A and the three SNPs of gene B, where the weight coefficients where randomly and uniformly chosen in the interval $[-2;0]$ using the following code.

```
> beta <- as.matrix(runif(6*3,-2,0), nrow = 300)
> y <- 100 + SNPdataAB %*% beta + rnorm(n, 0, 15)
```

Of note, we added an intercept of 100 and a normally distributed error of mean 0 and standard deviation 15.

Supplement S2

Estimating the kernel semi-parametric model parameters consists in estimating $\alpha_1, ..., \alpha_L$ and β. Then, estimators of $h_1(.), ..., h_L(.)$ are deduced from $\hat{\alpha}_1, ..., \hat{\alpha}_L$. When there are multiple kernels, estimation is iterative and so we proceed as follows. Partial derivatives of $l(\beta, h)$ according to $\alpha_1, ..., \alpha_L$ and β lead to the following equations:

$$\hat{\beta} = \left\{ X^\top \left(I - \sum_{\ell=1}^{L} K_\ell G_\ell^{-1} M_\ell \right) X \right\}^{-1} X^\top \left(I - \sum_{\ell=1}^{L} K_\ell G_\ell^{-1} M_\ell \right) Y \quad (9)$$

$$\forall \ell \in \{1, ..., L\}, \quad \hat{\alpha}_\ell = G_\ell^{-1} M_\ell \left(Y - X\hat{\beta} \right) \quad (10)$$

where I is the $n \times n$ identity matrix and $\forall \ell \in \{1, ..., L\}$, G_ℓ and M_ℓ are computed using an iterative process as described below:

- Re-order elements $\alpha_{1...L}$, $K_{1...L}$ and $\lambda_{1...L}$ such that the last is the ℓ^{th} element
- Note $\alpha_{(1)}$, ..., $\alpha_{(L)}$, $K_{(1)}$, ..., $K_{(L)}$ and $\lambda_{(1)}$, ..., $\lambda_{(L)}$ the re-ordered elements with $\alpha_{(L)} = \alpha_\ell$, $K_{(L)} = K_\ell$ and $\lambda_{(L)} = \lambda_\ell$
- Define iteratively $\mathcal{M}_{(0)...(L)}$ and $\mathcal{G}_{(1)...(L)}$ by $\mathcal{M}_{(0)} = I$ the $n \times n$ identity matrix and $\forall m \in \{1, ..., L\}$:
  $\mathcal{G}_{(m)} = \lambda_{(m)} I + \mathcal{M}_{(m-1)} K_{(m)}$
  $\mathcal{M}_{(m)} = \left(I - \mathcal{M}_{(m-1)} K_{(m)} \mathcal{G}_{(m)}^{-1} \right) \mathcal{M}_{(m-1)}$

- Compute G_ℓ and M_ℓ as:
  $G_\ell = \mathcal{G}_{(L)}$
  $M_\ell = \mathcal{M}_{(L-1)}$

Whereas the parameter estimation for single kernel model has been largely demonstrated, the equations for computing multiple kernel parameters were not theoretically developed yet. Above, we propose an iterative way to estimate the parameters for all kernel models. Below, we propose the proof for equations (9) and (10).

Proof:

Partial derivatives of penalized likelihood (6) lead to equations below:

$$\begin{cases} X^\top(Y - X\beta - \sum_{\ell=1}^{L} K_\ell\alpha_\ell) = 0 \\ Y - X\beta - \sum_{m=1}^{L} K_m\alpha_m - \lambda_\ell\alpha_\ell = 0 \quad \forall \ell \in \{1,...,L\} \end{cases} \tag{11}$$

Proposition 1: $\forall k \in \{1,...,L-1\}\,, \alpha_{(k)} = \mathcal{G}_{(k)}^{-1}\mathcal{M}_{(k-1)}\left(Y - X\beta - \sum_{m=k+1}^{L} K_{(m)}\alpha_{(m)}\right)$

Proposition 2: if proposition 1 is true until k, then

$$\sum_{j=1}^{k} K_{(m)}\alpha_{(m)} = (I - \mathcal{M}_{(k)})\left(Y - X\beta - \sum_{m=k+1}^{L} K_{(m)}\alpha_{(m)}\right)$$

(Proofs of propositions 1 and 2 are given below.)

Now, suppose proposition 1 and proposition 2 are true.

$$\begin{aligned} (11) \Rightarrow &\quad Y - X\beta - \sum_{m=1}^{L} K_{(m)}\alpha_{(m)} - \lambda_{(L)}\alpha_{(L)} = 0 \\ \Leftrightarrow &\quad Y - X\beta - \sum_{m=1}^{L-1} K_{(m)}\alpha_{(m)} - K_{(L)}\alpha_{(L)} - \lambda_{(L)}\alpha_{(L)} = 0 \\ \Leftrightarrow &\quad Y - X\beta - (I - \mathcal{M}_{(L-1)})\left(Y - X\beta - K_{(L)}\alpha_{(L)}\right) - K_{(L)}\alpha_{(L)} - \lambda_{(L)}\alpha_{(L)} = 0 \\ \Leftrightarrow &\quad \left(\lambda_{(L)}I + \mathcal{M}_{(L-1)}K_{(L)}\right)\alpha_{(L)} - \mathcal{M}_{(L-1)}(Y - X\beta) = 0 \end{aligned}$$

Then, $\alpha_\ell = \alpha_{(L)} = \mathcal{G}_{(L)}^{-1}\mathcal{M}_{(L-1)}\,(Y - X\beta)$. Let G_ℓ and M_ℓ the $n \times n$ matrix defined as $G_\ell = \mathcal{G}_{(L)}$ and $M_\ell = \mathcal{M}_{(L-1)}$, thus $\alpha_\ell = G_\ell^{-1}M_\ell\,(Y - X\beta)$. All this process is done for each $\ell \in \{1,...,L\}$.

Now, we can derive the estimate for β.

$$\begin{aligned} (11) \Rightarrow &\quad X^\top\left(Y - X\beta - \sum_{\ell=1}^{L} K_\ell\alpha_\ell\right) = 0 \\ \Leftrightarrow &\quad X^\top\left(Y - X\beta - \sum_{\ell=1}^{L} K_\ell G_\ell^{-1}M_\ell(Y - X\beta)\right) = 0 \\ \Leftrightarrow &\quad X^\top\left(I - \sum_{\ell=1}^{L} K_\ell G_\ell^{-1}M_\ell\right)Y = X^\top\left(I - \sum_{\ell=1}^{L} K_\ell G_\ell^{-1}M_\ell\right)X\beta \end{aligned}$$

Thus $\beta = \left\{ X^\top \left(I - \sum_{\ell=1}^{L} K_\ell G_\ell^{-1} M_\ell \right) X \right\}^{-1} X^\top \left(I - \sum_{\ell=1}^{L} K_\ell G_\ell^{-1} M_\ell \right) Y.$

Proof of proposition 1:

• Proposition 1 is true for $k = 1$:

$$\begin{aligned}
(11) \Rightarrow \quad & Y - X\beta - \sum_{m=1}^{L} K_{(m)}\alpha_{(m)} - \lambda_{(1)}\alpha_{(1)} = 0 \\
\Leftrightarrow \quad & Y - X\beta - \sum_{m=2}^{L} K_{(m)}\alpha_{(m)} - K_{(1)}\alpha_{(1)} - \lambda_{(1)}\alpha_{(1)} = 0 \\
\Leftrightarrow \quad & \mathcal{M}_{(0)} \left(Y - X\beta - \sum_{m=2}^{L} K_{(m)}\alpha_{(m)} \right) = \left(\lambda_{(1)} I + \mathcal{M}_{(0)} K_{(1)} \right) \alpha_{(1)} \\
\Leftrightarrow \quad & \mathcal{G}_{(1)}^{-1} \mathcal{M}_{(0)} \left(Y - X\beta - \sum_{m=2}^{L} K_{(m)}\alpha_{(m)} \right) = \alpha_{(1)}
\end{aligned}$$

• If proposition 1 is true for k, then proposition 1 is true for $k + 1$:

According to proposition 2:

$$\begin{aligned}
\sum_{m=1}^{k} K_{(m)}\alpha_{(m)} &= (I - \mathcal{M}_{(k)}) \left(Y - X\beta - \sum_{m=k+1}^{L} K_{(m)}\alpha_{(m)} \right) \\
&= (I - \mathcal{M}_{(k)}) \left(Y - X\beta - \sum_{m=k+2}^{L} K_{(m)}\alpha_{(m)} \right) - (I - \mathcal{M}_{(k)}) K_{(k+1)}\alpha_{(k+1)}
\end{aligned}$$

$$\begin{aligned}
(11) \Rightarrow \quad & Y - X\beta - \sum_{m=1}^{L} K_{(m)}\alpha_{(m)} - \lambda_{(k+1)}\alpha_{(k+1)} = 0 \\
\Leftrightarrow \quad & Y - X\beta - \sum_{m=1}^{k} K_{(m)}\alpha_{(m)} - K_{(k+1)}\alpha_{(k+1)} - \sum_{m=k+2}^{L} K_{(m)}\alpha_{(m)} - \lambda_{(k+1)}\alpha_{(k+1)} = 0 \\
\Leftrightarrow \quad & - \left((I - \mathcal{M}_{(k)}) \left(Y - X\beta - \sum_{m=k+2}^{L} K_{(m)}\alpha_{(m)} \right) - (I - \mathcal{M}_{(k)}) K_{(k+1)}\alpha_{(k+1)} \right) \\
& + Y - X\beta - K_{(k+1)}\alpha_{(k+1)} - \sum_{m=k+2}^{L} K_{(m)}\alpha_{(m)} - \lambda_{(k+1)}\alpha_{(k+1)} = 0 \\
\Leftrightarrow \quad & \mathcal{M}_{(k)} \left(Y - X\beta - \sum_{m=k+2}^{L} K_{(m)}\alpha_{(m)} \right) - \mathcal{M}_{(k)} K_{(k+1)}\alpha_{(k+1)} - \lambda_{(k+1)}\alpha_{(k+1)} = 0 \\
\Leftrightarrow \quad & \mathcal{M}_{(k)} \left(Y - X\beta - \sum_{m=k+2}^{L} K_{(m)}\alpha_{(m)} \right) - \mathcal{G}_{(k+1)}\alpha_{(k+1)} = 0 \\
\Leftrightarrow \quad & \mathcal{G}_{(k+1)}^{-1} \mathcal{M}_{(k)} \left(Y - X\beta - \sum_{m=k+2}^{L} K_{(m)}\alpha_{(m)} \right) = \alpha_{(k+1)}
\end{aligned}$$

Proof of proposition 2:

• Proposition 2 is true for $k = 1$:

Suppose $\alpha_{(1)} = \mathcal{G}_{(1)}^{-1} \mathcal{M}_{(0)} \left(Y - X\beta - \sum_{m=2}^{L} K_{(m)}\alpha_{(m)} \right)$.

$$\begin{aligned}
\sum_{m=1}^{1} K_{(m)}\alpha_{(m)} &= K_{(1)}\alpha_{(1)} \\
&= K_{(1)}\mathcal{G}_{(1)}^{-1}\mathcal{M}_{(0)}\left(Y - X\beta - \sum_{m=2}^{L} K_{(m)}\alpha_{(m)}\right) \\
&= \left(I - \mathcal{M}_{(0)} + \mathcal{M}_{(0)}K_{(1)}G_1^{-1}\mathcal{M}_{(0)}\right)\left(Y - X\beta - \sum_{m=2}^{L} K_{(m)}\alpha_{(m)}\right) \\
&= \left(I - (I - \mathcal{M}_{(0)}K_{(1)}G_1^{-1})\mathcal{M}_{(0)}\right)\left(Y - X\beta - \sum_{m=2}^{L} K_{(m)}\alpha_{(m)}\right) \\
&= \left(I - \mathcal{M}_{(1)}\right)\left(Y - X\beta - \sum_{m=2}^{L} K_{(m)}\alpha_{(m)}\right)
\end{aligned}$$

• If proposition 2 is true for k, then proposition 2 is true for $k+1$:

Suppose proposition 1 is true until $k+1$, then $\forall j \in \{1, ..., k+1\}$,

$\alpha_{(j)} = \mathcal{G}_{(j)}^{-1}\mathcal{M}_{(j-1)}\left(Y - X\beta - \sum_{m=j+1}^{L} K_{(m)}\alpha_{(m)}\right)$

$$\begin{aligned}
\sum_{m=1}^{k+1} K_{(m)}\alpha_{(m)} &= \sum_{m=1}^{k} K_{(m)}\alpha_{(m)} + K_{(k+1)}\alpha_{(k+1)} \\
&= (I - \mathcal{M}_{(k)})\left(Y - X\beta - \sum_{m=k+1}^{L} K_{(m)}\alpha_{(m)}\right) + K_{(k+1)}\alpha_{(k+1)} \\
&= (I - \mathcal{M}_{(k)})\left(Y - X\beta - \sum_{m=k+2}^{L} K_{(m)}\alpha_{(m)}\right) + \mathcal{M}_{(k)}K_{(k+1)}\alpha_{(k+1)} \\
&= (I - \mathcal{M}_{(k)})\left(Y - X\beta - \sum_{m=k+2}^{L} K_{(m)}\alpha_{(m)}\right) + \mathcal{M}_{(k)}K_{(k+1)}\mathcal{G}_{(k+1)}^{-1}\mathcal{M}_{(k)}\left(Y - X\beta - \sum_{m=k+2}^{L} K_{(m)}\alpha_{(m)}\right) \\
&= (I - \mathcal{M}_{(k)} + \mathcal{M}_{k}K_{(k+1)}\mathcal{G}_{(k+1)}^{-1}\mathcal{M}_{k})\left(Y - X\beta - \sum_{m=k+2}^{L} K_{(m)}\alpha_{(m)}\right) \\
&= (I - (I + \mathcal{M}_{(k)}K_{(k+1)}\mathcal{G}_{(k+1)}^{-1})\mathcal{M}_{(k)})\left(Y - X\beta - \sum_{m=k+2}^{L} K_{(m)}\alpha_{(m)}\right) \\
&= (I - \mathcal{M}_{(k+1)})\left(Y - X\beta - \sum_{m=k+2}^{L} K_{(m)}\alpha_{(m)}\right)
\end{aligned}$$

$\square$

Of note, for $L = 1$, equations (9) and (10) correspond to those provided in Liu et al. (2007).

Catherine Schramm
Research center, Ste Justine Hospital
Montreal university
Lady Davis Institute for Medical Research, Jewish General Hospital
Montreal
Canada
ORCID: 0000-0002-1185-8809
`cath.schramm@gmail.com`

Sébastien Jacquemont
Research center, Ste Justine Hospital

Montreal university
Montreal
Canada
ORCID: 0000-0001-6838-8767
sebastien.jacquemont@umontreal.ca

Karim Oualkacha
Department of Mathematics
Université du Quebec à Montreal
Canada
ORCID: 0000-0002-9911-079X
oualkacha.karim@uqam.ca

Aurélie Labbe
Department of decision sciences
HEC Montreal
Canada
aurelie.labbe@hec.ca

Celia M.T. Greenwood
Lady Davis Institute for Medical Research, Jewish General Hospital
Gerald Bronfman Department of Oncology, Department of Epidemiology, Biostatistics and Occupational Health, and Department of Human Genetics, McGill University
Montreal
Canada
ORCID: 0000-0002-2427-5696
celia.greenwood@mcgill.ca

Comparing Multiple Survival Functions with Crossing Hazards in R

by Hsin-wen Chang, Pei-Yuan Tsai, Jen-Tse Kao and Guo-You Lan

Abstract It is frequently of interest in time-to-event analysis to compare multiple survival functions nonparametrically. However, when the hazard functions cross, tests in existing R packages do not perform well. To address the issue, we introduce the package **survELtest**, which provides tests for comparing multiple survival functions with possibly crossing hazards. Due to its powerful likelihood ratio formulation, this is the only R package to date that works when the hazard functions cross. We illustrate the use of the procedures in **survELtest** by applying them to data from randomized clinical trials and simulated datasets. We show that these methods lead to more significant results than those obtained by existing R packages.

Introduction

The nonparametric comparison of multiple survival functions is of interest in numerous biomedical settings, such as clinical trials (Robert et al., 2015), preclinical studies (Liebl et al., 2015) and observational studies (Loupy et al., 2013) with right-censored time-to-event endpoints. It has been implemented in existing R packages using log-rank-type statistics. However, these log-rank-type tests can fail to detect differences among survival curves when the hazard functions cross. For example, consider the Kaplan–Meier (KM) estimated survival functions in Figure 1 for the treatment and control groups of patients in a randomized clinical trial. There is a clear gap between the survival curves, which we would expect to be detected by a reasonable statistical test. Nevertheless, the log-rank test provided in the **survival** (Therneau et al., 2020) package returns a p-value of 0.07, indicating no significant difference between the two survival functions at $\alpha = 0.05$. Note that in this case the gap between the survival curves shrinks then widens in the middle of the follow-up period, suggesting that the estimated hazard functions may cross at some time point.

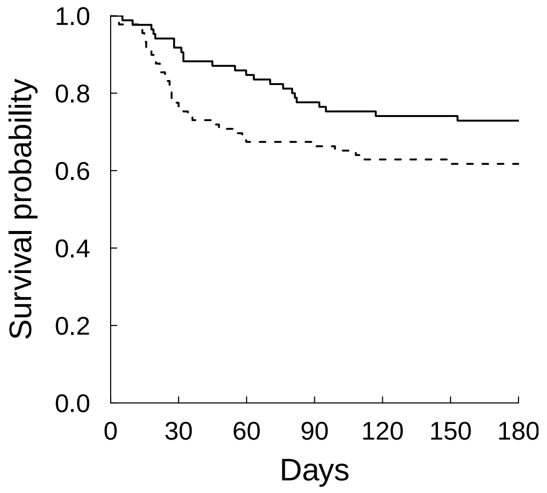


Figure 1: Estimated survival functions for treatment (solid line) versus control (dashed line) groups, based on a randomized clinical trial for treatment of severe alcoholic hepatitis (Nguyen-Khac et al., 2011).

The need to compare survival functions with crossing hazards has been documented in the statistics literature (see, e.g., Pepe and Fleming, 1989; Yang and Prentice, 2010). There are many practical situations in which the hazard functions cross, indicating that the instantaneous treatment effect changes direction. For example, some treatments are initially harmful due to toxicity or other complications, but may be beneficial later on. Other treatments can have short-term benefits but produce side effects in the long run. Despite the varying instantaneous treatment effect, the cumulative treatment effect can still be positive, as reflected by a positive difference between the treatment and control survival functions throughout the follow-up period. It is important to be able to detect such a difference, as the treatment would be worth considering in this case. To this end, an adaptive weighted log-rank test was implemented in the R package **YPmodel** (Sun and Yang, 2020), but this test involves a parametric assumption on the hazard functions, is limited to two-sample comparisons, and cannot deal with the general *k*-sample situation. Another method, the restricted mean survival time (RMST) approach, was implemented in the R package **survRM2** (Uno et al., 2020). However, to our knowledge the RMST method can only deal with two-sample comparisons nonparametrically. For the *k*-sample case, certain model-based assumptions still need to be made (see, e.g., Cronin et al., 2016).

To address this issue, the package **survELtest** (Chang, 2020) provides nonparametric tests that can deal with general *k*-sample comparisons while accounting for possibly crossing hazards. It avoids the pitfalls of log-rank-type statistics, in which negative and positive differences among the estimated hazard functions cancel each other in a weighted sum (see Section Existing test statistics in R and their pitfalls for more details). Further, the statistics are constructed using empirical likelihood (EL), which has been shown to produce tests with optimal power (see, e.g., Kitamura et al., 2012). EL is a nonparametric likelihood which does not assume that the data come from a particular parametric family of distributions. It serves as the basis for constructing a nonparametric likelihood ratio (i.e., the EL ratio), which leads to more efficient inference than Wald-type procedures such as log-rank-type tests, as seen in the literature on parametric (see, e.g., Mukerjee, 1994) and nonparametric inference (Bravo, 2003; Kitamura et al., 2012). There are R packages available for survival analysis using EL, namely **emplik** (Zhou, 2020), **emplik2** (Barton, 2018) and **ELYP** (Zhou, 2018), but they are limited to inference regarding finite-dimensional parameters, whereas our package handles survival functions, an infinite dimensional problem.

Our approach computes EL ratios at each observed uncensored time point, then summarizes them into maximal-deviation-type and integral-type statistics. The statistical theory of this approach and the empirical levels and powers in various simulation scenarios have been studied in Chang and McKeague (2016; 2019), but these authors focused on the technical development of one-sided tests, and did not provide a software package or an accessible guide for implementing the method. In this paper we provide a general framework for both two-sided and one-sided testing, an accessible guide to the R package **survELtest**, and a comparison with existing R packages (reviewed briefly in Section Existing test statistics in R and their pitfalls) in applications to data from clinical trials and simulated datasets.

For *k*-sample nonparametric testing under right censorship, to our knowledge all existing R packages use log-rank-type statistics (see the CRAN Task View *Survival*), often referred to as the weighted log-rank statistics. The package **FHtest** (Oller and Langohr, 2017) and the `survdiff` function in the package **survival** consider the Fleming-Harrington G^ρ family, which belongs to the class of weighted log-rank statistics. The package **clinfun** (Seshan, 2018) adopts a permutation version of the log-rank test. The package **LogrankA** (Richter-Dumke and Rau, 2013) provides a log-rank test based on aggregated survival data. `SurvivalTests` in the **coin** (Hothorn et al., 2019) package implements a reformulated weighted log-rank test as a linear rank test. The **maxstat** (Hothorn, 2017) package performs tests using maximally selected log-rank statistics.

This paper is organized as follows. In the next section, we provide a brief review of *k*-sample nonparametric methods used in existing R packages, their pitfalls, and the use of EL tests as a solution. Section Program description describes our package functions, along with a flow chart showing the procedure for using those functions. In Sections Application of supELtest to threearm data and Application of intELtest to hepatitis data, we apply the proposed routines to datasets from clinical trials, and obtain more significant results than the log-rank-type tests. Some concluding remarks are made in Section Discussion. The availability of the program is given in Section Availability. In the Appendix, we compare our procedures with more existing methods, including those in the aforementioned packages **YPmodel** and **survRM2**, which cannot deal with the general *k*-sample case nonparametrically.

Theoretical background

Existing test statistics in R and their pitfalls

This section briefly reviews the log-rank-type statistics in existing R packages, for testing whether the k survival functions are the same. The null and alternative hypotheses are $H_0\colon S_1 = \ldots = S_k$ and $H_1\colon H_0$ is not true, respectively, where S_j is the survival function of the j-th sample. To quantify the discrepancy between the j-th sample ($j = 1, \ldots, k-1$) and other samples, a weighted sum of differences between the estimated hazard function of the j-th sample and that of the pooled sample is computed. The $k-1$ weighted sums are then summarized using a quadratic form to obtain the final log-rank-type statistic. Different choices of the weight lead to different log-rank-type statistics, of which the commonly used log-rank test is a special case.

To illustrate the pitfalls of this formulation under crossing hazards, we restrict our attention to $k = 2$ for simplicity. When $k = 2$, there is only one weighted sum involved, which can be expressed as

$$\sum_{i=1}^{m} v_i \left\{ \hat{h}_1 (t_i) - \hat{h}_2 (t_i) \right\}, \tag{1}$$

where $0 < t_1 < \ldots < t_m < \infty$ are the (ordered) observed uncensored times, $\hat{h}_j(t_i)$ are the estimated hazard functions at time t_i for the j-th sample, and $v_i \geq 0$ is the corresponding weight at t_i. When the survival functions are different, the hazard functions can cross each other. In this case, there are both positive differences (i.e., when $\hat{h}_1(\cdot) > \hat{h}_2(\cdot)$) and negative differences (i.e., when $\hat{h}_1(\cdot) < \hat{h}_2(\cdot)$) in (1). These differences between the estimated hazard functions cancel out, leading to a smaller value of the statistic and hence a less significant result. Consequently, the formulation can fail to detect the difference between the survival curves.

EL ratio and test statistics

In the proposed package **survELtest**, we use a likelihood ratio statistic, namely a pointwise EL statistic, to replace the estimated hazard difference in (1). This pointwise EL statistic quantifies, at each time point, the difference in the multiple survival functions. It is always non-negative, as are all typical likelihood ratio statistics, which prevents the problematic cancellation described in the previous section. In the rest of Section Theoretical background, we provide a brief description of this approach. More details can be found in Chang and McKeague (2016, 2019).

The pointwise EL statistic is constructed from the following likelihood ratio:

$$\mathcal{R}(t) = \frac{\sup \left\{ L(S_1, S_2, \ldots, S_k) : S_1(t) = S_2(t) = \ldots = S_k(t) \right\}}{\sup \left\{ L(S_1, S_2, \ldots, S_k) \right\}}, \tag{2}$$

where $L(S_1, S_2, \ldots, S_k)$ is a nonparametric likelihood which does not assume that the data come from a particular parametric family of distributions (Thomas and Grunkemeier, 1975). As in a usual (parametric) likelihood ratio, the numerator of (2) maximizes the likelihood subject to the constraint under H_0, whereas the denominator maximizes the likelihood globally, as it corresponds to the union of H_0 and H_1. We then use $-2 \log \mathcal{R}(t)$ as our pointwise EL statistic; such transformation of likelihood ratios has been widely used in the literature. A larger $-2 \log \mathcal{R}(t)$ gives less evidence for $S_1(t) = S_2(t) = \ldots = S_k(t)$.

For the desired simultaneous inference, we summarize the pointwise statistics in two ways. The first, provided by the routine `intELtest`, takes a weighted sum:

$$I = \sum_{i=1}^{m} w_i \left\{ -2 \log \mathcal{R}(t_i) \right\}, \tag{3}$$

where $w_i \geq 0$ is the weight at each t_i. This is an integral-type statistic because the summation can be written into a stochastic integral. The form of a weighted sum is similar to the components of the log-rank-type statistics shown in the previous section. Here we avoid ad hoc choices of the weight w_i by setting equal weight for data with no ties. We do this because the EL statistic $-2 \log \mathcal{R}(t_i)$ implicitly provides optimal (i.e., nonparametric-likelihood-optimized) weighting for contrasting the survival functions. More details regarding the weighting schemes are given in Section Weight.

Another way to summarize the pointwise statistics is to take a maximum $K = \sup_{i=1,\ldots,m}\{-2 \log \mathcal{R}(t_i)\}$, which is provided by the function `supELtest`. Such maximal-deviation-type statistics have been used in the classical Kolmogorov–Smirnov test, and are more sensitive to local differences amongst survival curves (i.e., differences among survival curves that appear only in a short period of time). In contrast, the integral-type statistic I in (3) is designed to detect moderate differences spread over a sizable portion of the follow-up period. The choice between the two statistics should be guided by prior knowledge and practical considerations. In particular, if prior knowledge does not suggest the presence of local differences, we recommend I for general use. Otherwise K can be implemented to exploit the additional knowledge of the existence of a local difference. For example, a local difference is present when medical knowledge suggests that a treatment has a benefit in some localized time interval, or when a pilot study shows that the difference among the KM estimated survival curves appears only over a short period of time. In the latter case, the evidence would be even stronger if a significant result was obtained from a statistical test that is sensitive to such local differences, such as the maximal-deviation-type statistics described above.

Two-step procedure for one-sided testing

So far, we have focused on two-sided testing. For one-sided testing, we consider the alternative H_1^o: $S_1 \succ S_2 \succ \ldots \succ S_k$ that there is an ordering among the survival functions, where $f \succ g$ for functions $f(t)$ and $g(t)$ of t means $f(t) \geq g(t)$ for all t with a strict inequality for some t. The EL statistics are the same as the ones in the previous section, except that now we put an additional constraint $S_1(t) \geq S_2(t) \geq \ldots \geq S_k(t)$ in the denominator of $\mathcal{R}(t)$ in (2).

The resulting EL test will be preceded by an initial test that excludes the possibility of crossings or alternative orderings among the survival functions. The reason is due to the fact that for functional parameters (e.g., survival functions), testing a one-sided alternative hypothesis usually involves certain assumptions, such as that the functions are not crossed and that their ordering is as hypothesized. These assumptions may be checked using the initial test, with the null hypothesis being that the assumptions are not satisfied, versus the alternative that they are. If the null hypothesis of the initial test is rejected, we conclude that the assumptions are satisfied and proceed to the EL test. Rejection of the null hypothesis H_0 of the EL test then gives support for H_1. On the other hand, if the null hypothesis of the initial test is not rejected, we conclude that the assumptions are not satisfied and do not proceed to the EL test. The family-wise error rate of this two-step procedure has been shown to be asymptotically controlled at the same α-level as the individual tests.

Weight

As mentioned in Section EL ratio and test statistics, we need to specify w_i in (3). This can be done by setting the value of the argument `wt` in the routine `intELtest`. The default is an objective weight $w_i = d_i/n$, where d_i denotes the number of events at each time point t_i and n is the total sample size. This simplifies to equal weight $w_i = 1/n$ when there are no ties (i.e., $d_i = 1$) in the data. This default weight is specified by the option `wt = "p.event"`.

Despite the default weight, we provide in `intELtest` two alternative options that have been used for integral-type statistics in the literature. One option is $w_i = \hat{F}(t_i) - \hat{F}(t_{i-1})$ for $i = 1, \ldots, m$, where $\hat{F}(t) = 1 - \hat{S}(t)$, $\hat{S}(t)$ is the pooled KM estimator, and $t_0 \equiv 0$. This w_i reduces to the objective weight $w_i = d_i/n$ when there is no censoring (see, e.g., El Barmi and McKeague, 2013). The resulting I can be seen as an empirical version of the expected negative two log EL ratio under H_0. This weight can be chosen via the option `wt = "dF"`.

Another weight, proposed by Pepe and Fleming (1989), is $w_i = t_{i+1} - t_i$ for $i = 1, \ldots, m$, where $t_{m+1} \equiv t_m$. This approach gives more weight to the time intervals when there are fewer observed uncensored times, but can be affected by extreme observations. This weight can be chosen via the option `wt = "dt"`.

Bootstrap critical values

Having computed the statistics, we need to calibrate the tests. Possible methods include bootstrapping or simulating the limiting distributions. We choose the former for the following two reasons: (a) in small samples, calibration using the bootstrap can perform better than using the limiting distribution (Heller and Venkatraman, 1996). (b) in our experience the bootstrap can be more computationally efficient, since the limiting distributions of I and K involve stochastic processes that depend on unknown parameters.

Here we adopt a Gaussian multiplier bootstrap approach, which is commonly used instead of the nonparametric bootstrap in survival analysis to avoid producing tied data in the bootstrap samples. To form the bootstrap samples, the original data are perturbed using independent standard Gaussian random variables, termed Gaussian multipliers (see, e.g., Parzen et al., 1997). We denote the number of bootstrap samples as B, which is specified by the `nboot` argument (default is 1000). In cases when m is too large for the computation to be handled with reasonable speed, we split the calculation of the B bootstrap replications into `nsplit` parts, where `nsplit` = $\lceil m$ / `nlimit`$\rceil$ (default `nlimit = 200`). Here and in the sequel, if we do not specify which R function an option or argument applies to, then the option or argument applies to all the functions provided by the **survELtest** package.

Since the bootstrap involves random sampling, the critical values will differ based on different sets of bootstrap samples. To make the critical values reproducible, we set a seed for random number generation via the `seed` option in our routines.

User guide and numerical examples

Program description

The **survELtest** package can be installed along with **survELtest** using the following R code:

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

The following code loads the package:

```
>   library(survELtest)
```

The main routines in **survELtest** are `intELtest`, `supELtest`, `nocrossings`, and `ptwiseELtest`. The `intELtest` routine conducts testing based on the integrated EL statistics I in (3) that can detect moderate differences among the survival curves over time. The `supELtest` routine conducts testing based on the maximally selected EL statistics K that is more sensitive to differences locally in time. Each routine gives a two- or one-sided test statistic, the critical value based on bootstrap, and the p-value of the test. As mentioned in Section EL ratio and test statistics, the choice between the two routines should be guided by prior knowledge and practical considerations regarding whether there is a local difference among the survival curves. The choice between two-sided and one-sided testing should be determined a priori as well, depending on the research question of interest. One-sided testing can be specified by the option `sided = 1` in both `intELtest` and `supELtest`, but should be preceded by the initial test in `nocrossings` to exclude the possibility of crossings or alternative orderings among the survival functions. While the first three routines provide simultaneous testing, `ptwiseELtest` conducts pointwise testing to compare the survival curves at each time point. It can be used to identify periods of local differences, after `intELtest` or `supELtest` test gives a significant result. A flow chart of the procedure for using the **survELtest** package is given in Figure 2. Methods defined for the objects produced by the main routines are provided for `print` and `summary`. In addition to the aforementioned routines, **survELtest** contains four datasets: `hepatitis`, `threearm`, `hazardcross` and `hazardcross_Weibull`, which will be analyzed in Sections Application of supELtest to threearm data, Application of intELtest to hepatitis data, and the Appendix to illustrate the use of the routines.

A summary of the R code and the input arguments of the routines are given as follows. Among the input arguments below, only the `formula` input is compulsory. The rest of the arguments can be omitted if the default settings are used.

```
>   intELtest(formula, data = NULL, group_order = NULL, t1 = 0, t2 = Inf, sided = 2,
+   nboot = 1000, wt = "p.event", alpha = 0.05, seed = 1011, nlimit = 200)

>   supELtest(formula, data = NULL, group_order = NULL, t1 = 0, t2 = Inf, sided = 2,
+   nboot = 1000, alpha = 0.05, seed = 1011, nlimit = 200)

>   nocrossings(formula, data = NULL, group_order = NULL, t1 = 0, t2 = Inf, sided = 2,
+   nboot = 1000, alpha = 0.05, seed = 1011, nlimit = 200)

>   ptwiseELtest(formula, data = NULL, group_order = NULL, t1 = 0, t2 = Inf, sided = 2,
+   nboot = 1000, alpha = 0.05, seed = 1011, nlimit = 200)
```

The time needed to run these functions depends on the total number n of observations, the number k of samples, the speed of the processor and the amount of PC memory. For example, to run `intELtest` with the default settings, it takes about 0.32 seconds on the dataset `hepatitis` with $n = 174$ and $k = 2$, and 1.79 minutes on the dataset `threearm` with $n = 664$ and $k = 3$, on a desktop computer with Intel i7-7700 CPU @ 3.60 GHz and 64 GB RAM.

- `formula`: a formula object, with a Surv object on the left of the $\sim$ operator and the grouping variable on the right. The Surv object involves two variables: the observed survival and censoring times, and the censoring indicator, which takes a value of 1 if the observed time is uncensored and 0 otherwise. The grouping variable takes different values for different groups. If not found in `data` described below, the variables in the `formula` should be already defined by the user or in attached R objects.
- `data`: an optional data frame containing the variables in the `formula`: the observed survival and censoring times, the censoring indicator, and the grouping variable. The default is the data frame with three columns of variables taken from the `formula`: column 1 contains the observed survival and censoring times, column 2 the censoring indicator, and column 3 the grouping variable.

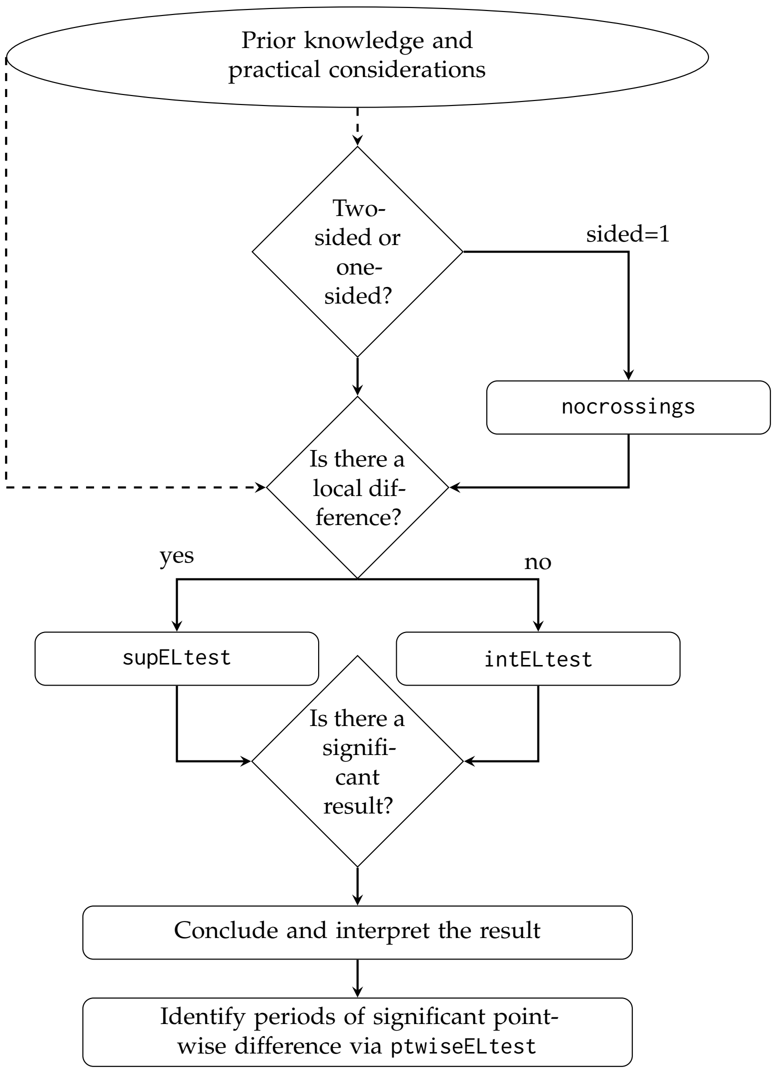

Prior knowledge and practical considerations
Two-sided or one-sided?
sided=1
nocrossings
Is there a local difference?
yes
no
supELtest
intELtest
Is there a significant result?
Conclude and interpret the result
Identify periods of significant pointwise difference via ptwiseELtest

Figure 2: Flow chart of the procedure for using the routines in the **survELtest** package.

- `group_order`: a k-vector containing the values of the grouping variable, with the j-th element being the group hypothesized to have the j-th highest survival rates, $j = 1, \dots, k$. The default is the vector of sorted grouping variables.
- `t1`: the first endpoint of a prespecified time interval, if any, to which the comparison of the survival functions is restricted. The default value is 0.
- `t2`: the second endpoint of a prespecified time interval, if any, to which the comparison of the survival functions is restricted. The default value is ∞.
- `sided`: 2 if two-sided test, and 1 if one-sided test. The default value is 2.
- `nboot`: the number of bootstrap replications in calculating critical values for the tests. The default value is 1000.
- `wt`: the name of the weight to be used in the integrated EL statistics in `intELtest`: "`p.event`", "`dF`", or "`dt`". The default is "`p.event`".
- `alpha`: the pre-specified significance level of the tests. The default value is 0.05.
- `seed`: the seed for the random number generator in R, for generating bootstrap samples needed to calculate the critical values for the tests. The default value is 1011.
- `nlimit`: a number used to calculate `nsplit = `$\lceil m$` / nlimit`$\rceil$, the number of parts into which the calculation of the `nboot` bootstrap replications is split. The use of this variable can make computation faster when the number of time points m is large. The default value for `nlimit` is 200.

Application of supELtest to threearm data

In this section we apply the routines `supELtest` and `ptwiseELtest` to the dataset `threearm` provided in the **survELtest** package, and compare the results with the log-rank-type tests for trend. The dataset is obtained by resampling from a perturbed dataset of patients from a randomized clinical trial for the treatment of major depression, where the perturbation is achieved by adding a random $\mathrm{U}(-0.01\ell, 0.01\ell)$ variable to existing observations, ℓ is the smallest observation in the original data, and the resampling is done by conditional bootstrapping with stratified survival and censoring distributions using the `censboot` function in the package **boot** (Canty and Ripley, 2020). The original data were analyzed by Chang and McKeague (2019), who observed a local difference among the survival functions.

The purpose of analyzing the `threearm` dataset is to assess whether the survival functions of the three arms are ordered: that is, whether the experimental treatment group ($n_1 = 262$) is better than the standard treatment group ($n_2 = 267$), which is in turn superior to the placebo group ($n_3 = 135$). This question can be answered using the one-sided tests described in Section Two-step procedure for one-sided testing. Since prior knowledge suggests that there is a local difference among the survival functions, here we conduct the maximally selected EL test via `supELtest`.

The endpoint of the clinical trial is time (in days) to first remission. Because a shorter time to first remission is desirable, a treatment with a lower value of the survival function is better in this dataset. Based on this information, from the KM estimated survival curves in the left panel of Figure 3, it seems that the three groups are similar initially but become ordered for the rest of the follow-up period.

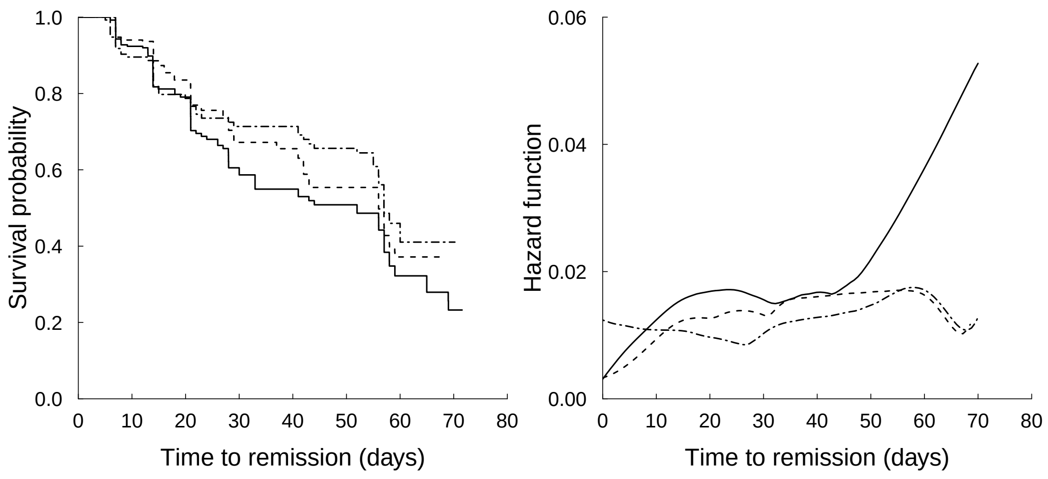


Figure 3: The KM estimated survival curves (left) and the estimated hazard functions (right) in the `threearm` dataset: experimental treatment group (solid), standard treatment group (dashed) and placebo group (two-dashed).

To see if the curves are statistically significantly ordered, we start with conducting the commonly used log-rank-type tests. The trend test is needed for the one-sided research question. Using the common choice $c(3, 2, 1)$ for the score vector (see, e.g., Andersen et al., 1993, page 388), the log-rank test for trend is implemented as follows:

```
>   library(survival)
>   dat = Surv(threearm[, 1], threearm[, 2])
>   logrank = survdiff(dat ~ threearm[, 3])
>   score_vec = 3 : 1
>   logrankteststat = matrix(score_vec, nrow = 1, ncol = 3)
+   %*% (logrank$obs - logrank$exp) / sqrt(matrix(score_vec, nrow = 1, ncol = 3)
+   %*% (logrank$var) %*% matrix(score_vec, nrow = 3, ncol = 1))
>   if(logrankteststat < 0){
+     pval = 2 * pnorm(logrankteststat)
+   }else{
+     pval = 2 * (1 - pnorm(logrankteststat))
+   }
>   round(pval, 2)
```

```
     [,1]
[1,] 0.04
```

As the log-rank test for trend gives a p-value of 0.04, we conclude that the three survival functions are ordered at $\alpha = 0.05$. The other extreme in the G^{ρ} family can be implemented by setting `survdiff(dat ~ threearm[, 3], rho = 1)` in the above code, which leads to a p-value of 0.08. These results mean the weighted log-rank statistics in the entire G^{ρ} family give a p-value that ranges from 0.04 to 0.08 for the trend test.

Now we conduct the proposed one-sided testing for the `threearm` data. We anticipate a more significant result than the log-rank-type tests, as there seems to be crossing among the estimated hazard functions in the right panel of Figure 3, created using the function `muhaz` in the package **muhaz** (Hess and Gentleman, 2019) with the default settings. The initial test and the maximally selected EL test are implemented by the routines `nocrossings` and `supELtest`, respectively. (Note that if two-sided testing is conducted instead, then the initial test is not needed.) To use the routines, we need to specify two options: `sided = 1` for the one-sided test, and `group_order = c(3, 2, 1)`, since the hypothesized order among the three arms with the survival rates ranging from the largest to the smallest is the placebo (coded as 3 in the grouping variable), standard treatment (coded as 2), and experiment treatment (coded as 1). The rest of the options are kept at their default values. The R code for performing the initial test is as follows:

```
>   nocrossings(Surv(threearm$time, threearm$censor) ~ threearm$group,
+   group_order = c(3, 2, 1), sided = 1)

Call:
nocrossings(formula = Surv(threearm$time, threearm$censor) ~ threearm$group,
group_order = c(3, 2, 1), sided = 1)

Decision = 1
```

A decision value of 1 means there is no crossing or alternative orderings among the survival functions. Thus, we can proceed to the main (maximally selected EL) test in the second step:

```
>   supELtest(Surv(threearm$time, threearm$censor) ~ threearm$group,
+   group_order = c(3, 2, 1), sided = 1)

Call:
supELtest(formula = Surv(threearm$time, threearm$censor) ~ threearm$group,
group_order = c(3, 2, 1), sided = 1)

One-sided maximally selected EL test statistic = 14.23, p = 0.004
```

As the maximally selected EL test gives a p-value < 0.01, we obtain the same conclusion—that the three survival functions are significantly ordered—as the log-rank-type tests for trend, but with a statistically more significant result. This finding is as we anticipated after seeing the crossing estimated hazard functions in the right panel of Figure 3.

Since our procedure leads to the conclusion that the survival functions are ordered, it can be of interest to identify periods of local differences for further clinical investigation. To this end, we can use the routine `ptwiseELtest` for pointwise testing at each observed uncensored time point:

```
>   ptwise = ptwiseELtest(Surv(threearm$time, threearm$censor) ~ threearm$group,
+   group_order = c(3, 2, 1), sided = 1)
```

The list of the time points at which the survival functions are ordered (i.e., `decision == 1`) is obtained by

```
>   round(ptwise$result_dataframe$time_pts[ptwise$result_dataframe$decision == 1], 2)

 [1] 13.91 13.91 13.91 13.92 13.92 13.92 13.92 13.93 13.98 13.99 13.99 14.00 14.00
[14] 14.00 14.01 14.01 20.96 20.96 27.98 27.99 28.00 28.00 28.00 28.02 28.02 28.98
[27] 28.99 29.01 30.00 32.96 36.97 40.97 40.98 40.99 41.02 41.98 41.98 41.99 42.00
[40] 42.01 42.02 42.99 43.01 43.02 43.02 44.00 44.00 51.97 51.97 55.00 56.01 56.01
[53] 56.02 59.03 59.04 64.97 68.98 69.01
```

From the result, we see there are local differences occurring near the time points 14, 21, 30, 40, 52, 56, 59, 65 and 69 days.

Application of intELtest to hepatitis data

Now we turn to our motivating example in the Introduction and demonstrate the use of `intELtest` and its benefit over the log-rank-type tests. The corresponding dataset `hepatitis` is provided in the **survELtest** package. The dataset was obtained by reconstructing survival and censoring information (Guyot et al., 2012) based on digitizing the KM curves presented in Nguyen-Khac et al. (2011). It contains survival data (in days, rounded to one decimal place) from patients in a randomized clinical trial for the treatment of severe alcoholic hepatitis. The purpose of the clinical trial was to assess if the treatment group ($n_1 = 85$) had a significantly different survival rate than the control group ($n_2 = 89$).

From the KM estimated survival curves in Figure 1, the survival rate of the treatment group seems to be greater than that of the control group over the entire follow-up period. To see whether the difference between the survival functions are statistically significant, we start with conducting the commonly used two-sided log-rank test:

```
>   library(survival)
>   dat = Surv(hepatitis[, 1], hepatitis[, 2])
>   logrank = survdiff(dat ~ hepatitis[, 3])
>   round(1 - pchisq(logrank$chisq, df = 1), 2)

[1] 0.07
```

The log-rank test gives a p-value of 0.07, failing to detect a difference between the survival curves at $\alpha = 0.05$. The reason may be due to the crossing estimated hazard functions in Figure 4 (created using the function `muhaz` in the package **muhaz** with the default settings). We also conduct another log-rank-type test—the Peto and Peto's modification of the Gehan-Wilcoxon test—by setting `survdiff(dat ~ hepatitis[, 3], rho = 1)` in the above code, which leads to a p-value of 0.05. Since this test and the log-rank test are the two extremes in the G^ρ family, these results mean the weighted log-rank statistics in the entire G^ρ family give either insignificant or borderline significant conclusions.

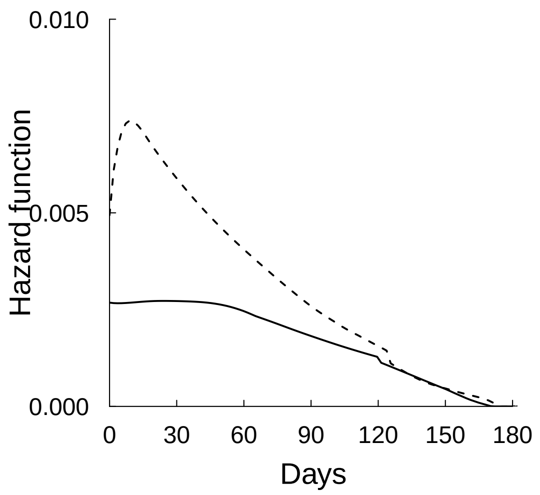


Figure 4: Estimated hazard functions for treatment (solid line) versus control (dashed line) groups.

Now we apply the proposed two-sided integrated EL test to the `hepatitis` data to see if we can better detect a difference between the survival functions. The default options are used and the R code is as simple as

```
>   intELtest(Surv(hepatitis$time, hepatitis$censor) ~ hepatitis$group)

Call:
intELtest(formula = Surv(hepatitis$time, hepatitis$censor) ~ hepatitis$group)

Two-sided integrated EL test statistic = 1.42, p = 0.007
```

As the integrated EL test gives a p-value < 0.01, we conclude there is a significant difference between the two survival functions at $\alpha = 0.05$. The p-value is much smaller than those given by the previous log-rank-type tests, which indicates that the integrated EL test is better at detecting the difference between the survival curves.

Note the decision as to whether there is a significant discrepancy between the two survival functions is totally different for the log-rank and the integrated EL tests at $\alpha = 0.05$. It may be tempting to pick the most significant result, but this practice is data snooping and has been shown to be problematic. Instead, we recommend setting a primary method prior to the data analysis and making the decision based on that method. Any other methods are treated as secondary, and their results can serve an exploratory purpose for future work.

Discussion

In this paper we introduce the R package **survELtest** for comparing two or more survival functions nonparametrically based on right-censored data. It is the only R package to date that utilizes the powerful likelihood ratio formulation instead of log-rank-type statistics, thereby performing well when the hazard functions cross. We provide both maximal-deviation-type and integral-type statistics, for detecting local and cumulative differences among the survival functions, respectively.

The use of the software is illustrated using two data sets from randomized clinical trials, where the estimated survival functions seem to be ordered, but the estimated hazard functions cross. In these cases, our procedures lead to more significant results than the results obtained from the log-rank-type tests. Specifically, in one of the examples, the original clinical trial concludes that there is no significant difference between the treatment and the control groups (log-rank $p = 0.07$), whereas our test suggests otherwise, based on a much smaller p-value < 0.01. We envision the **survELtest** package will be valuable for finding more significant results in numerous biomedical settings involving the comparison of multiple survival functions, especially in the presence of crossing hazards.

Availability

The package is available from the Comprehensive R Archive Network at `https://CRAN.R-project.org/package=survELtest`. The development website is available at `https://github.com/news11/survELtest`.

Acknowledgements

The research of Hsin-wen Chang was partially supported by Ministry of Science and Technology of Taiwan under grants 106-2118-M-001-015-MY3 and MOST 109-2118-M-001-005-. The authors thank Yu-Ju Wang for computational support and Shih-Hao Huang for helpful comments. The authors declare that they have no conflict of interest.

Bibliography

P. K. Andersen, Ø. Borgan, R. D. Gill, and N. Keiding. *Statistical Models Based on Counting Processes*. New York: Springer, 1993. URL `https://doi.org/10.1007/978-1-4612-4348-9`. [p113]

W. H. Barton. *emplik2: Empirical Likelihood Ratio Test for Two Samples with Censored Data*, 2018. URL `https://CRAN.R-project.org/package=emplik2`. R package version: 1.21. [p108]

F. Bravo. Second-order power comparisons for a class of nonparametric likelihood-based tests. *Biometrika*, 90(4):881–890, 2003. URL `https://doi.org/10.1093/biomet/90.4.881`. [p108]

A. Canty and B. Ripley. *boot: Bootstrap Functions (Originally by Angelo Canty for S)*, 2020. URL `https://CRAN.R-project.org/package=boot`. R package version: 1.3-25. [p113]

H.-w. Chang. *survELtest: Comparing Multiple Survival Functions with Crossing Hazards*, 2020. URL `https://CRAN.R-project.org/package=surELtest`. R package version: 2.0.1. [p108]

H.-w. Chang and I. W. McKeague. Empirical likelihood based tests for stochastic ordering under right censorship. *Electronic Journal of Statistics*, 10(2):2511–2536, 2016. URL `https://doi.org/10.1214/16-EJS1180`. [p108, 109]

H.-w. Chang and I. W. McKeague. Nonparametric testing for multiple survival functions with non-inferiority margins. *Annals of Statistics*, 47(1):205–232, 2019. URL https://doi.org/10.1214/18-AOS1686. [p108, 109, 113]

A. Cronin, L. Tian, and H. Uno. strmst2 and strmst2pw: New commands to compare survival curves using the restricted mean survival time. *Stata Journal*, 16(3):702–716, 2016. URL https://doi.org/10.1177/1536867X1601600310. [p108]

H. El Barmi and I. W. McKeague. Empirical likelihood based tests for stochastic ordering. *Bernoulli*, 19: 295–307, 2013. URL https://doi.org/10.3150/11-BEJ393. [p110]

P. Guyot, A. E. Ades, M. J. N. M. Ouwens, and N. J. Welton. Enhanced secondary analysis of survival data: reconstructing the data from published Kaplan–Meier survival curves. *BMC Medical Research Methodology*, 12(1):1–13, 2012. URL https://doi.org/10.1186/1471-2288-12-9. [p115]

G. Heller and E. S. Venkatraman. Resampling procedures to compare two survival distributions in the presence of right-censored data. *Biometrics*, 52(4):1204–1213, 1996. URL https://doi.org/10.2307/2532836. [p110]

K. Hess and R. Gentleman. *muhaz: Hazard Function Estimation in Survival Analysis*, 2019. URL https://CRAN.R-project.org/package=muhaz. R package version: 1.2.6.1. [p114]

T. Hothorn. *maxstat: Maximally Selected Rank Statistics*, 2017. URL https://CRAN.R-project.org/package=maxstat. R package version: 0.7-25. [p108]

T. Hothorn, H. Winell, K. Hornik, M. A. van de Wiel, and A. Zeileis. *coin: Conditional Inference Procedures in a Permutation Test Framework*, 2019. URL https://CRAN.R-project.org/package=coin. R package version: 1.3-1. [p108]

Y. Kitamura, A. Santos, and A. M. Shaikh. On the asymptotic optimality of empirical likelihood for testing moment restrictions. *Econometrica*, 80(1):413–423, 2012. URL https://doi.org/10.3982/ECTA8773. [p108]

M. Liebl, J. Windschmitt, A. S Besemer, A.-K. Schäfer, H. Reber, C. Behl, and A. Clement. Low-frequency magnetic fields do not aggravate disease in mouse models of Alzheimer's disease and amyotrophic lateral sclerosis. *Scientific Reports*, 5:8585, 2015. URL https://doi.org/10.1038/srep08585. [p107]

A. Loupy, C. Lefaucheur, D. Vernerey, C. Prugger, J.-P. D. van Huyen, N. Mooney, C. Suberbielle, V. Frémeaux-Bacchi, A. Méjean, F. Desgrandchamps, D. Anglicheau, D. Nochy, D. Charron, J.-P. Empana, M. Delahousse, C. Legendre, D. Glotz, G. S. Hill, A. Zeevi, and X. Jouven. Complement-binding anti-HLA antibodies and kidney-allograft survival. *New England Journal of Medicine*, 369 (13):1215–1226, 2013. URL https://doi.org/10.1056/NEJMoa1302506. [p107]

R. Mukerjee. Comparison of tests in their original forms. *Sankhyā: The Indian Journal of Statistics, Series A (1961-2002)*, 56(1):118–127, 1994. URL https://www.jstor.org/stable/25050974. [p108]

E. Nguyen-Khac, T. Thevenot, M.-A. Piquet, S. Benferhat, O. Goria, D. Chatelain, B. Tramier, F. Dewaele, S. Ghrib, M. Rudler, N. Carbonell, H. Tossou, A. Bental, B. Bernard-Chabert, and J.-L. Dupas. Glucocorticoids plus N-acetylcysteine in severe alcoholic hepatitis. *New England Journal of Medicine*, 365(19):1781–1789, 2011. URL https://doi.org/10.1056/NEJMoa1101214. [p107, 115]

R. Oller and K. Langohr. *FHtest: Tests for Right and Interval-Censored Survival Data Based on the Fleming-Harrington Class*, 2017. URL https://CRAN.R-project.org/package=FHtest. R package version: 1.4. [p108]

M. I. Parzen, L. J. Wei, and Z. Ying. Simultaneous confidence intervals for the difference of two survival functions. *Scandinavian Journal of Statistics*, 24(3):309–314, 1997. URL https://doi.org/10.1111/1467-9469.t01-1-00065. [p110]

M. S. Pepe and T. R. Fleming. Weighted Kaplan–Meier statistics: a class of distance tests for censored survival data. *Biometrics*, 45(2):497–507, 1989. URL https://doi.org/10.2307/2531492. [p107, 110]

J. Richter-Dumke and R. Rau. *LogrankA: Logrank Test for Aggregated Survival Data*, 2013. URL https://CRAN.R-project.org/package=LogrankA. R package version:1.0. [p108]

C. Robert, B. Karaszewska, J. Schachter, P. Rutkowski, A. Mackiewicz, D. Stroiakovski, M. Lichinitser, R. Dummer, F. Grange, L. Mortier, V. Chiarion-Sileni, K. Drucis, I. Krajsova, A. Hauschild, P. Lorigan, P. Wolter, G. V. Long, K. Flaherty, P. Nathan, A. Ribas, A.-M. Martin, P. Sun, W. Crist, J. Legos, S. D. Rubin, S. M. Little, and D. Schadendorf. Improved overall survival in melanoma with combined dabrafenib and trametinib. *New England Journal of Medicine*, 372(1):30–39, 2015. URL https://doi.org/10.1056/NEJMoa1412690. [p107]

V. E. Seshan. *clinfun: Clinical Trial Design and Data Analysis Functions*, 2018. URL https://CRAN.R-project.org/package=clinfun. R package version: 1.0.15. [p108]

J. Sun and S. Yang. *YPmodel: The Short-Term and Long-Term Hazard Ratio Model for Survival Data*, 2020. URL https://CRAN.R-project.org/package=YPmodel. R package version: 1.4. [p108]

T. M. Therneau, T. Lumley, E. Atkinson, and C. Crowson. *survival: Survival Analysis*, 2020. URL https://CRAN.R-project.org/package=survival. R package version: 3.2-3. [p107]

D. R. Thomas and G. L. Grunkemeier. Confidence interval estimation of survival probabilities for censored data. *Journal of the American Statistical Association*, 70:865–871, 1975. URL https://doi.org/10.1080/01621459.1975.10480315. [p109]

H. Uno, L. Tian, A. Cronin, C. Battioui, and M. Horiguchi. *survRM2: Comparing Restricted Mean Survival Time*, 2020. URL https://CRAN.R-project.org/package=survRM2. R package version: 1.0-3. [p108]

S. Yang and R. Prentice. Improved logrank-type tests for survival data using adaptive weights. *Biometrics*, 66(1):30–38, 2010. URL https://doi.org/10.1111/j.1541-0420.2009.01243.x. [p107]

M. Zhou. *ELYP: Empirical Likelihood Analysis for the Cox Model and Yang-Prentice (2005) Model*, 2018. URL https://CRAN.R-project.org/package=ELYP. R package version: 0.7-5. [p108]

M. Zhou. *emplik: Empirical Likelihood Ratio for Censored/Truncated Data*, 2020. URL https://CRAN.R-project.org/package=emplik. R package version: 1.1-1. [p108]

Hsin-wen Chang
Institute of Statistical Science
Academia Sinica
128 Academia Road, Section 2,
Nankang, Taipei 11529, Taiwan (R.O.C)
ORCiD: 0000-0003-4566-7047
hwchang@stat.sinica.edu.tw

Pei-Yuan Tsai
Institute of Statistical Science
Academia Sinica
Taipei, Taiwan (R.O.C)

Jen-Tse Kao
Institute of Statistical Science
Academia Sinica
Taipei, Taiwan (R.O.C)

Guo-You Lan
Department of Economics
National Chengchi University
Taipei, Taiwan (R.O.C)

Appendix: Comparison of survELtest with other existing tests in two simulated datasets

Here we provide two more examples for comparing our procedures with other existing tests in the literature, namely the log-rank test, the Peto and Peto's modification of the Gehan-Wilcoxon test, the adaptive weighted log-rank test implemented in the R package **YPmodel**, and the RMST method implemented in the R package **survRM2**. Since the latter two methods cannot deal with the general k-sample case nonparametrically, the examples provided here are restricted to the two-sample case. For the first dataset `hazardcross`, the survival time is generated from the piecewise exponential model displayed in the left panel of Figure 5. Since the difference between the true survival curves appears only during $[0, 6]$ but not later on, we use `supELtest` to detect such local differences. For the second dataset `hazardcross_Weibull`, the survival time is generated from the Weibull model displayed in the right panel of Figure 5. We use `intELtest` because the difference between the true survival curves is spread over the entire follow-up period. For both datasets, the true hazard functions cross, but there is an obvious gap between the survival curves. The censoring distributions are specified to be the same in each arm, and uniform with administrative censoring at t = 10 and a censoring rate of 25% in the first group. We use the default settings in implementing the tests given in the aforementioned two packages.

The results are given in Table 1. Our tests provide more significant results in detecting the gap between the survival curves than any of the other tests for both datasets.

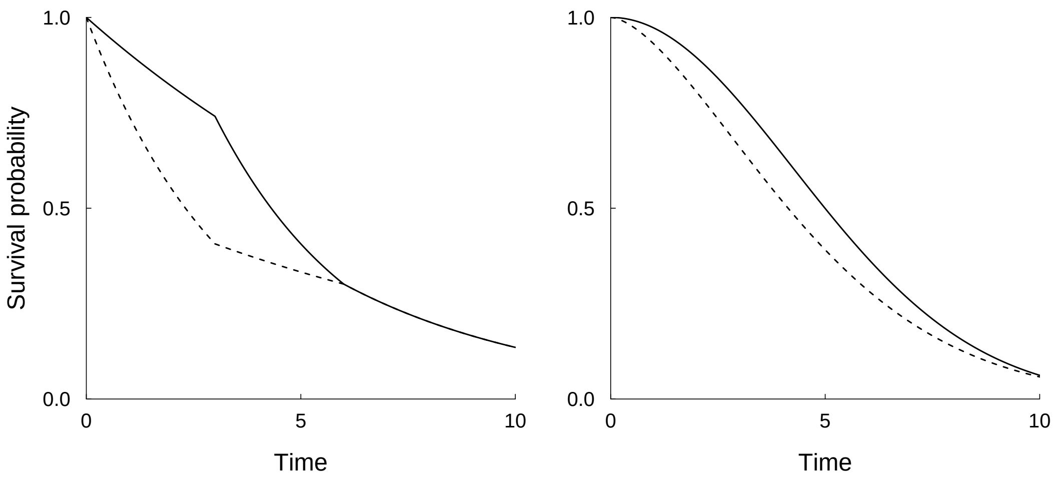


Figure 5: The true survival curves for generating `hazardcross` (left) and `hazardcross_Weibull` (right) datasets: the first (solid) and second (dashed) groups.

Table 1: p-values from various tests for comparing the survival curves in the `hazardcross` and `hazardcross_Weibull` datasets. EL denotes the suitable EL test implemented in the R package **survELtest**, PP denotes the Peto and Peto's modification of the Gehan-Wilcoxon test, YP denotes the adaptive weighted log-rank test implemented in the R package **YPmodel**, and dRMST and rRMST denote the results in the R package **survRM2** for the difference in and the ratio of RMST, respectively.

Datasets	EL	log-rank	PP	YP	dRMST	rRMST
`hazardcross`	0.037	0.106	0.060	0.096	0.126	0.130
`hazardcross_Weibull`	0.005	0.080	0.006	0.009	0.014	0.018

A Unified Algorithm for the Non-Convex Penalized Estimation: The ncpen Package

by Dongshin Kim, Sangin Lee[*] *and Sunghoon Kwon*[†]

Abstract Various R packages have been developed for the non-convex penalized estimation but they can only be applied to the smoothly clipped absolute deviation (SCAD) or minimax concave penalty (MCP). We develop an R package, entitled **ncpen**, for the non-convex penalized estimation in order to make data analysts to experience other non-convex penalties. The package **ncpen** implements a unified algorithm based on the convex concave procedure and modified local quadratic approximation algorithm, which can be applied to a broader range of non-convex penalties, including the SCAD and MCP as special examples. Many user-friendly functionalities such as generalized information criteria, cross-validation and ridge regularization are provided also.

Introduction

The penalized estimation has been one of the most important statistical techniques for high dimensional data analysis, and many penalties have been developed such as the least absolute shrinkage and selection operator (LASSO) (Tibshirani, 1996), smoothly clipped absolute deviation (SCAD) (Fan and Li, 2001), and minimax concave penalty (MCP) (Zhang, 2010). In the context of R, many authors released fast and stable R packages for obtaining the whole solution path of the penalized estimator for the generalized linear model (GLM). For example, **lars** (Efron et al., 2004), **glmpath** (Park and Hastie, 2007) and **glmnet** (Friedman et al., 2007) implement the LASSO. Packages such as **plus** (Zhang, 2010), **sparsenet** (Mazumder et al., 2011), **cvplogit** (Jiang and Huang, 2014) and **ncvreg** (Breheny and Huang, 2011) implement the SCAD and MCP. Among them, **glmnet** and **ncvreg** are very fast, stable, and well-organized, presenting various user-friendly functionalities such as the cross-validation and ℓ_2-stabilization (Zou and Hastie, 2005; Huang et al., 2016b).

The non-convex penalized estimation has been studied by many researchers (Fan and Li, 2001; Kim et al., 2008; Huang et al., 2008; Zou and Li, 2008; Zhang and Zhang, 2012; Kwon and Kim, 2012; Friedman, 2012). However, there is still a lack in research on the algorithms that exactly implement the non-convex penalized estimators for the non-convexity of the objective function. One nice approach is using the coordinate descent (CD) algorithm (Tseng, 2001; Breheny and Huang, 2011). The CD algorithm fits quite well for some quadratic non-convex penalties such as the SCAD and MC (Mazumder et al., 2011; Breheny and Huang, 2011; Jiang and Huang, 2014) since each coordinate update in the CD algorithm becomes an easy convex optimization problem with a closed form solution. This is the main reason for the preference of the CD algorithm implemented in many R packages such as **sparsenet** and **ncvreg**. However, coordinate updates in the CD algorithm require extra univariate optimizations for other non-convex penalties such as the log and bridge penalties (Zou and Li, 2008; Huang et al., 2008; Friedman, 2012), which severely lowers the convergence speed. Another subtle point is that the CD algorithm requires standardization of the input variables and need to enlarge the concave scale parameter in the penalty (Breheny and Huang, 2011) to obtain the local convergence, which may cause to lose an advantage of non-convex penalized estimation (Kim and Kwon, 2012) and give much different variable selection performance (Lee, 2015).

In this paper, we develop an R package **ncpen** for the non-convex penalized estimation based on the convex-concave procedure (CCCP) or difference-convex (DC) algorithm (Kim et al., 2008; Shen et al., 2012) and the modified local quadratic approximation algorithm (MLQA) (Lee et al., 2016). The main contribution of the package **ncpen** is that it encompasses most of existing non-convex penalties, including the truncated ℓ_1 (Shen et al., 2013), log (Zou and Li, 2008; Friedman, 2012), bridge (Huang et al., 2008), moderately clipped LASSO (Kwon et al., 2015), sparse ridge (Huang et al., 2016a; Kwon et al., 2013) penalties as well as the SCAD and MCP and covers a broader range of regression models: multinomial and Cox models as well as the GLM. Further, **ncpen** provides two unique options: the investigation of initial dependent solution paths and non-standardization of input variables, which allow the users more flexibility.

The rest of the paper is organized as follows. In the next section, we describe the algorithm implemented in **ncpen** with major steps and details. Afterwards, the main functions and various options in **ncpen** are presented with numerical illustrations. Finally, the paper concludes with remarks.

[*]Co-first author: D. Kim and S. Lee equally contributed to this work
[†]Corresponding author

An algorithm for the non-convex penalized estimation

We consider the problem of minimizing

$$Q_\lambda(\boldsymbol{\beta}) = L(\boldsymbol{\beta}) + \sum_{j=1}^{p} J_\lambda\left(|\beta_j|\right), \tag{1}$$

where $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_p)^T$ is a p-dimensional parameter vector of interest, L is a convex loss function and J_λ is a non-convex penalty with tuning parameter $\lambda > 0$. We first introduce the CCCP-MLQA algorithm for minimizing Q_λ when λ is fixed, and then explain how to construct the whole solution path over a decreasing sequence of λs by using the algorithm.

A class of non-convex penalties

We consider a class of non-convex penalties that satisfy $J_\lambda(|t|) = \int_0^{|t|} \nabla J_\lambda(s)\, ds, t \in \mathbb{R}$ for some non-decreasing function ∇J_λ and

$$D_\lambda(t) = J_\lambda(|t|) - \kappa_\lambda |t| \tag{2}$$

is concave function, where $\kappa_\lambda = \lim_{t \to 0+} \nabla J_\lambda(t)$. The class includes most of existing non-convex penalties: SCAD (Fan and Li, 2001),

$$\nabla J_\lambda(t) = \lambda I[0 < t < \lambda] + \{(\tau\lambda - t)/(\tau - 1)\} I[\lambda \le t < \tau\lambda]$$

for $\tau > 2$, MCP (Zhang, 2010),

$$\nabla J_\lambda(t) = (\lambda - t/\tau) I[0 < t < \tau\lambda]$$

for $\tau > 1$, truncated ℓ_1-penalty (Shen et al., 2013),

$$\nabla J_\lambda(t) = \lambda I[0 < t < \tau]$$

for $\tau > 0$, moderately clipped LASSO (Kwon et al., 2015),

$$\nabla J_\lambda(t) = (\lambda - t/\tau)[0 < t < \tau(\lambda - \gamma)] + \gamma[t \ge \tau(\lambda - \gamma)]$$

for $\tau > 1$ and $0 \le \gamma \le \lambda$, sparse ridge (Kwon et al., 2013),

$$\nabla J_\lambda(t) = (\lambda - t/\tau) I[0 < t < \tau\lambda/(\tau\gamma + 1)] + \gamma t I[t \ge \tau\lambda/(\tau\gamma + 1)]$$

for $\tau > 1$ and $\gamma \ge 0$, modified log (Zou and Hastie, 2005).

$$\nabla J_\lambda(t) = (\lambda/\tau)[0 < t < \tau] + (\lambda/t)[t \ge \tau]$$

for $\tau > 0$, and modified bridge (Huang et al., 2008)

$$\nabla J_\lambda(t) = \left(\lambda/2\sqrt{\tau}\right)[0 < t < \tau] + (\lambda/2\sqrt{t})[t \ge \tau]$$

for $\tau > 0$.

The moderately clipped LASSO and sparse ridge are simple smooth interpolations between the MCP (near the origin) and the LASSO and ridge, respectively. The log and bridge penalties are modified to be linear over $t \in (0, \tau]$ so that they have finite right derivative at the origin. See the plot for graphical comparison of the penalties introduced here.

CCCP-MLQA algorithm

The CCCP-MLQA algorithm iteratively conducts two main steps: CCCP (Yuille and Rangarajan, 2003) and MLQA (Lee et al., 2016) steps. The CCCP step decomposes the penalty J_λ as in (2) and then minimizes the tight convex upper bound obtained from a linear approximation of D_λ. The MLQA step first minimizes a quadratic approximation of the loss L and then modifies the solution to keep descent property.

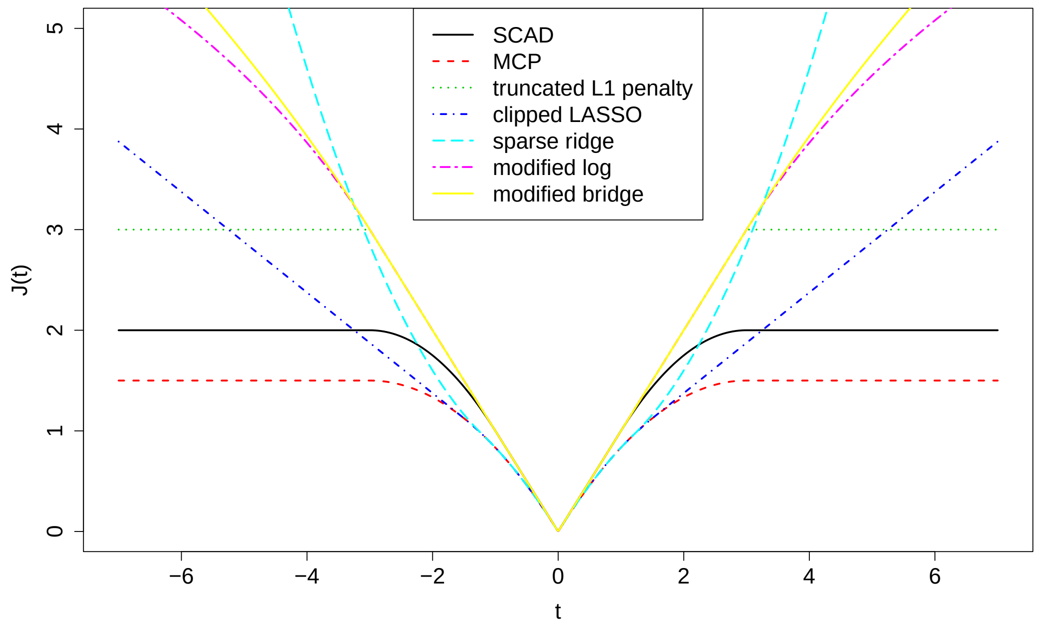


Figure 1: Plot of various penalties with $\lambda = 1, \tau = 3$ and $\gamma = 0.5$.

Concave-convex procedure

The objective function Q_λ in (1) can be rewritten by using the decomposition in (2) as

$$Q_\lambda(\boldsymbol{\beta}) = L(\boldsymbol{\beta}) + \sum_{j=1}^{p} D_\lambda(\beta_j) + \kappa_\lambda \sum_{j=1}^{p} |\beta_j| \tag{3}$$

so that $Q_\lambda(\boldsymbol{\beta})$ becomes a sum of convex, $L(\boldsymbol{\beta}) + \kappa_\lambda \sum_{j=1}^{p} |\beta_j|$, and concave, $\sum_{j=1}^{p} D_\lambda(\beta_j)$, functions. Hence the tight convex upper bound of $Q_\lambda(\boldsymbol{\beta})$ (Yuille and Rangarajan, 2003) becomes

$$U_\lambda(\boldsymbol{\beta}; \tilde{\boldsymbol{\beta}}) = L(\boldsymbol{\beta}) + \sum_{j=1}^{p} \partial D_\lambda(\tilde{\beta}_j)\beta_j + \kappa_\lambda \sum_{j=1}^{p} |\beta_j|, \tag{4}$$

where $\tilde{\boldsymbol{\beta}} = (\tilde{\beta}_1, \ldots, \tilde{\beta}_p)^T$ is a given point and $\partial D_\lambda(\tilde{\beta}_j)$ is a subgradient of $D_\lambda(\beta_j)$ at $\beta_j = \tilde{\beta}_j$. Algorithm 1 summarizes the CCCP step for minimizing Q_λ.

Algorithm 1: minimizing $Q_\lambda(\boldsymbol{\beta})$

```
1. Set $\tilde{\boldsymbol{\beta}}$.
2. Update $\tilde{\boldsymbol{\beta}}$ by $\tilde{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} U_\lambda(\boldsymbol{\beta}; \tilde{\boldsymbol{\beta}})$.
3. Repeat the Step 2 until convergence.
```

Modified Local quadratic approximation

Algorithm 1 includes minimizing $U_\lambda(\boldsymbol{\beta}; \tilde{\boldsymbol{\beta}})$ in (4) given a solution $\tilde{\boldsymbol{\beta}}$. An easy way is iteratively minimizing local quadratic approximation (LQA) of L around $\tilde{\boldsymbol{\beta}}$:

$$L(\boldsymbol{\beta}) \approx \tilde{L}(\boldsymbol{\beta}; \tilde{\boldsymbol{\beta}}) = L(\tilde{\boldsymbol{\beta}}) + \nabla L(\tilde{\boldsymbol{\beta}})^T (\boldsymbol{\beta} - \tilde{\boldsymbol{\beta}}) + (\boldsymbol{\beta} - \tilde{\boldsymbol{\beta}})^T \nabla^2 L(\tilde{\boldsymbol{\beta}}) (\boldsymbol{\beta} - \tilde{\boldsymbol{\beta}}) / 2,$$

where $\nabla L(\boldsymbol{\beta}) = \partial L(\boldsymbol{\beta}) / \partial \boldsymbol{\beta}$ and $\nabla^2 L(\boldsymbol{\beta}) = \partial^2 L(\boldsymbol{\beta}) / \partial \boldsymbol{\beta}^2$. Then $U_\lambda(\boldsymbol{\beta}; \tilde{\boldsymbol{\beta}})$ can be minimized by iteratively minimizing

$$\tilde{U}_\lambda(\boldsymbol{\beta}; \tilde{\boldsymbol{\beta}}) = \tilde{L}(\boldsymbol{\beta}; \tilde{\boldsymbol{\beta}}) + \sum_{j=1}^{p} \partial D_\lambda(|\tilde{\beta}_j|)\beta_j + \kappa_\lambda \sum_{j=1}^{p} |\beta_j|. \tag{5}$$

It is easy to minimize $\tilde{U}_\lambda\left(\boldsymbol{\beta};\tilde{\boldsymbol{\beta}}\right)$ since it is simply a quadratic function and the penalty term is the LASSO. For the algorithm, we use the coordinate descent algorithm introduced by Friedman et al. (2007). Note that the LQA algorithm may not have the descent property. Hence, we incorporate the modification step to ensure the descent property. Let $\tilde{\boldsymbol{\beta}}^a$ be the minimizer of $\tilde{U}_\lambda\left(\boldsymbol{\beta};\tilde{\boldsymbol{\beta}}\right)$. Then we modify the solution $\tilde{\boldsymbol{\beta}}^a$ by $\tilde{\boldsymbol{\beta}}^{\hat{h}}$ whenever it violates the descent property, i.e., $U_\lambda(\tilde{\boldsymbol{\beta}}^a;\tilde{\boldsymbol{\beta}}) > U_\lambda\left(\tilde{\boldsymbol{\beta}};\tilde{\boldsymbol{\beta}}\right)$:

$$\tilde{\boldsymbol{\beta}}^{\hat{h}} = \hat{h}\tilde{\boldsymbol{\beta}}^a + \left(1-\hat{h}\right)\tilde{\boldsymbol{\beta}}, \tag{6}$$

where $\hat{h} = \arg\min_{h>0} U_\lambda\left(h\tilde{\boldsymbol{\beta}}^a + (1-h)\,\tilde{\boldsymbol{\beta}};\tilde{\boldsymbol{\beta}}\right)$. This modification step in (6) guarantees the descent property of the LQA algorithm (Lee et al., 2016).

```
Algorithm 2: minimizing $U_\lambda\left(\boldsymbol{\beta};\tilde{\boldsymbol{\beta}}\right)$
  1. Set $\tilde{\boldsymbol{\beta}}$.
  2. Find $\tilde{\boldsymbol{\beta}}^a = \arg\min_{\boldsymbol{\beta}} \tilde{U}_\lambda\left(\boldsymbol{\beta};\tilde{\boldsymbol{\beta}}\right)$.
  3. Find $\hat{h} = \arg\min_{h>0} U_\lambda\left(h\tilde{\boldsymbol{\beta}}^a + (1-h)\,\tilde{\boldsymbol{\beta}};\tilde{\boldsymbol{\beta}}\right)$.
  4. Update $\tilde{\boldsymbol{\beta}}$ by $\hat{h}\tilde{\boldsymbol{\beta}}^a + \left(1-\hat{h}\right)\tilde{\boldsymbol{\beta}}$.
  5. Repeat the Step 2–4 until convergence.
```

Efficient path construction over λ

Usually, the computation time of the algorithm rapidly increases as the number of non-zero parameters increases or λ decreases toward zero. To accelerate the algorithm, we incorporate the active-set-control procedure while constructing the solution path over a decreasing sequence of λ.

Assume that λ is given and we have an initial solution $\tilde{\boldsymbol{\beta}}$ which is expected to be very close to the minimizer of $Q_\lambda\left(\boldsymbol{\beta}\right)$. First we check the first order KKT optimality conditions:

$$\partial Q_\lambda\left(\tilde{\boldsymbol{\beta}}\right)/\partial\beta_j = 0, j\in\mathcal{A} \text{ and } |\partial Q_\lambda\left(\tilde{\boldsymbol{\beta}}\right)/\partial\beta_j| \le \kappa_\lambda, j\in\mathcal{N}, \tag{7}$$

where $\mathcal{A} = \{j : \tilde{\beta}_j \neq 0\}$ and $\mathcal{N} = \{j : \tilde{\beta}_j = 0\}$. We stop the algorithm if the conditions are satisfied else update $\mathcal{N}$ and $\tilde{\boldsymbol{\beta}}$ by $\mathcal{N} = \mathcal{N}\setminus\{j_{\max}\}$ and

$$\tilde{\boldsymbol{\beta}} = \arg\min_{\beta_j=0, j\in\mathcal{N}} Q_\lambda\left(\boldsymbol{\beta}\right), \tag{8}$$

respectively, where $j_{\max} = \arg\max_{j\in\mathcal{N}}|\partial Q_\lambda\left(\tilde{\boldsymbol{\beta}}\right)/\partial\beta_j|$. We keep these iterations until the KKT conditions in (7) are satisfied with $\tilde{\boldsymbol{\beta}}$. The key step is (8) which is easy and fast to obtain by using Algorithm 1 and 2 since the objective function only includes the parameters in $\mathcal{A}\cup\{j_{\max}\}$.

```
Algorithm 3: minimizing $Q_\lambda\left(\boldsymbol{\beta}\right)$
  1. Set $\tilde{\boldsymbol{\beta}}$.
  2. Set $\mathcal{A} = \{j : \tilde{\beta}_j \neq 0\}$ and $\mathcal{N} = \{j : \tilde{\beta}_j = 0\}$.
  3. Check whether $\partial Q_\lambda\left(\tilde{\boldsymbol{\beta}}\right)/\partial\beta_j = 0, j\in\mathcal{A}$ and $|\partial Q_\lambda\left(\tilde{\boldsymbol{\beta}}\right)/\partial\beta_j| \le \kappa_\lambda, j\in\mathcal{N}$.
  4. Update $\mathcal{N}$ by $\mathcal{N}\setminus\{j_{\max}\}$, where $j_{\max} = \arg\max_{j\in\mathcal{N}}|\partial Q_\lambda\left(\tilde{\boldsymbol{\beta}}\right)/\partial\beta_j|$.
  5. Update $\tilde{\boldsymbol{\beta}}$ by $\tilde{\boldsymbol{\beta}} = \arg\min_{\beta_j=0,j\in\mathcal{N}} Q_\lambda\left(\boldsymbol{\beta}\right)$.
  6. Repeat the Step 2–5 until the KKT conditions satisfy.
```

Remark 1 *The number of variables that violates the KKT conditions could be large for some high-dimensional cases. In this case, it may be inefficient to add only one variable $j_{\max}$ into $\mathcal{A}$. It would be more efficient to add more variables into $\mathcal{A}$. However, when the number variables added is too large, it also is inefficient. With many experiences, we found that the algorithm would be efficient with 10 variables.*

In practice, we want to approximate the whole solution path or surface of the minimizer $\hat{\boldsymbol{\beta}}^\lambda$ as a function of λ. For the purpose, we first construct a decreasing sequence $\lambda_{\max} = \lambda_0 > \lambda_1 > \cdots > \lambda_{n-1} > \lambda_n = \lambda_{\min}$ and then obtain the corresponding sequence of minimizers $\hat{\boldsymbol{\beta}}^{\lambda_0}, \ldots, \hat{\boldsymbol{\beta}}^{\lambda_n}$. Let $\partial Q_\lambda\left(\mathbf{0}\right)$ be the subdifferential of Q_λ at $\mathbf{0}$. Then we can see that $\mathbf{0} \in \partial Q_\lambda\left(\mathbf{0}\right) = \{\nabla L\left(\mathbf{0}\right) + \boldsymbol{\delta} : \max_j|\delta_j| \le \kappa_\lambda\}$, for any $\kappa_\lambda > \kappa_{\lambda_{\max}} = \max_j|\partial L\left(\mathbf{0}\right)/\partial\beta_j|$, which implies the p-dimensional zero vector is the exact minimizer of $Q_\lambda\left(\boldsymbol{\beta}\right)$ when $\kappa_\lambda \ge \kappa_{\lambda_{\max}}$. Hence, we start from the largest value $\lambda = \lambda_{\max}$ that satisfies $\kappa_{\lambda_{\max}} = \max_j|\partial L\left(\mathbf{0}\right)/\partial\beta_j|$, and then we continue down to $\lambda = \lambda_{\min} = \epsilon\lambda_{\max}$, where ϵ is a predetermined ratio such as $\epsilon = 0.01$. Once we obtain the minimizer $\hat{\boldsymbol{\beta}}^{\lambda_{k-1}}$ then it is easy to find $\hat{\boldsymbol{\beta}}^{\lambda_k}$ by using $\hat{\boldsymbol{\beta}}^{\lambda_{k-1}}$ as an initial solution in Algorithm 3, which is expected to be close to $\hat{\boldsymbol{\beta}}^{\lambda_k}$ for a finely divided λ sequence. This scheme is called the *warm start strategy*, which makes the algorithm more stable and efficient (Friedman et al., 2010).

The R package ncpen

In this section, we introduce the main functions with various options and user-friendly functions implemented in the package **ncpen** for the users. Next section will illustrate how the various options in the main function make a difference in data analysis through numerical examples.

The R package **ncpen** contains the main functions: `ncpen()` for fitting various nonconvex penalized regression models, `cv.ncpen()` and `gic.ncpen()` for selecting the optimal model from a sequence of the regularization path based on cross-validation and a generalized information criterion(Wang et al., 2007, 2009; Fan and Tang, 2013; Wang et al., 2013). In addition, the useful functions are also implemented in the package: `sam.gen.ncpen()` for generating a synthetic data from various models with the correlation structure in (11), `plot()` for graphical representation, `coef()` for extracting coefficients from the fitted object, `predict()` for making predictions from new design matrix. The followings are the basic usage of the main functions:

```
## linear regression with scad penalty
n=100; p=10
sam = sam.gen.ncpen(n=n,p=p,q=5,cf.min=0.5,cf.max=1,corr=0.5,family="gaussian")
x.mat = sam$x.mat; y.vec = sam$y.vec
fit = ncpen(y.vec=y.vec,x.mat=x.mat,family="gaussian",penalty="scad")
coef(fit); plot(fit)

# prediction from a new dataset
test.n=20
newx = matrix(rnorm(test.n*p),nrow=test.n,ncol=p)
predict(fit,new.x.mat=newx)

# selection of the optimal model based on the cross-validation
cv.fit = cv.ncpen(y.vec=y.vec,x.mat=x.mat,family="gaussian",penalty="scad")
coef(cv.fit)

# selection of the optimal model using the generalized information criterion
fit = ncpen(y.vec=y.vec,x.mat=x.mat,family="gaussian",penalty="scad")
gic.ncpen(fit)
```

The main function `ncpen()` provides various options which produce different penalized estimators. The other packages for nonconvex penalized regressions also have similar options: ℓ_2-regularization and penalty weights. However, the package **ncpen** provides the two unique options for standaradization and initial value. Below, we briefly describe the options in the main function `ncpen()`.

Ridge regularization

The option `alpha` in the main functions forces the algorithm to solve the following penalized problem with the ℓ_2-regularization or ridge effect (Zou and Hastie, 2005).

$$Q_\lambda(\boldsymbol{\beta}) = \frac{1}{n}\sum_{i=1}^{n} \ell_i(\boldsymbol{\beta}) + \alpha \sum_{j=1}^{p} J_\lambda(|\beta_j|) + (1-\alpha)\lambda \sum_{j=1}^{p} \beta_j^2, \tag{9}$$

where $\alpha \in [0,1]$ is the value from the option `alpha`, which is the mixing parameter between the penalties J_λ and ridge. The objective function in (9) includes the elastic net (Zou and Hastie, 2005) when $J_\lambda(t) = \lambda t$ and Mnet (Huang et al., 2016a) when $J_\lambda(\cdot)$ is the MC penalty. By controlling the option `alpha`, we can treat the problem with highly correlated variables, and it makes the algorithm more stable also since the minimum eigenvalue of the Hessian marix becomes large up to factor $(1-\alpha)\lambda$ (Zou and Hastie, 2005; Lee and Breheny, 2015; Huang et al., 2016a).

Observation and penalty weights

We can give different weights for each observation and penalty by the options `obs.weight` and `pen.weight`, which provides the minimizer of

$$Q_\lambda(\boldsymbol{\beta}) = \sum_{i=1}^{n} d_i \ell_i(\boldsymbol{\beta}) + \sum_{j=1}^{p} w_j J_\lambda(|\beta_j|), \tag{10}$$

where d_i is the weight for the ith observation and w_j is the penalty weight for the jth variable. For example, controlling observation weights is required for the linear regression model with heteroscedastic error variance. Further, we can compute adaptive versions of penalized estimators by giving different penalty weights as in the adaptive LASSO (Zou, 2006).

Standardization

It is common practice to standardize variables prior to fitting the penalized models, but one may opt not to. Hence, we provide the option `x.standardize` for flexible analysis. The option `x.standardize=TRUE` means that the algorithm solves the original penalized problem in (1), with the standardized (scaled) variables, and then the resulting solution $\hat{\beta}_j$ is converted to the original scale by $\hat{\beta}_j/s_j$, where $s_j = \sum_{i=1}^{n} x_{ij}^2/n$. When the penalty J_λ is the LASSO penalty, this procedure is equivalent to solving following penalized problem

$$Q_\lambda^s(\boldsymbol{\beta}) = L(\boldsymbol{\beta}) + \sum_{j=1}^{p} \lambda_j |\beta_j|,$$

where $\lambda_j = \lambda s_j$, which is another adaptive version of the LASSO being different from the adaptive LASSO (Zou, 2006).

Initial value

We introduced the warm start strategy for speed up the algorithm, but the solution path, in fact, depends on the initial solution of the CCCP algorithm because of the non-convexity. The option `local=TRUE` in **ncpen** provides the same initial value specified by the option `local.initial` into each CCCP iterations for whole λ values. The use of the option `local=TRUE` makes the algorithm slower but the performance of the resulting estimator would be often improved as provided a good initial such as the maximum likelihood estimator or LASSO.

Numerical illustrations

Elapsed times

We consider the linear and logistic regression models to calculate the total elapsed time for constructing the solution path over 100 λ values:

$$y = \mathbf{x}^T\boldsymbol{\beta} + \varepsilon \text{ and } \mathbf{P}(y = 1|\mathbf{x}) = \frac{exp(\mathbf{x}^T\boldsymbol{\beta})}{1 + exp(\mathbf{x}^T\boldsymbol{\beta})} \tag{11}$$

where $\mathbf{x} \sim N_p(\mathbf{0}, \Sigma)$ with $\Sigma_{jk} = 0.5^{|j-k|}$, $\beta_j = 1/j$ for $j, k = 1, \cdots, p$ and $\varepsilon \sim N(0, 1)$. The averaged elapsed times of **ncpen** in 100 random repetitions are summarized in Table 1 and 2 for various n and p, where the penalties are the SCAD, MCP, truncated ℓ_1 (TLP), moderately clipped LASSO (CLASSO), sparse ridge (SR), modified bridge (MBR) and log (MLOG). For comparison, we try **ncvreg** for the SCAD also. The results show that all methods in **ncpen** are feasible for high-dimensional data.

Standardization effect

We compare the solution paths based on the diabetes samples available from **lars** package (Efron et al., 2004), where the sample size $n = 442$ and the number of covariates $p = 64$, including quadratic and interaction terms. Figure 2 shows four plots where the top two panels draw the solution paths from the LASSO and SCAD with $\tau = 3.7$ given by **ncvreg** and bottom two panels draw those from the SCAD with $\tau = 3.7$ based on **ncpen** with and without standardization of covariates. Two solution paths from **ncvreg** and **ncpen** with standardization are almost the same since **ncvreg** standardizes the covariates by default, which is somewhat different from that of **ncpen** without standardization.

Table 1: Elapsed times for constructing the entire solution path where $p = 500$ and various n

Model	n	**ncvreg**	SCAD	MCP	TLP	CLASSO	SR	MBR	MLOG
Linear regression	200	0.0226	0.1277	0.1971	0.0333	0.0696	0.0618	0.0620	0.0476
	400	0.0329	0.1082	0.2031	0.0662	0.1041	0.1025	0.1160	0.0919
	800	0.0347	0.1008	0.1867	0.0865	0.0993	0.1067	0.1425	0.1197
	1600	0.0665	0.2035	0.3170	0.1717	0.1847	0.1983	0.2669	0.2301
	3200	0.1394	0.4341	0.6173	0.3541	0.3962	0.4161	0.5505	0.4678
	6400	0.2991	0.9853	1.2045	0.7955	0.8788	0.9066	1.2281	1.0148
Logistic regression	200	0.0565	0.0454	0.0400	0.0391	0.0148	0.0160	0.0379	0.0411
	400	0.0787	0.1113	0.0971	0.0747	0.0556	0.0608	0.0969	0.0808
	800	0.0907	0.1570	0.1623	0.1198	0.0777	0.1015	0.1511	0.1298
	1600	0.1682	0.2965	0.3007	0.2294	0.1640	0.2088	0.3002	0.2451
	3200	0.3494	0.6480	0.6258	0.4655	0.3513	0.4423	0.6395	0.5305
	6400	0.7310	1.4144	1.3711	1.0268	0.8389	1.0273	1.4445	1.1827

Table 2: Elapsed times for constructing the entire solution path where $n = 500$ and various p

Model	p	**ncvreg**	SCAD	MCP	TLP	CLASSO	SR	MBR	MLOG
Linear regression	200	0.0150	0.0733	0.2201	0.0433	0.0629	0.0981	0.0909	0.0721
	400	0.0210	0.0664	0.1588	0.0532	0.0617	0.0678	0.0941	0.0813
	800	0.0538	0.1650	0.2172	0.1107	0.1505	0.1457	0.1750	0.1383
	1600	0.0945	0.2703	0.2946	0.1793	0.2253	0.2221	0.2672	0.2045
	3200	0.1769	0.5071	0.5032	0.3379	0.3972	0.3986	0.4801	0.3684
	6400	0.3439	1.0781	1.0228	0.7366	0.8001	0.8207	1.0210	0.7830
Logistic regression	200	0.0590	0.1065	0.1029	0.0750	0.0465	0.0696	0.0978	0.0804
	400	0.0568	0.1054	0.1044	0.0753	0.0453	0.0593	0.0941	0.0809
	800	0.1076	0.1555	0.1349	0.1103	0.0873	0.0934	0.1423	0.1163
	1600	0.1327	0.1944	0.1591	0.1419	0.1122	0.1151	0.1842	0.1460
	3200	0.2073	0.3120	0.2529	0.2382	0.1885	0.1948	0.3055	0.2415
	6400	0.3843	0.5893	0.4792	0.4646	0.3539	0.3576	0.5978	0.4677

Figure 3 shows the solution paths from six penalties with standardization by default in **ncpen**: the MCP, truncated ℓ_1, modified log, bridge, moderately clipped LASSO and sparse ridge.

Ridge regularization effect

There are cases when we need to introduce the ridge penalty for some reasons, and **ncpen** provides a hybrid version of the penalties: $\alpha J_\lambda(|t|) + (1-\alpha)|t|^2$, where α is the mixing parameter between the penalty J_λ and ridge effects. For example, the non-convex penalties often produce parameters that diverge to infinity for the logistic regression because of perfect fitting. Figure 4 shows the effects of ridge penalty where the prostate tumor gene expression data in **spls** are used for illustration. The solution paths using the top 50 variables with high variances are drawn when $\alpha \in \{1, 0.7, 0.3, 0\}$ for the SCAD and modified bridge penalties. The solution paths without ridge effect ($\alpha = 1$) tend to diverge as λ decreases and become stabilized as the ridge effect increases ($\alpha \downarrow$) (Lee and Breheny, 2015).

Initial based solution path

We introduced the warm start strategy for speed up the algorithm but the solution path, in fact, depends on the initial solution because of the non-convexity. For comparison, we use the prostate tumor gene expression data and the results are displayed in Figure 5 and Table 3. In the figure, left panels show the solution paths for the SCAD, MCP and clipped LASSO obtained by the warm start, and the right panels show those obtained by using the LASSO as a global initial for the CCCP algorithm. Figure 5 shows two strategies for initial provide very different solution paths, which may result in different performances of the estimators. We compare the prediction accuracy and selectivity of the estimators by two strategies. The results are obtained by 300 random partitions of data set divided into two parts, training (70%) and test (30%) datasets. For each training data, the optimal tuning parameter values are selected by 10-fold cross-validation, and then we compute the prediction error(the percentage of the misclassified samples) on each test dataset and the number of selected

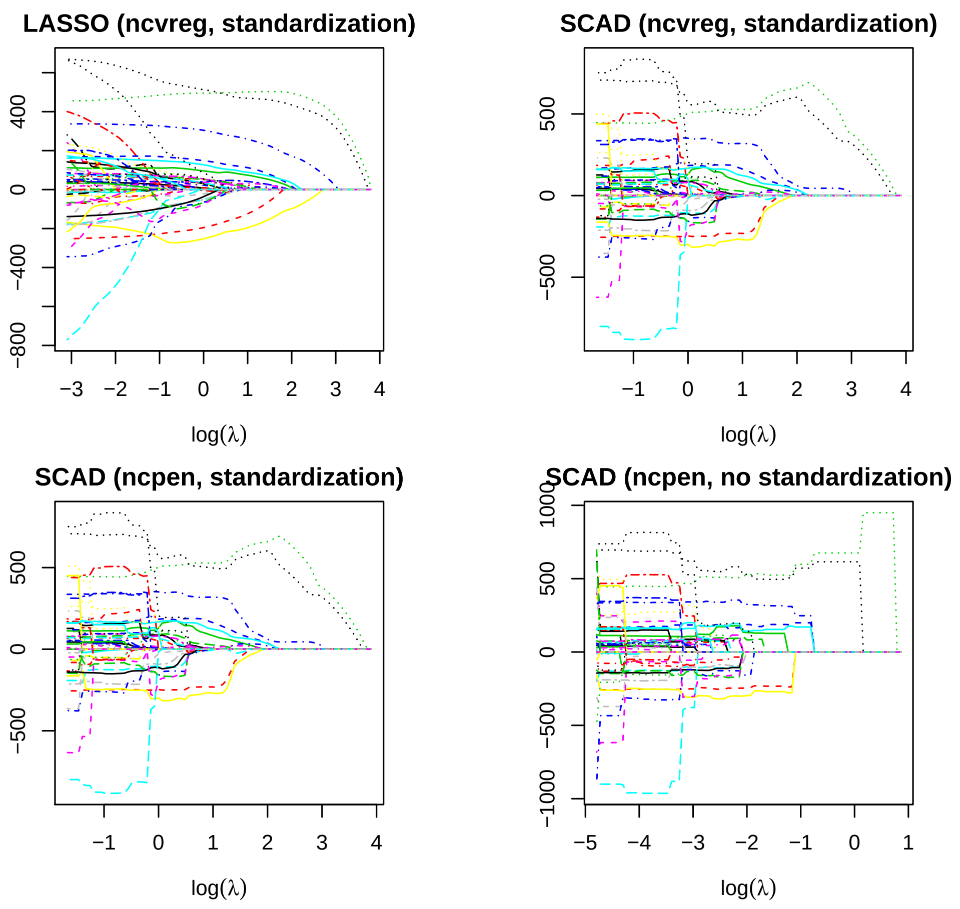


Figure 2: Solution paths from the **ncvreg** and **ncpen** for the LASSO and SCAD.

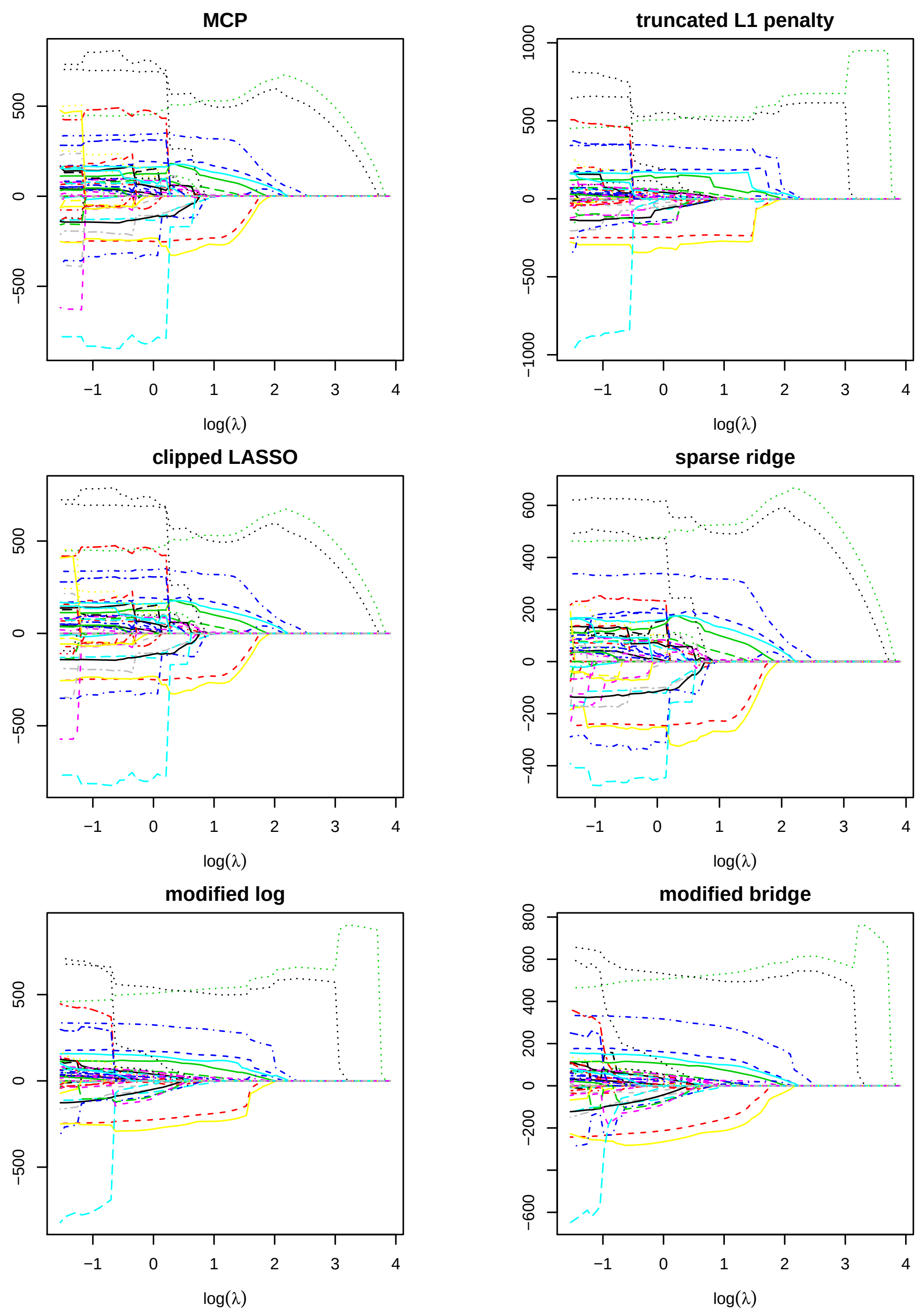


Figure 3: Solution paths from **ncpen** with six non-convex penalties.

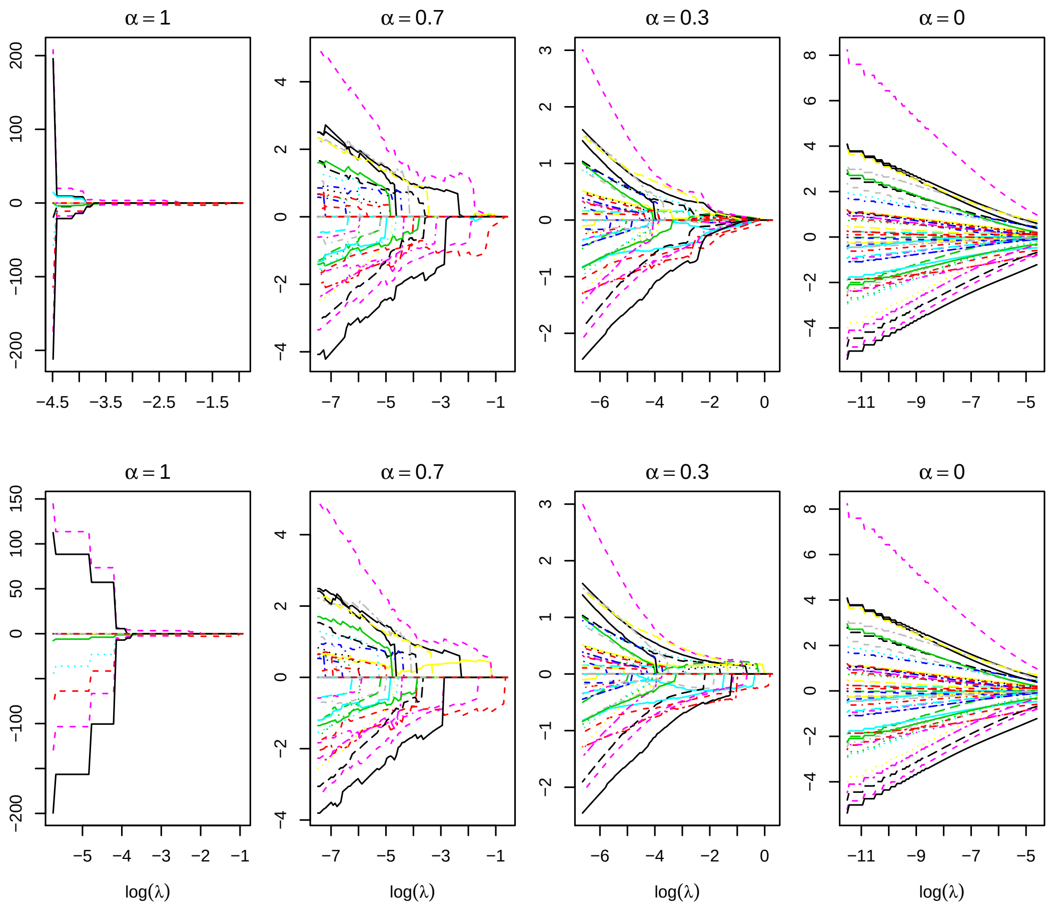


Figure 4: Solution path traces with ridge penalty. Top and bottom panels are drawn from the SCAD and modified bridge penalties, respectively.

nonzero variables on each training dataset. Table 3 shows all methods by the global initial perform better than those by the warm start strategy. In summary, the nonconvex penalized estimation depends on the initial solution, and the non-convex penalized estimator by a good initial would improve its performance.

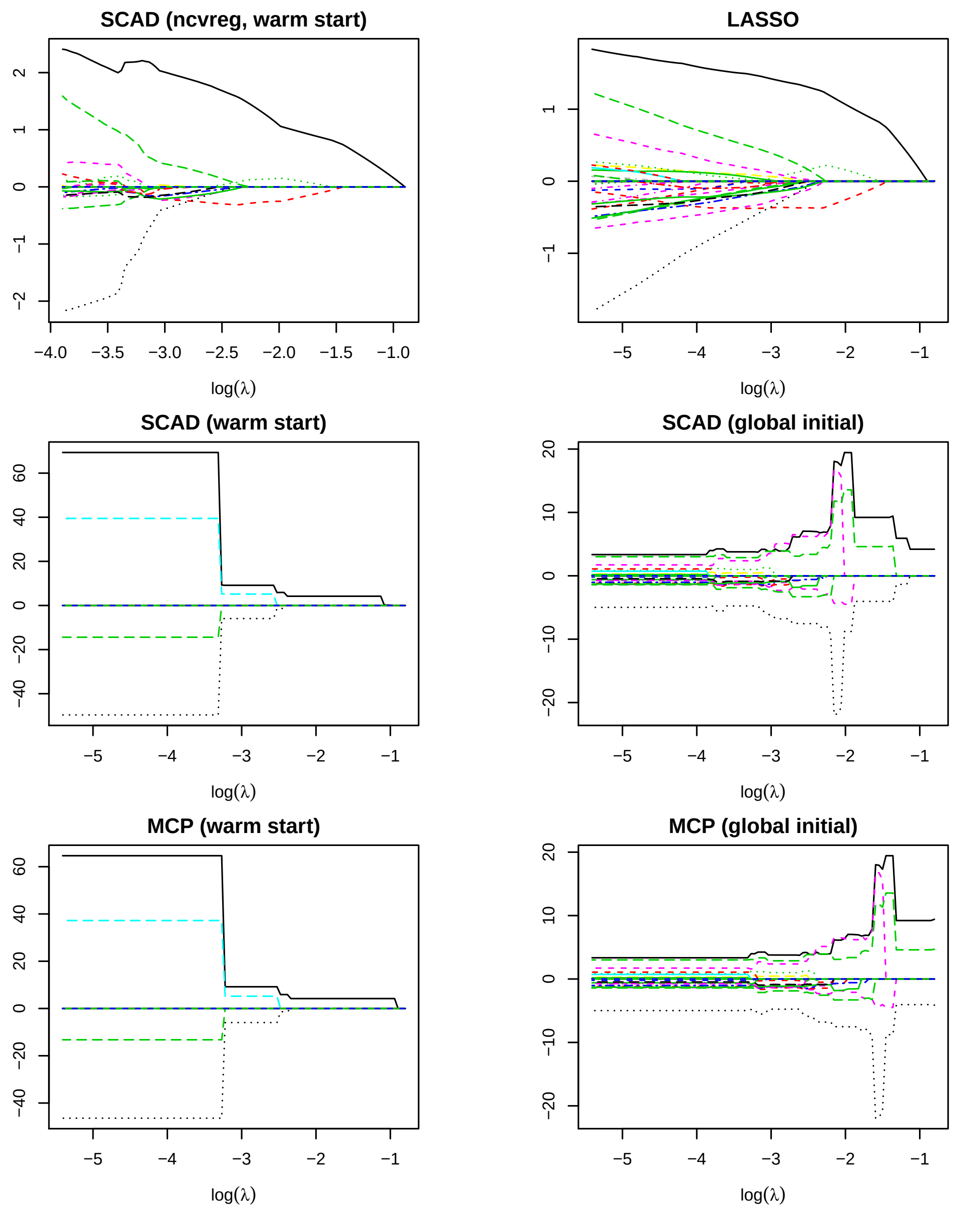


Figure 5: Solution paths of the SCAD and MCP with warm start and global initial solution.

Table 3: Comparison of the warm start and global initial strategies for each method.

	warm start		global initial	
Method	prediction error	# variables	prediction error	# variables
SCAD	9.44 (.2757)	1.19 (.0268)	9.30 (.3091)	7.02 (.3907)
MCP	9.38 (.2774)	1.14 (.0234)	8.84 (.2657)	7.41 (.3771)
TLP	9.44 (.2647)	1.18 (.0254)	7.47 (.2677)	19.01 (.1710)
CLASSO	9.12 (.2749)	4.65 (.1914)	8.20 (.2785)	8.14 (.1542)
SR	9.38 (.2875)	5.15 (.2290)	7.94 (.2677)	15.13 (.2283)
MBR	9.78 (.2621)	1.29 (.0352)	8.21 (.3004)	8.75 (.1712)
MLOG	9.51 (.2627)	1.16 (.0233)	7.66 (.2746)	15.21 (.1816)

Concluding remarks

We have developed the R package **ncpen** for estimating generalized linear models with various concave penalties. The unified algorithm implemented in **ncpen** is flexible and efficient. The package also provides various user-friendly functions and user-specific options for different penalized estimators. The package is currently available with a general public license (GPL) from the Comprehensive R Archive Network at `https://CRAN.R-project.org/package=ncpen`. Our **ncpen** package implements internal optimization algorithms implemented in C++ benefiting from **Rcpp** package (Eddelbuettel et al., 2011).

Acknowledgements

This research is supported by the Julian Virtue Professorship 2016-18 Endowment at the Pepperdine Graziadio Business School at Pepperdine University, and the National Research Foundation of Korea (NRF) funded by the Korea government (No. 2020R1I1A3071646 and 2020R1F1A1A01071036).

Bibliography

P. Breheny and J. Huang. Coordinate descent algorithms for nonconvex penalized regression, with applications to biological feature selection. *Annals of Applied Statistics*, 5(1):232, 2011. URL `https://doi.org/10.1214/10-AOAS388`. [p120]

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. URL `https://doi.org/10.18637/jss.v040.i08`. [p131]

B. Efron, T. Hastie, I. Johnstone, R. Tibshirani, et al. Least angle regression. *Annals of Statistics*, 32(2): 407–499, 2004. URL `https://www.jstor.org/stable/3448465`. [p120, 125]

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`. [p120, 121]

Y. Fan and C. Y. Tang. Tuning parameter selection in high dimensional penalized likelihood. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 75(3):531–552, 2013. URL `https://doi.org/10.1111/rssb.12001`. [p124]

J. Friedman. Fast sparse regression and classification. *International Journal of Forecasting*, 28(3):722–738, 2012. URL `https://doi.org/10.1016/j.ijforecast.2012.05.001`. [p120]

J. Friedman, T. Hastie, H. Höfling, R. Tibshirani, et al. Pathwise coordinate optimization. *Annals of Applied Statistics*, 1(2):302–332, 2007. URL `https://doi.org/10.1214/07-AOAS131`. [p120, 123]

J. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):1, 2010. URL `https://doi.org/10.18637/JSS.V033.I01`. [p124]

J. Huang, J. L. Horowitz, and S. Ma. Asymptotic properties of bridge estimators in sparse high-dimensional regression models. *Annals of Statistics*, pages 587–613, 2008. URL `https://doi.org/10.1214/009053607000000875`. [p120, 121]

J. Huang, P. Breheny, S. Lee, S. Ma, and C. Zhang. The mnet method for variable selection. *Statistica Sinica*, 10, 2016a. URL `https://doi.org/10.5705/ss.202014.0011`. [p120, 124]

J. Huang, P. Breheny, S. Lee, S. Ma, and C.-H. Zhang. The mnet method for variable selection. *Statistica Sinica*, pages 903–923, 2016b. [p120]

D. Jiang and J. Huang. Majorization minimization by coordinate descent for concave penalized generalized linear models. *Statistics and Computing*, 24(5):871–883, 2014. URL `https://doi.org/10.1007/s11222-013-9407-3`. [p120]

Y. Kim and S. Kwon. Global optimality of nonconvex penalized estimators. *Biometrika*, 99(2):315–325, 2012. URL `https://doi.org/10.1093/biomet/asr084`. [p120]

Y. Kim, H. Choi, and H.-S. Oh. Smoothly clipped absolute deviation on high dimensions. *Journal of the American Statistical Association*, 103(484):1665–1673, 2008. URL https://doi.org/10.1198/016214508000001066. [p120]

S. Kwon and Y. Kim. Large sample properties of the scad-penalized maximum likelihood estimation on high dimensions. *Statistica Sinica*, pages 629–653, 2012. URL https://doi.org/10.5705/ss.2010.027. [p120]

S. Kwon, Y. Kim, and H. Choi. Sparse bridge estimation with a diverging number of parameters. *Statistics and Its Interface*, 6(2):231–242, 2013. URL https://doi.org/10.4310/SII.2013.V6.N2.A7. [p120, 121]

S. Kwon, S. Lee, and Y. Kim. Moderately clipped lasso. *Computational Statistics & Data Analysis*, 92: 53–67, 2015. URL https://doi.org/10.1016/j.csda.2015.07.001. [p120, 121]

S. Lee. A note on standardization in penalized regressions. *Journal of the Korean Data and Information Science Society*, 26(2):505–516, 2015. URL https://doi.org/10.7465/jkdi.2015.26.2.505. [p120]

S. Lee and P. Breheny. Strong rules for nonconvex penalties and their implications for efficient algorithms in high-dimensional regression. *Journal of Computational and Graphical Statistics*, 24(4): 1074–1091, 2015. URL https://doi.org/10.1080/10618600.2014.975231. [p124, 126]

S. Lee, S. Kwon, and Y. Kim. A modified local quadratic approximation algorithm for penalized optimization problems. *Computational Statistics & Data Analysis*, 94:275–286, 2016. URL https://doi.org/10.1016/j.csda.2015.08.019. [p120, 121, 123]

R. Mazumder, J. H. Friedman, and T. Hastie. Sparsenet: Coordinate descent with nonconvex penalties. *Journal of the American Statistical Association*, 106(495):1125–1138, 2011. URL https://doi.org/10.1198/jasa.2011.tm09738. [p120]

M. Y. Park and T. Hastie. L1-regularization path algorithm for generalized linear models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 69(4):659–677, 2007. URL https://doi.org/10.1111/j.1467-9868.2007.00607.x. [p120]

X. Shen, W. Pan, and Y. Zhu. Likelihood-based selection and sharp parameter estimation. *Journal of the American Statistical Association*, 107(497):223–232, 2012. URL https://doi.org/10.1080/01621459.2011.645783. [p120]

X. Shen, W. Pan, Y. Zhu, and H. Zhou. On constrained and regularized high-dimensional regression. *Annals of the Institute of Statistical Mathematics*, 65(5):807–832, 2013. URL https://doi.org/10.1007/s10463-012-0396-3. [p120, 121]

R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Statistical Methodology)*, pages 267–288, 1996. URL https://doi.org/10.1111/j.2517-6161.1996.tb02080.x. [p120]

P. Tseng. Convergence of a block coordinate descent method for nondifferentiable minimization. *Journal of optimization theory and applications*, 109(3):475–494, 2001. URL https://doi.org/10.1023/A:1017501703105. [p120]

H. Wang, R. Li, and C.-L. Tsai. Tuning parameter selectors for the smoothly clipped absolute deviation method. *Biometrika*, 94(3):553–568, 2007. URL https://doi.org/10.1093/biomet/asm053. [p124]

H. Wang, B. Li, and C. Leng. Shrinkage tuning parameter selection with a diverging number of parameters. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 71(3):671–683, 2009. URL https://doi.org/10.1111/j.1467-9868.2008.00693.x. [p124]

L. Wang, Y. Kim, and R. Li. Calibrating non-convex penalized regression in ultra-high dimension. *Annals of Statistics*, 41(5):2505, 2013. URL https://doi.org/10.1214/13-AOS1159. [p124]

A. Yuille and A. Rangarajan. The concave-convex procedure. *Neural Computation*, 15:915–936, 2003. URL https://doi.org/10.1162/08997660360581958. [p121, 122]

C. Zhang. Nearly unbiased variable selection under minimax concave penalty. *Ann. Stat.*, 38(2): 894–942, 2010. URL https://doi.org/10.1214/09-AOS729. [p120, 121]

C.-H. Zhang and T. Zhang. A general theory of concave regularization for high-dimensional sparse estimation problems. *Statistical Science*, pages 576–593, 2012. URL https://doi.org/10.1214/12-STS399. [p120]

H. Zou. The adaptive lasso and its oracle properties. *Journal of the American Statistical Association*, 101 (476):1418–1429, 2006. URL https://doi.org/10.1198/016214506000000735. [p125]

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. URL https://doi.org/10.1111/j.1467-9868.2005.00503.x. [p120, 121, 124]

H. Zou and R. Li. One-step sparse estimates in nonconcave penalized likelihood models. *Annals of Statistics*, 36(4):1509, 2008. URL https://doi.org/10.1214/009053607000000802. [p120]

Dongshin Kim
Pepperdine Graziadio Business School
Pepperdine University
United States of America
dongshin.kim@pepperdine.edu

Sangin Lee
Department of Information and Statistics
Chungnam National University
Korea
sanginlee44@gmail.com

Sunghoon Kwon
Department of Applied Statistics
Konkuk University
Korea
shkwon0522@gmail.com

TULIP: A Toolbox for Linear Discriminant Analysis with Penalties

by Yuqing Pan, Qing Mai and Xin Zhang

Abstract Linear discriminant analysis (LDA) is a powerful tool in building classifiers with easy computation and interpretation. Recent advancements in science technology have led to the popularity of datasets with high dimensions, high orders and complicated structure. Such datasetes motivate the generalization of LDA in various research directions. The R package **TULIP** integrates several popular high-dimensional LDA-based methods and provides a comprehensive and user-friendly toolbox for linear, semi-parametric and tensor-variate classification. Functions are included for model fitting, cross validation and prediction. In addition, motivated by datasets with diverse sources of predictors, we further include functions for covariate adjustment. Our package is carefully tailored for low storage and high computation efficiency. Moreover, our package is the first R package for many of these methods, providing great convenience to researchers in this area.

Introduction

Linear discriminant analysis (LDA) is one of the most popular classification method and a cornerstone for multivariate statistics (Michie et al., 1994, e.g). Classical LDA builds a linear classifier based on p-dimensional multivariate predictor $\mathbf{X} \in \mathbb{R}^p$ to distinguish K classes and to predict the class label $Y \in \{1, \ldots, K\}$. Despite its simplicity, LDA is shown to be very accurate on many benchmark datasets (Lim et al., 2000; Dettling, 2004; Hand, 2006). Moreover, LDA is easily interpretable and is thus often used as a visualizing tool for exploratory data analysis.

In recent decades, the advancements in science and technology have enabled researchers to collect datasets with increasing sizes and complexity. Such datasets pose challenges to LDA. Four challenges that we tackle with this package are as follows. First, in research areas such as biology, genomics and psychology, we often have more predictors than samples. However, LDA is not applicable on these high-dimensional data, because sample covariance matrix becomes not invertible when the number of predictors exceeds the sample size.

Secondly, when we have a large number of predictors, variable selection is often desired such that we can obtain a sparse classifier involving only a small proportion of the variables. On one hand, Fan and Fan (2008); Bickel and Levina (2008) showed in theory that variable selection is critical for accurate classification. On the other hand, sparse classifiers much easier to interpret in practice. However, LDA generally does not perform variable selection.

Thirdly, contemporary datasets often have complicated structure that renders the linear classifier in LDA inadequate. For example, in the presence of thousands of predictors, it may be inappropriate them to model all of them with the normal distribution. Moreover, research in neuroimaging, computational biology and personalized recommendation produces data in the form of matrices (2-way tensor) or tensors. The analysis of tensor datasets requires considerable modification to the vector-based LDA model.

Last but not least, integrative analysis with multiple data sources are drawing researchers' attention recently. Co-existence of diverse data types, such as vector, matrix and tensor calls for more sophisticated models to integrate the information from them to improve classification accuracy. It is critical to model the dependence among different types of data to reduce the noise level in the data and improve prediction accuracy (Pan et al., 2019).

Motivated by these challenges, many methods have been proposed to generalize LDA to datasets with high dimensions, non-normality and/or higher order predictors. In this package, we implement six methods that generalize LDA to contemporary complicated datasets. All of them are developed under models closely related to the LDA model, and penalties are imposed to achieve classification accuracy and variable selection in high dimensions. These methods include:

1. Direct sparse discriminant analysis (DSDA): DSDA generalizes the classical LDA model to high dimensions when there are only two classes (Mai et al., 2012). It formulates high-dimensional LDA into a penalized least squares problem.
2. Regularized optimal affine discriminant (ROAD): under the same model as DSDA, ROAD fits a sparse classifier by minimizing the classification error under the ℓ_1 constraint (Fan et al., 2012).
3. Sparse optimal scoring (SOS) for binary problems: SOS is also developed under the LDA model (Clemmensen et al., 2011). It penalizes the optimal scoring problem (Hastie et al., 1994). We focus on its application in binary problems.

	DSDA	SOS	ROAD	SeSDA	MSDA	CATCH
Classes	Binary	Binary	Binary	Binary	Multi-class	Multi-class
Data type	Vector	Vector	Vector	Vector	Vector	Tensor
Model	LDA	LDA	LDA	SeLDA	LDA	TDA/CATCH
Covariate adjustment	Yes	No	No	No	Yes	Yes

Table 1: Comparison of model settings between models. SOS was originally proposed to deal with both binary and multiclass problems, but we focus on binary problems in the package. Model SeLDA stands for Semi-parametric linear discriminant analysis, which is introduced in Section 2.2.4. Model TDA/CATCH represents tensor discriminant analysis and covariate-adjusted tensor in high-dimensions, which are illustrated in Section 2.2.5 and 2.3.6.

4. Semiparametric sparse discriminant analysis (SeSDA): SeSDA assumes a semiparametric model where data transformation can be applied to alleviate the non-normality. In practice, SeSDA finds the data-driven transformation and then performs model-fitting on the transformed data (Mai and Zou, 2015).
5. Multiclass sparse discriminant analysis (MSDA): Instead of focusing on binary problems, MSDA considers the multiclass LDA model (Mai et al., 2015). It takes note of the fact that the Bayes' rule can be estimated with minimizing a quadratic loss. To account for the multiclass structure, a group lasso penalty (Yuan and Lin, 2006) is applied to achieve variable selection.
6. Covariate-adjusted tensor classification in high-dimensions (CATCH): CATCH (Pan et al., 2019) is developed for tensor predictors. It takes advantage of the tensor structure to significantly reduce the number of parameters and hence alleviate computation complexity.

See Table 1 for a comparison of these methods. Despite their different model assumptions and formulas, all of them have strong theoretical support and excellent empirical performance. We further note that they can be combined with covariate adjustment when multiple data sources are available. Our package **TULIP** (Pan et al., 2021) integrates diverse discriminant analysis models and supportive functions to make it a convenient and well-equipped toolbox. It has several notable advantages. First, we not only include functions for model fitting, but also cross validation functions for easy control of the sparsity level, and prediction functions for the prediction of future observations. In addition, we provide covariate adjustment functions that efficiently remove heterogeneity in the predictors and combine information from covariates. Second, our package greatly facilitates the application of DSDA, ROAD and SeSDA for R users, as they do not have public R packages on CRAN outside ours. Third, although MSDA and SOS have been implemented in packages **msda** (Mai et al., 2015) and **sparseLDA** (Clemmensen and Kuhn, 2016), we carefully modify their algorithms in our implementation to lower storage cost and/or speed up computation.

We acknowledge that many other efforts have been spent on topics closely related to that of our paper. On one hand, by now a large number of high-dimensional discriminant analysis methods have been developed. Some excellent examples include Fan and Fan (2008); Tibshirani et al. (2002); Trendafilov and Jolliffe (2007); Fan et al. (2012); Wu et al. (2009); Cai et al. (2011); Shao et al. (2011); Clemmensen et al. (2011); Witten and Tibshirani (2011); Xu et al. (2015); Niu et al. (2015). On the other hand, in the literature, many works study matrix/tensor regression and classification methods. Many of them impose low rank assumption (Zhou et al., 2013; Kolda and Bader, 2009; Chi and Kolda, 2012; Liu et al., 2017; Li and Schonfeld, 2014; Lai et al., 2013; Zhong and Suslick, 2015; Zeng et al., 2015). All these methods have been reported to have great performance, but a comprehensive study of them is apparently out of the scope of our current paper.

The rest of this paper is organized as follows. We start with a brief overview of discriminant analysis models in Section 2.2. Model estimation and implementation details are discussed in Section 2.3. Section 2.4 contains instructions and examples on the usage of the package. A real data example is given in Section 2.5 to confirm the numerical performance of methods in the package.

Discriminant analysis models and Bayes rules

Bayes rule for classification

Recall that $Y \in \{1, \ldots, K\}$ is the categorical response (class indicator), and we use the generic $\mathcal{X}$ to denote the predictor and (potential) additional covariate. Specifically, $\mathcal{X} = \mathbf{X} \in \mathbb{R}^p$ in classical multivariate discriminant analysis; $\mathcal{X} = \mathbf{X} \in \mathbb{R}^{p_1 \times \cdots \times p_M}$ in tensor discriminant analysis; and $\mathcal{X} = (\mathbf{X}, \mathbf{U})$ in covariate-adjusted classification settings, where $\mathbf{U} \in \mathbb{R}^q$ is additional covariates and $\mathbf{X}$ can be either vector or tensor. Our goal is to construct the optimal classifier to distinguish and predict Y based on $\mathcal{X}$ under various settings. Denote $\pi_k = \Pr(Y = k)$ and f_k as the conditional distribution of $\mathcal{X}$ within Class k (e.g. $f_k(\mathcal{X}) = f(\mathbf{X}, \mathbf{U} \mid Y = k)$ is the joint distribution of $\mathbf{X}$ and $\mathbf{U}$ given $Y = k$, in presence of $\mathbf{U}$). The optimal classifier, often referred to as the Bayes rule, is thus

$$\delta(\mathbf{X}) = \arg\max_k \{\log \pi_k + \log f_k(\mathcal{X})\}. \tag{1}$$

The Bayes rule achieves the lowest classification error possible (Friedman et al., 2001). Therefore, it is our ultimate goal to estimate the Bayes rule. However, additional model assumptions are often needed for f_k to ensure statistical and computational efficiency. Consider the classical LDA setting of $\mathbf{X} \in \mathbb{R}^p$ and $Y \in \{1, \ldots, K\}$. To gain intuition, we often assume that within each class, the predictor follows a normal distribution with different means and a common covariance matrix. Then the Bayes rule is a linear function of $\mathbf{X}$ and can be straightforwardly estimated.

In the rest of this section, we discuss various statistical models that have been widely studied in the literature, along with the Bayes rules under these assumptions. Specifically, we review the classical LDA model, the semiparametric LDA model, and the tensor discriminant analysis model. We also discuss a general framework for covariate adjustment.

The linear discriminant analysis model (LDA)

Given a multivariate predictor $\mathbf{X} \in \mathbb{R}^p$ and $Y \in \{1, \ldots, K\}$, the LDA model assumes that $\mathbf{X}$ is normally distributed within each class, i.e,

$$\mathbf{X} \mid (Y = k) \sim N(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}), \quad \Pr(Y = k) = \pi_k, \quad k = 1, \ldots, K, \tag{2}$$

where $\boldsymbol{\mu}_k \in \mathbb{R}^p$ is the mean of $\mathbf{X}$ within class k, and $\boldsymbol{\Sigma} \in \mathbb{R}^{p \times p}$ is the common within class covariance matrix.

Define $\boldsymbol{\beta}_k = \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_k - \boldsymbol{\mu}_1)$ for $k = 1, \cdots, K$. The Bayes' rule turns out to be a linear function:

$$\widehat{Y} = \arg\max_k \Pr(Y = k \mid \mathbf{X}) = \arg\max_{k=1,\ldots,K} \{\log \pi_k + \boldsymbol{\beta}_k^T(\mathbf{X} - \boldsymbol{\mu}_k/2)\}. \tag{3}$$

The LDA model is simple yet elegant. All the parameters in this model have natural interpretations, while the Bayes rule has a nice linear form. An interesting fact about the Bayes rule in (3) is that it does not explicitly involve the p^2-dimensional parameter $\boldsymbol{\Sigma}^{-1}$. Instead, $\boldsymbol{\Sigma}^{-1}$ is only implicitly included in the discriminant directions $\boldsymbol{\beta}_k$. Moreover, it can be shown that the Bayes rule is equivalent to first reducing data to $\mathbf{X}^T\boldsymbol{\beta}_2, \ldots \mathbf{X}^T\boldsymbol{\beta}_K$ and then fitting the LDA model on the $(K-1)$-dimensional space. Therefore, to estimate the Bayes rule in high dimensions, our interest centers on the estimation of $\boldsymbol{\beta}_k$. We assume that $\boldsymbol{\beta}_k$'s are sparse with many elements being zero. Enforcement of this sparsity assumption will facilitate our estimation and naturally lead to variable selection.

Although the Bayes rule is derived under the somewhat restrictive normality and equal covariance assumptions, the discriminant directions $\boldsymbol{\beta}_k$ are still meaningful when data are non-normal, thanks to their geometric properties. It can be shown that, if we project $\mathbf{X}$ to $\boldsymbol{\beta}_k, k = 1, \ldots, K$, the separation between classes is maximized over all possible sets of $K-1$ linear projections. Consequently, the LDA model is reasonably resistant to model misspecification. However, in some of the cases where the LDA model assumptions are severely violated, one can resort to more flexible models. For example, the quadratic discriminant analysis model (Jiang et al., 2015; Fan et al., 2015; Li and Shao, 2015; Sun and Zhao, 2015) relaxes the equal covariance assumption, while severely non-normal data can be modeled by the semiparametric model to be discussed in Section 2.2.4.

Covariates adjustment

In many real-life problems, we have additional covariates along with the predictors. The covariates play two roles in the classification: it has predictive power on it own, and it also accounts part of the variation in the predictors. For example, in genomics studies, we record not only gene expression levels but also age and clinical measurements. In this case, we may view the gene expression levels as the high-dimensional predictor, and the age and clinical measurements as the covariates. We consider an LDA-type model to incorporate the covariates. In addition to the response Y and the predictor $\mathbf{X}$, we denote the covariates as $\mathbf{U} \in \mathbb{R}^q$. We assume that

$$\mathbf{U} \mid (Y = k) \sim N(\boldsymbol{\phi}_k, \boldsymbol{\Psi}), \tag{4}$$

$$\mathbf{X} \mid (\mathbf{U} = \mathbf{u}, Y = k) \sim N(\boldsymbol{\mu}_k + \boldsymbol{\alpha}\mathbf{u}, \boldsymbol{\Sigma}), \tag{5}$$

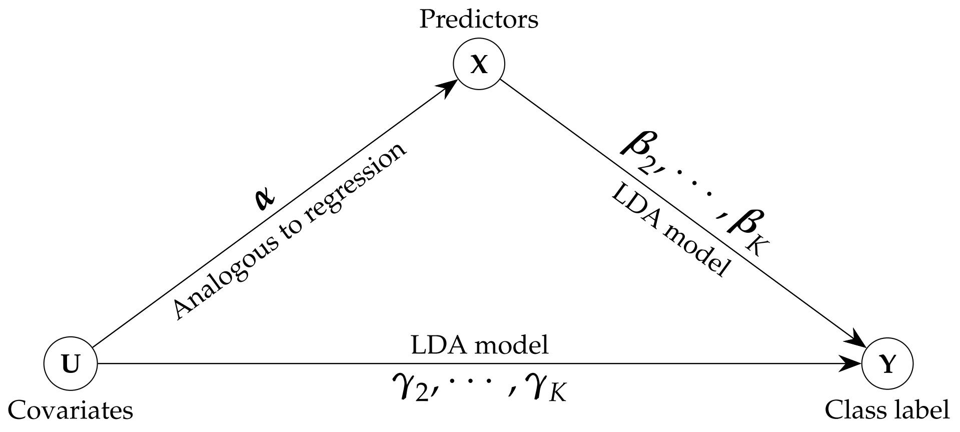


Figure 1: Graphical illustration of the direct and indirect effects. The direct effect of covariate $\mathbf{U}$ on Y follows classical discriminant analysis model measured by $\{\gamma_2, \ldots, \gamma_K\}$. Meanwhile, $\mathbf{U}$ also affects class label through affecting $\mathbf{X}$. Therefore we have $\widehat{Y} = f(\mathbf{X}, \mathbf{U})$.

where $\boldsymbol{\phi}_k \in \mathbb{R}^q$ is the within-class mean, $\boldsymbol{\Psi} \in \mathbb{R}^{q\times q}$, $\boldsymbol{\Psi} > 0$ is the common within class covariance matrix of covariates, and $\boldsymbol{\alpha} \in \mathbb{R}^{p\times q}$ is the dependence of $\mathbf{X}$ on $\mathbf{U}$. We refer to this model as the covariate-adjusted LDA (CA-LDA) model. The CA-LDA model is conceptually similar to the CATCH model (Pan et al., 2019) for tensor, which is to be introduced in Section 2.2.5, but the CA-LDA model focuses on vector predictor $\mathbf{X}$ rather than tensor predictor.

Obviously, the CA-LDA model reduces to the LDA model in the absence of covariates. With the covariates, the CA-LDA model continues to have natural interpretations. Equation (4) indicates that $(\mathbf{U}, Y)$ marginally follow the LDA model. Equation (5) implies that the distribution of $\mathbf{X}$ not only depends on Y, but also $\mathbf{U}$ through mean dependence. Therefore, within each class, $\mathbf{X}$ is linked to $\mathbf{U}$ through a linear regression model, while, after we adjust for $\mathbf{U}$, $(\mathbf{X}, Y)$ follow the LDA model as well. See Figure 1 for a graphical illustration of the relationship among $\mathbf{X}$, $\mathbf{U}$ and Y.

Under the CA-LDA model, the Bayes' rule is

$$\widehat{Y} = \arg\max_{k=1,\ldots K} \left\{ a_k + \gamma_k^T \mathbf{U} + \boldsymbol{\beta}_k^T (\mathbf{X} - \boldsymbol{\alpha}\mathbf{U}) \right\} \tag{6}$$

where $\gamma_k = \boldsymbol{\Psi}^{-1}(\boldsymbol{\phi}_k - \boldsymbol{\phi}_1)$, $\boldsymbol{\beta}_k = \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_k - \boldsymbol{\mu}_1)$ and $a_k = \log(\pi_k/\pi_1) - \frac{1}{2}\gamma_k^T(\boldsymbol{\phi}_k + \boldsymbol{\phi}_1) - \frac{1}{2}\boldsymbol{\beta}_k^T(\boldsymbol{\mu}_k + \boldsymbol{\mu}_1))$ is a scalar that does not involve $\mathbf{X}$ or $\mathbf{U}$. Throughout this paper, we assume that $\mathbf{U}$ is low-dimensional and does not need variable selection, but $\mathbf{X}$ is high-dimensional. In the presence of covariates, $\mathbf{X}$ needs to be first adjusted to $\mathbf{X} - \boldsymbol{\alpha}\mathbf{U}$ before entering the Bayes rule. Similar to the LDA model, we assume that the coefficient of $\mathbf{X} - \boldsymbol{\alpha}\mathbf{U}$, $\boldsymbol{\beta}_k$, is sparse.

The semiparametric LDA model

Although LDA is reasonably resistant to model misspecification, we may still need more flexible models when data are heavily non-normal. The semiparametric linear discriminant analysis (SeLDA) model (Lin and Jeon, 2003) is proposed for this purpose. SeLDA assumes that there exists a set of strictly monotone univariate transformations $h_1, \ldots, h_p$ such that

$$(h_1(X_1), \cdots, h_p(X_p)) \mid (Y = k) \sim N(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}). \tag{7}$$

For identifiability, we further assume that all the diagonal elements in $\boldsymbol{\Sigma}$ are 1, and all elements in $\boldsymbol{\mu}_1$ are 0. We also use the shorthand notation $h(\mathbf{X}) = (h_1(X_1), \cdots, h_p(X_p))$. The transformation h is assumed to be unknown and needs to be estimated from data. The SeLDA model assumes that the LDA model is true up to an unknown transformation. It has the same spirit as the well-known Box-Cox transformation, with which model assumptions are relaxed by proper data mapping.

It is easy to see that the LDA model is a special case of the SeLDA model, if we restrict $h(\mathbf{X}) = \mathbf{X}$. However, in the SeLDA model, we do not impose any parametric assumptions on h, which leads to great flexibility in practice. We further review a formula for h_j that will facilitate its estimation. It can be shown that

$$h_j = \Phi^{-1} \circ F_{1j} = \Phi^{-1} \circ F_{kj} + \mu_{kj}, \tag{8}$$

where Φ is the cumulative distribution function (CDF) of the standard normal random variable, and

F_{kj} is the CDF of X_j within Class k. Equation (8) will be used in Section 2.3.4. The SeLDA model also amounts to assuming that the data follow the Gaussian copula model within each class (Klaassen and Wellner, 1997; Hoff et al., 2014; Liu et al., 2009).

Although the SeLDA model requires much weaker conditions than the LDA model, it preserves many of the desirable properties. One of them is that the Bayes rule continues to be a linear function of the transformed data $h(\mathbf{X})$:

$$\widehat{Y} = \arg \max_{k=1,\ldots,K} \{\log \pi_k + \boldsymbol{\beta}_k^T (h(\mathbf{X}) - \boldsymbol{\mu}_k/2)\}. \tag{9}$$

Consequently, just as in the LDA model, when the dimension is high, we assume that $\boldsymbol{\beta}_k$ is sparse to allow accurate estimation.

Tensor discriminant analysis (TDA) and covariate adjustment

The tensor discriminant analysis (TDA) model is proposed for classification based on tensor predictors. We first briefly introduce some standard tensor notation (Kolda and Bader, 2009). See Appendenx A for more rigorous definitions. An *M-way tensor* is denoted by a multidimensional array $\mathbf{A} \in \mathbb{R}^{p_1 \times \cdots \times p_M}$ where $M \geq 2$, $p_1, \ldots, p_M$ are all positive integers. We often need to multiply an M-way tensor $\mathbf{C}$ by M matrices along each mode $\mathbf{G}_i, i = 1, \ldots, M$, denoted by $[\![\mathbf{C}; \mathbf{G}_1, \ldots, \mathbf{G}_M]\!]$. For example, in Figure 2 we obtain $\mathbf{A} = [\![\mathbf{C}; \mathbf{G}_1, \ldots, \mathbf{G}_3]\!]$ by multiplying a 3-way tensor $\mathbf{C}$ with matrices $\mathbf{G}_i$ along each mode. If $\mathbf{G}_i, i \neq m$ are identity matrices and $\mathbf{G}_m$ is a vector, then we write $\mathbf{C} \bar{\times}_m \mathbf{G}_m = [\![\mathbf{C}; \mathbf{I}, \ldots, \mathbf{G}_m, \ldots, \mathbf{I}]\!]$.

Further, we say a tensor $\mathbf{X} \in \mathbb{R}^{p_1 \times \cdots \times p_M}$ follows the tensor normal distribution $TN(\boldsymbol{\mu}, \boldsymbol{\Sigma}_1, \ldots, \boldsymbol{\Sigma}_M)$ if it can be written as

$$\mathbf{X} = \boldsymbol{\mu} + [\![\mathbf{Z}; \boldsymbol{\Sigma}_1^{1/2}, \ldots, \boldsymbol{\Sigma}_M^{1/2}]\!],$$

where $\mathbf{Z} \in \mathbb{R}^{p_1 \times \cdots \times p_M}$ has elements all independently standard normal, $\boldsymbol{\mu} \in \mathbb{R}^{p_1 \times \cdots \times p_M}$ is the mean tensor, and $\boldsymbol{\Sigma}_m \in \mathbb{R}^{p_m \times p_m}$ are covariance matrices. See Figure 3 for an illustration.

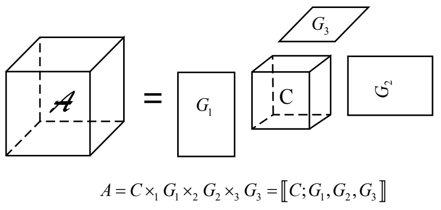

G_3
$\mathcal{A}$
=
G_1
C
G_2
$A = C \times_1 G_1 \times_2 G_2 \times_3 G_3 = [\![C; G_1, G_2, G_3]\!]$

Figure 2: Tucker decomposition of tensor **A**.

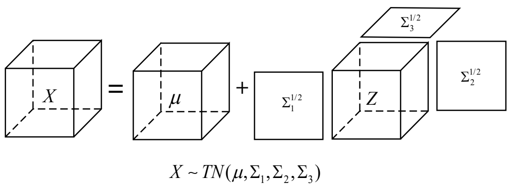


Figure 3: Tensor normal distribution.

Now we discuss the tensor discriminant analysis (TDA) model. Consider the M-way tensor predictor $\mathbf{X} \in \mathbb{R}^{p_1 \times \cdots \times p_M}$ where $M \geq 2$ and class label $Y \in \{1, \ldots, K\}$. The TDA model assumes that

$$\mathbf{X} \mid (Y = k) \sim TN(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_1, \ldots, \boldsymbol{\Sigma}_M), \quad \Pr(Y = k) = \pi_k \tag{10}$$

where $\boldsymbol{\mu}_k \in \mathbb{R}^{p_1 \times \cdots p_M}$, $\boldsymbol{\Sigma}_m \in \mathbb{R}^{p_m \times p_m}$ is the within-class mean, $\boldsymbol{\Sigma}_m > 0$ is the common within-class covariance matrix along the m-th mode of the tensor, and $0 < \pi_k < 1$ is the prior probability for Class k. Compared to the LDA model, TDA utilizes the tensor normal distribution to model $\mathbf{X}$ within each class. By taking advantage of the tensor structure, TDA drastically reduces the number of unknown parameters (Pan et al., 2019). It can be seen that the TDA model requires $O(\sum_{m=1}^M p_m^2)$ parameters to model the dependence among $\mathbf{X}$. However, if we ignore the tensor structure and assume the LDA model on the vectorized version of $\mathbf{X}$, the covariance matrix has $O(\prod_{m=1}^M p_m^2)$ parameters.

Under the TDA model, the Bayes' rule is

$$\widehat{Y} = \arg \max_{k=1,\ldots K} \{a_k + \langle \mathbf{B}_k, \mathbf{X} \rangle\} \tag{11}$$

where $\mathbf{B}_k = [\![\boldsymbol{\mu}_k - \boldsymbol{\mu}_1; \boldsymbol{\Sigma}_1^{-1}, \ldots, \boldsymbol{\Sigma}_M^{-1}]\!]$, and $a_k = \log(\pi_k/\pi_1) - \langle \mathbf{B}_k, \frac{1}{2}(\boldsymbol{\mu}_k + \boldsymbol{\mu}_1) \rangle$ is a scalar that does not involve $\mathbf{X}$. It can be seen that the Bayes rule is again a linear function in $\mathbf{X}$, with the linear coefficients $\mathbf{B}_k$. In high dimensions, we again impose the sparsity assumption by assuming that many elements in $\mathbf{B}_k$ are zeros.

Similar to the vector case, when additional covariates are provided, the TDA model can be combined with covariate adjustment. Pan et al. (2019) proposed the CATCH model for this purpose.

In addition to $(Y, \mathbf{X})$, we are given the covariates $\mathbf{U} \in \mathbb{R}^q$. The CATCH model assumes that

$$\mathbf{U} \mid (Y = k) \sim N(\boldsymbol{\phi}_k, \boldsymbol{\Psi}), \tag{12}$$

$$\mathbf{X} \mid (\mathbf{U} = \mathbf{u}, Y = k) \sim TN(\boldsymbol{\mu}_k + \boldsymbol{\alpha} \bar{\times}_{(M+1)} \mathbf{u}, \boldsymbol{\Sigma}_1, \ldots, \boldsymbol{\Sigma}_M). \tag{13}$$

where $\boldsymbol{\phi}_k \in \mathbb{R}^q$ is the within-class mean of $\mathbf{U}$, $\boldsymbol{\Psi} \in \mathbb{R}^{q \times q}$ is the within-class covariance of $\mathbf{U}$, and $\boldsymbol{\alpha} \in \mathbb{R}^{p_1 \times \cdots \times p_M \times q}$ characterizes the dependence of $\mathbf{X}$ on $\mathbf{U}$. The parameters in the CATCH model can be interpreted in the same way as the CA-LDA model in Section 2.2.3.

The Bayes' rule under the CATCH model is

$$\widehat{Y} = \arg\max_{k=1,\ldots K} \left\{ a_k + \boldsymbol{\gamma}_k^T \mathbf{U} + \langle \mathbf{B}_k, \mathbf{X} - \boldsymbol{\alpha} \bar{\times}_{(M+1)} \mathbf{U} \rangle \right\}, \tag{14}$$

where $\boldsymbol{\gamma}_k = \boldsymbol{\Psi}^{-1}(\boldsymbol{\phi}_k - \boldsymbol{\phi}_1)$, and $a_k = \log(\pi_k/\pi_1) - \frac{1}{2}\boldsymbol{\gamma}_k^T(\boldsymbol{\phi}_k + \boldsymbol{\phi}_1) - \langle \mathbf{B}_k, \frac{1}{2}(\boldsymbol{\mu}_k + \boldsymbol{\mu}_1) \rangle$ is a scalar that does not involve $\mathbf{X}$ or $\mathbf{U}$. Similar to the TDA model, we assume that $\mathbf{B}_k$ is sparse in high dimensions, but impose no further sparsity assumptions on other parameters.

Methods

In this section, we formally introduce the six methods implemented by the package: DSDA, ROAD, SOS, SeSDA, MSDA and CATCH. Throughout the rest of this paper, we denote $\widehat{\boldsymbol{\Sigma}}$ as the pooled sample covariance, $\widehat{\boldsymbol{\mu}}_k$ as the within-class sample mean, n as the sample size, and n_k as the sample size in class k. All the methods involve a tuning parameter $\lambda > 0$ that controls the amount of sparsity. Hence, when we refer to an estimate $\widehat{\boldsymbol{\beta}}$, it should be understood as $\widehat{\boldsymbol{\beta}}(\lambda)$, although we suppress λ in most estimates for presentation convenience. We will discuss the tuning parameter in detail in Section 2.3.8.

Direct sparse discriminant analysis (DSDA)

The direct sparse discriminant analysis (DSDA) is proposed for binary classification under the LDA model in (2). Recall that our main interest is in estimating the coefficients $\boldsymbol{\beta}_k$ in the Bayes rule (3). Because DSDA assumes that there are only two classes, it suffices to estimate $\boldsymbol{\beta} = \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)$. In high dimensions, we assume that $\boldsymbol{\beta}$ is sparse. Let $y_i = -\frac{n_1}{n}$ if $Y_i = 1$ and $y_i = \frac{n}{n_2}$ if $Y_i = 2$. DSDA first solves the penalized least squares problem

$$(\widehat{\boldsymbol{\beta}}^{\mathrm{DSDA}}, \widehat{\beta}_0^{\mathrm{DSDA}}) = \arg\min_{\boldsymbol{\beta} \in \mathbb{R}^p, \beta_0 \in \mathbb{R}} \left\{ n^{-1} \sum_{i=1}^{n} (y_i - \beta_0 - \mathbf{X}_i^T \boldsymbol{\beta})^2 + \lambda \sum_{j=1}^{p} |\beta_j| \right\}, \tag{15}$$

where $\lambda > 0$ is the tuning parameter, $\sum_{j=1}^{p} |\beta_j|$ is the LASSO penalty (Tibshirani, 1996), and $\widehat{\boldsymbol{\beta}}^{\mathrm{DSDA}}$ is our estimate for $\boldsymbol{\beta}$. Because of the LASSO penalty, $\widehat{\boldsymbol{\beta}}^{\mathrm{DSDA}}$ is typically sparse. To estimate the Bayes rule, we further estimate the LDA model on the reduced data $\{Y_i, \mathbf{X}_i^T \widehat{\boldsymbol{\beta}}^{\mathrm{DSDA}}\}_{i=1}^n$.

Numerical and theoretical studies show that DSDA consistently estimate the Bayes rule under mild conditions. Also, DSDA can be computed very efficiently, as (15) is a heavily-studied ℓ_1 penalized least squares problem. Our implementation utilizes `glmnet` to solve (15).

Regularized optimal affine discriminant (ROAD)

Regularized optimal affine discriminant (ROAD, Fan et al. (2012)) is another binary penalized discriminant analysis method for high-dimensional data. ROAD estimates $\boldsymbol{\beta}$ by

$$\widehat{\boldsymbol{\beta}}^{\mathrm{ROAD}} = \arg\min \boldsymbol{\beta}^T \widehat{\boldsymbol{\Sigma}} \boldsymbol{\beta} \tag{16}$$

$$\|\boldsymbol{\beta}\|_1 \le c, \boldsymbol{\beta}^T(\widehat{\boldsymbol{\mu}}_2 - \widehat{\boldsymbol{\mu}}_1)/2 = 1. \tag{17}$$

We remark that Wu et al. (2009) independently proposed the ℓ_1-Fisher's discriminant analysis method that closely resembles ROAD, but the developments of ROAD and the ℓ_1-Fisher's discriminant analysis have different emphasis. ROAD clarifies several theoretical aspects of high-dimensional classification, while ℓ_1-Fisher's discriminant analysis is developed for simultaneous testing for gene pathways. For simplicity, we focus on ROAD in what follows.

In its optimization, the constraint of ℓ_1-norm can be recast as a ℓ_1-penalty with parameter λ. ROAD rewrites (16) as

$$\widehat{\boldsymbol{\beta}}^{\text{ROAD}} = \arg \min_{\boldsymbol{\beta}^T(\widehat{\boldsymbol{\mu}}_2-\widehat{\boldsymbol{\mu}}_1)/2=1} \boldsymbol{\beta}^T\widehat{\boldsymbol{\Sigma}}\boldsymbol{\beta} + \lambda\|\boldsymbol{\beta}\|_1 \tag{18}$$

The authors of ROAD proposed to solve (18) by replacing the nonconvex constraint with a quadratic penalty. However, we adopt a different approach to solve (18). It is showed in Mai and Zou (2013) that the solution paths of DSDA and ROAD are equivalent. In other words, for any $\lambda > 0$, there exists $\tilde{\lambda} > 0$ such that $\widehat{\boldsymbol{\beta}}^{\text{DSDA}}(\lambda) \propto \widehat{\boldsymbol{\beta}}^{\text{ROAD}}(\tilde{\lambda})$. Because DSDA produces a solution path much faster than the original proposal of ROAD, we solve ROAD by first finding the solution path of DSDA for a range of λ, and then find each corresponding $\tilde{\lambda}$ to recover the solution path of ROAD.

Sparse optimal scoring (SOS) in binary problems

We also implement the successful discriminant analysis method, sparse optimal scoring (SOS, Clemmensen et al. (2011)). We focus on binary problems, where we are able to greatly improve the computation speed. For multiclass problems, SOS can be solved by the R package **sparseLDA**.

In binary problems, SOS creates a dummy variable $\mathbf{Y}^{dm} \in \mathbb{R}^{n\times 2}$ as a surrogate for the categorical response Y, where $Y^{dm}_{ik} = 1\{Y_i = k\}$. Then SOS estimates coefficient by solving

$$\widehat{\boldsymbol{\beta}}^{\text{SOS}} = \arg \min_{\boldsymbol{\theta}\in\mathbb{R}^2,\boldsymbol{\beta}\in\mathbb{R}^p} \{\|\mathbf{Y}^{dm}\boldsymbol{\theta} - \widetilde{\mathbf{X}}\boldsymbol{\beta}\|^2 + \lambda\|\boldsymbol{\beta}\|_1\},$$
$$\text{s.t } \tfrac{1}{n}\boldsymbol{\theta}^T\mathbf{Y}^{dm^T}\mathbf{Y}^{dm}\boldsymbol{\theta} = 1, \boldsymbol{\theta}^T\mathbf{Y}^{dm^T}\mathbf{Y}^{dm}1 = 0, \tag{19}$$

where $\widetilde{\mathbf{X}}$ is the centered $\mathbf{X}$, and $\boldsymbol{\theta} \in \mathbb{R}^2$ is the score for the two classes. SOS is a popular penalized discriminant analysis method because of its impressive empirical performance. It can be solved by iteratively minimizing the objective function in (19) over $\boldsymbol{\theta}$ and $\boldsymbol{\beta}$.

However, we take another approach to solve SOS with lower computation cost. Mai and Zou (2013) showed that $\widehat{\boldsymbol{\beta}}^{\text{SOS}}$ is closely related to the DSDA estimator defined in (15). Let $\hat{\pi}_y = \frac{n_y}{n}$. We have that

$$\widehat{\boldsymbol{\beta}}^{\text{SOS}}(\lambda) = \sqrt{\hat{\pi}_1\hat{\pi}_2}\widehat{\boldsymbol{\beta}}^{\text{DSDA}}(\frac{\lambda}{\sqrt{\hat{\pi}_1\hat{\pi}_2}}). \tag{20}$$

Therefore, to solve for $\widehat{\boldsymbol{\beta}}^{\text{SOS}}(\lambda)$, we first find $\widehat{\boldsymbol{\beta}}^{\text{DSDA}}(\frac{\lambda}{\sqrt{\hat{\pi}_1\hat{\pi}_2}})$ with DSDA, and rescale it to obtain the SOS solution. This approach avoids iteration between $\boldsymbol{\theta}$ and $\boldsymbol{\beta}$, and is often faster than the original algorithm for SOS.

Semiparametric sparse discriminant analysis (SeSDA)

SeSDA (Mai and Zou, 2015) fits the SeLDA model in (7) for binary problems. It is expected to have better performance than DSDA when data are heavily non-normal. SeSDA has two steps. First, we find an estimate $\widehat{h}$ for the unknown function h. Second, we apply DSDA on the pseudo data $(\widehat{h}(\mathbf{X}), Y)$. In what follows, we focus on the estimation of h.

Two estimators have been proposed for h based on (8), the naive estimator and the pooled estimator. Without loss of generality, we assume that Class 1 has more observations than Class 2. Denote $\tilde{F}_{1j}$ as the empirical CDF of X_j within Class 1. To avoid infinity values at tails, we further Winsorize $\tilde{F}_{1j}$ to $\hat{F}_{1j}$, where

$$\hat{F}_{1j}(x) = \begin{cases} 1-1/n_1^2 & \text{if } \tilde{F}_{1j}(x) > 1-1/n_1^2 \\ \tilde{F}_{1j}(x) & \text{if } 1/n_1^2 \le \tilde{F}_{1j}(x) \le 1-1/n_1^2 \\ 1/n_1^2 & \text{if } \tilde{F}_{1j}(x) < 1/n_1^2. \end{cases}$$

The naive estimator is shown to consistently estimate h, but in practice it is vulnerable to loss of efficiency, as it only utilizes one class of data. Therefore, the pooled estimator is proposed as a more efficient estimator.

Similar to $\hat{F}_{1j}$, we denote $\hat{F}_{2j}$ as the empirical CDF of X_j within Class 2 Winsorized at $(1/n_2^2, 1-1/n_2^2)$. We first find an estimate for $\boldsymbol{\mu}_{2j}$ as $\widehat{\boldsymbol{\mu}}^{(\text{pool})}_{2j} = \hat{\pi}_1\hat{\mu}^{(1)}_{2j} + \hat{\pi}_2\hat{\mu}^{(2)}_{2j}$, where $\hat{\mu}^{(1)}_{2j} = \frac{1}{n_2}\sum_{Y_i=2}\Phi^{-1}\circ\hat{F}_{1j}(X_{ij}), \hat{\mu}^{(2)}_{2j} = -\frac{1}{n_1}\sum_{Y_i=1}\Phi^{-1}\circ\hat{F}_{2j}(X_{ij})$. Then the pooled estimator for h_j is

$$\widehat{h}^{(\text{pool})}_j = \hat{\pi}_1\hat{h}^{(1)}_j + \hat{\pi}_2\hat{h}^{(2)}_j, \tag{21}$$

Algorithm 1 Algorithm for MSDA

```
1. Compute $\widehat{\Sigma}$ and $\widehat{\delta}^k = (\widehat{\mu}_k - \widehat{\mu}_1), k = 1, 2, \cdots, K$.
2. Initialize $\widehat{\beta}_k^{(0)}$ and compute $\widetilde{\beta}_k^{(0)}$ by $\widetilde{\beta}_{k,j} = \frac{\hat{\delta}_j^k - \sum_{l \neq j} \hat{\sigma}_{lj} \hat{\beta}_{kl}}{\hat{\sigma}_{jj}}$.
3. For steps $w = 1, 2, \ldots$, do the following until convergence:
   for each element $j = 1, \ldots, p$,
      (a) Compute
          $\widehat{\beta}_{\cdot j}^{(w)} = \widetilde{\beta}_{\cdot j}^{(w-1)} (1 - \frac{\lambda}{\| \widetilde{\beta}_{\cdot j}^{(w-1)} \|})_+;$   (24)
      (b) Update
          $\tilde{\beta}_{kj} = \frac{\hat{\delta}_j^k - \sum_{l \neq j} \hat{\sigma}_{lj} \hat{\beta}_{kl}^{(w)}}{\hat{\sigma}_{jj}}.$   (25)
4. At convergence, output $\beta_k$.
```

where $\hat{h}_j^{(1)} = \Phi^{-1} \circ \widehat{F}_{1j}$ and $\hat{h}_j^{(2)} = \Phi^{-1} \circ \widehat{F}_{2j} + \hat{\mu}_{2j}^{(\text{pool})}$. The pooled estimator is usually more accurate than the naive estimator because it utilizes both classes to form an estimate for h_j.

Multiclass sparse discriminant analysis (MSDA)

Up to now, we have focused on binary classifiers. In this section, we discuss a multiclass classifier under the LDA model (2). Assume that $K \geq 2$. By the Bayes rule (3), we need to estimate the coefficients $\beta_k = \Sigma^{-1}(\mu_k - \mu_1), k = 2, \ldots, K$. There is no need to estimate β_1, as it is zero by definition. As in the binary problems, we continue to assume that the classifier is sparse in high dimensions, in the sense that only a few predictors are relevant to classification. However, this sparsity assumption has slightly different implication in multiclass problems. Note that, for any X_j, if any one of $\beta_{2j}, \ldots, \beta_{Kj}$ is nonzero, X_j is important for classification, as it helps with distinguishing between at least one pair of classes. Therefore, in order for an X_j to be unimportant, we have to have $\beta_{2j} = \ldots = \beta_{Kj} = 0$. In other words, the coefficients $\beta_2, \ldots, \beta_K$ has a group sparsity structure.

The multi-class sparse discriminant analysis (MSDA) has been proposed for fitting a sparse classifier under the context of interest. It takes note of the fact that, on the population level, we have

$$(\beta_2, \cdots, \beta_K) = \arg \min_{\beta_2, \cdots, \beta_K} \sum_{k=2}^{K} \{ \frac{1}{2} \beta_k^T \Sigma \beta_k - (\mu_k - \mu_1)^T \beta_k \}. \tag{22}$$

Therefore, in high dimensions, MSDA replaces the parameters with the sample estimates and impose the group sparsity structure through group lasso (Yuan and Lin, 2006). More specifically, MSDA estimates β by

$$(\widehat{\beta}_2, \cdots, \widehat{\beta}_K) = \arg \min_{\beta_2, \cdots, \beta_K} \sum_{k=2}^{K} \{ \frac{1}{2} \beta_k^T \widehat{\Sigma} \beta_k - (\widehat{\mu}_k - \widehat{\mu}_1)^T \beta_k \} + \lambda \sum_{j=1}^{P} \| \beta_{\cdot j} \| . \tag{23}$$

The problem in (23) can be solved by a blockwise coordinate descent algorithm (Mai et al., 2015) summarized in Algorithm 1. We refer to Algorithm 1 as the original MSDA algorithm. The R package **msda** implements such an algorithm. However, the original MSDA algorithm can be demanding on storage for high-dimensional data, because it requires the input of $\widehat{\Sigma} \in \mathbb{R}^{p \times p}$. When p is very large, the original MSDA algorithm can be practically inapplicable. Moreover, because of the sparse nature of β, many elements in $\widehat{\Sigma}$ are never used, and the calculation of them leads to unnecessary computation burden.

Therefore, in our implementation we modify the original MSDA algorithm for lower storage and computation cost for high-dimensional data. Note that $\widehat{\Sigma}$ is only used in updating rule (25). We take advantage of two properties of this updating rule (25). First, given the natural element-wise property of coordinate descent algorithm, only the j-th column of covariance matrix $\widehat{\Sigma}_{\cdot j}$ is needed in each iteration. The full covariance matrix is never used during the computation process. Therefore, it is not necessary to store the huge covariance matrix. Secondly, a large number elements of $\widehat{\boldsymbol{\beta}}$ are exactly 0. Hence among the column $\widehat{\boldsymbol{\Sigma}}_{\cdot j}$, we only need to compute the rows corresponding to the nonzero coefficients. These facts motivate us to develop the modified MSDA algorithm. The modified MSDA algorithm is largely identical to the original algorithm, but with two important distinctions. On one hand, in Step 1 we only require the input of $\widehat{\delta}^k$ but not $\widehat{\boldsymbol{\Sigma}}$. On the other hand, Step 3(b) in (25) is replaced with

$$\tilde{\beta}_{kj} = \frac{(n-K)\hat{\beta}_j^k - \sum_{l\neq j}\hat{\beta}_{kl}^{(m)}(\sum_{k=1}^K[\sum_{i\in\mathbf{T}_k}(X_{il}-\mu_{kl})(X_{ij}-\mu_{kj})])}{\sum_{k=1}^K[\sum_{i\in\mathbf{T}_k}(X_{ij}-\mu_{kj})^2]}, \tag{26}$$

where $\mathbf{T}_k = \{i : y_i = k\}$. By doing so, we avoid the storage and the computation of the full matrix of $\widehat{\boldsymbol{\Sigma}}$. In computing (26), we further use three tricks to speed up the computation. Firstly, we calculate and store all diagonal elements in the covariance matrix as they will be called multiple times. Secondly, we keep the indexes of nonzero elements in $\mathbf{T}_k$ and update it every time we observe a new nonzero element. Hence we do not need to check all elements to locate the nonzero ones in each iteration. Thirdly, we update equation (26) by only computing elements corresponding to the nonzero indexes in $\mathbf{T}_k$. With these three tricks, the modified algorithm reduces the space complexity from $O(p^2)$ to $O(p)$, and is also faster than the original algorithm for large p.

Covariate-adjusted tensor classification in high dimensions (CATCH)

When $\mathbf{X}$ is a tensor instead of a vector, we need to fit the TDA model or the CATCH model (in presence of covariates) for better efficiency and accuracy. Pan et al. (2019) proposed the CATCH method to fit both models, but in this section we focus on the CATCH method on the TDA model, where there is no covariate. The inclusion of covariates will be discussed in Section 2.3.7.

Recall that, under the TDA model, we aim to estimate the parameters $\mathbf{B}_k = [\![\boldsymbol{\mu}_k - \boldsymbol{\mu}_1; \boldsymbol{\Sigma}_1^{-1}, \ldots, \boldsymbol{\Sigma}_M^{-1}]\!]$. We first rewrite $\mathbf{B}_k$ as solutions to estimating equations:

$$(\mathbf{B}_2, \ldots, \mathbf{B}_K) = \arg\min_{\mathbf{B}_2,\ldots,\mathbf{B}_K} \sum_{k=2}^K \left(\langle \mathbf{B}_k, [\![\mathbf{B}_k; \boldsymbol{\Sigma}_1, \ldots, \boldsymbol{\Sigma}_M]\!]\rangle - 2\langle \mathbf{B}_k, \boldsymbol{\mu}_k - \boldsymbol{\mu}_1\rangle\right),$$

where for two M-way tensors $\mathbf{A}, \mathbf{C}$, $\langle \mathbf{A}, \mathbf{C}\rangle = \sum_{j_1\cdots j_M} a_{j_1\cdots j_M} c_{j_1\cdots j_M}$ is the inner product of two tensors. To estimate $\mathbf{B}_k$, we find the within-class sample mean $\widehat{\boldsymbol{\mu}}_k$ as the estimate for $\boldsymbol{\mu}_k$, and moment-based unbiased estimators $\widehat{\boldsymbol{\Sigma}}_m$ for $\boldsymbol{\Sigma}_m$; see the formulas in Appendix C. We further add the group LASSO penalty for variable selection. Therefore, CATCH solves the following problem:

$$\min_{\mathbf{B}_2,\ldots,\mathbf{B}_K} \left[\sum_{k=2}^K \left(\langle \mathbf{B}_k, [\![\mathbf{B}_k; \widehat{\boldsymbol{\Sigma}}_1, \ldots, \widehat{\boldsymbol{\Sigma}}_M]\!]\rangle - 2\langle \mathbf{B}_k, \widehat{\boldsymbol{\mu}}_k - \widehat{\boldsymbol{\mu}}_1\rangle\right) + \lambda \sum_{j_1\ldots j_M} \sqrt{\sum_{k=2}^K b_{k,j_1\cdots j_M}^2}\right]. \tag{27}$$

CATCH can be solved by a coordinate descent algorithm with an explicit updating formula in each iteration.

Covariates adjustment

When we have additional covariates $\mathbf{U}$, the CA-LDA model or the CATCH model should be fitted. Whether $\mathbf{X}$ is a vector or a tensor, a key step for the covariate adjustment is the estimation of $\boldsymbol{\alpha}$, the dependence of $\mathbf{X}$ on $\mathbf{U}$. We use the maximum likelihood estimator (MLE). Denote $\overline{\mathbf{U}}_k$ as the sample mean of $\mathbf{U}$ within class k and $\overline{\mathbf{X}}_k$ as the sample mean of $\mathbf{X}$ within class k. Define group-wise centered data $\widetilde{\mathbf{X}}_i = \mathbf{X}_i - \overline{\mathbf{X}}_{Y_i}$, $\widetilde{\mathbf{U}}_i = \mathbf{U}_i - \overline{\mathbf{U}}_{Y_i}$.

For vector-variate $\mathbf{X}_i \in \mathbb{R}^p$, we adjust for covariate $\mathbf{U}$ by $\mathbf{X}_i - \widehat{\boldsymbol{\alpha}}\mathbf{U}_i$, where $\widehat{\boldsymbol{\alpha}} \in \mathbb{R}^{q\times p}$ is the MLE,

$$\widehat{\boldsymbol{\alpha}} = (\widetilde{\mathbf{U}}^T\widetilde{\mathbf{U}})^{-1}\widetilde{\mathbf{U}}^T\widetilde{\mathbf{X}}. \tag{28}$$

For tensor-variate $\mathbf{X}_i \in \mathbb{R}^{p_1\times\cdots\times p_M}$, we let $\boldsymbol{\alpha}_{j_1\cdots j_M} \in \mathbb{R}^q$ be the regression coefficient of univariate $X_{i,j_1\cdots j_M}$ on multivariate $\mathbf{U}_i \in \mathbb{R}^q$. Then the MLE for $\boldsymbol{\alpha}_{j_1\cdots j_M}$ is $\widehat{\boldsymbol{\alpha}}_{j_1\cdots j_M} = (\widetilde{\mathbf{U}}^T\widetilde{\mathbf{U}})^{-1}\widetilde{\mathbf{U}}^T\widetilde{X}_{j_1\cdots j_M}$, which can be expressed more explicitly as,

$$\widehat{\boldsymbol{\alpha}}_{j_1\cdots j_M} = \left\{\sum_{k=1}^K\sum_{Y_i=k}(\mathbf{U}_i - \overline{\mathbf{U}}_k)(\mathbf{U}_i - \overline{\mathbf{U}}_k)^T\right\}^{-1}\left\{\sum_{k=1}^K\sum_{Y_i=k}(\mathbf{U}_i - \overline{\mathbf{U}}_k)(X_{i,j_1\cdots j_M} - \overline{X}_{k,j_1\cdots j_M})\right\}. \tag{29}$$

	Parameters	λ	`dfmax`	`model` option	Covariate	Cross validation
DSDA	Binary vector	✓			✓	✓
ROAD	Binary vector	✓				
SOS	Binary vector	✓				
SeSDA	Binary vector	✓				✓
MSDA	Multi-class vector	✓	✓	✓	✓	✓
CATCH	Multi-class tensor	✓	✓		✓	✓

Table 2: Method description and major parameters. Penalty parameter λ controls the size of ℓ_1-penalty. Parameter `dfmax` limits the maximum number of non-zero variables. Parameter `model` specifies the version of implementation for MSDA.

Afterwards, the ensemble of all $\widehat{\boldsymbol{\alpha}}_{j_1\cdots j_M}$, $\widehat{\boldsymbol{\alpha}}$, is our estimator for $\boldsymbol{\alpha}$. The covariate-adjusted predictor is then obtained as $\mathbf{X}_i - \widehat{\boldsymbol{\alpha}}\bar{\times}_{M+1}\mathbf{U}_i$.

Selection of the tuning parameter

We recommend selecting the tuning parameter in all methods by cross validation, which is implemented in our package as supportive functions for most of the methods. In cross validation, a sequence of potential tuning parameters is supplied. For each candidate tuning parameter λ, the dataset is random split into L folds. Then we fit L classifiers, each of which is fitted on $L-1$ folds of the data and validated on the remaining one fold. The average validation error rate of the L classifiers is used as a measurement of the performance of the corresponding λ. The λ with the smallest average validation error is used in our final model fitting.

If desired, our package can automatically generate a sequence of tuning parameters for all the methods. They will first compute the smallest λ that shrinks all coefficients to zero; this value is taken as the upper bound of tuning range. Then the upper bound is multiplied by a small number to generate the lower bound. Finally, a sequence of tuning parameters is uniformly generated between the lower and the upper bound.

Using the R package

The R package **TULIP** provides user-friendly functions to fit discriminant analysis model and perform predictions on vector and tensor data. The package can be downloaded through `https://cran.r-project.org/web/packages/TULIP` or install in R through `install.packages('TULIP')`. In installing package, the pre-required packages **MASS**(Venables and Ripley, 2002) for LDA model fitting, packages **Matrix** (Bates and Maechler, 2016) and **tensr** (Gerard and Hoff, 2016) for matrix and tensor operations, and the package **glmnet**(Friedman et al., 2010) for LASSO are also automatically installed. Users do not need to install them separately. To guarantee higher computation efficiency of the package, core algorithms of MSDA and CATCH are implemented in Fortran, which have already been compiled and can also be used directly.

Among all the six methods, there is always a tuning parameter λ to control the size of sparsity. On the implementation aspect, MSDA and CATCH also have parameter `dfmax` to limit the number of selected variables and will only return the solutions with number of non-zero elements less than `dfmax`. Furthermore, MSDA has a `model` option to specify version of implementation between multi.original and multi.modified. The methods are summarized in Table 2.

The functions in the package consists of two parts. One part contains core functions which generate solution paths of all the methods, including functions `dsda`, `road`, `sos`, `SeSDA`, `msda` and `catch`. Since binary classification can be regarded as a special case of multi-class problems, we also embedded DSDA into `msda` function. See Section 2.4.1 for details. The other part includes supportive functions to perform covariate adjustment, prediction, cross validation and handle some special cases.

To illustrate how to use the functions, we first simulate a binary vector data set named `dat.vec` with dimension $p = 500$ and sample size $n_k = 75$. In the data set, we have $\mathbf{X}_i \mid (Y_i = k) \sim N(\boldsymbol{\mu}_k, \boldsymbol{\Sigma})$, where $\boldsymbol{\mu}_1 = 0$, $\boldsymbol{\mu}_2 = \boldsymbol{\Sigma}\boldsymbol{\beta}$, $\sigma_{ij} = 0.3$ if $i \neq j$ and $\sigma_{ii} = 1$, $\boldsymbol{\beta}_j = 0.5$ for $1 \leq j \leq 10$ and $\boldsymbol{\beta}_j = 0$ otherwise. We further generate a testing data set with sample size 1000 from the same distribution. Variables in `dat.vec` is summarized in Table 3. Data set `dat.vec` can be simulated by code

```
dat.vec<-sim.bi.vector(1000)
```

Variable	Type	Dimension
x	matrix	150×500
y	vector	150
testx	matrix	1000×500
testy	vector	1000

Table 3: Data set dat.vec. Variables type and dimension are listed.

Moreover, we include two real data sets, GDS1615 and colorimetric sensor array data set, in the package to demonstrate usage of the functions. Data set GDS1615 (Burczynski et al., 2006) is a vector data set where observations belong to three classes. The original data set contains 127 observations and 22283 variables. Package **msda** preprocessed the data by computing F-test statistics of each variable (Mai et al., 2015), whose definition is in appendix. Hence only 127 variables are kept in the data set. Colorimetric sesor array data (CSA) was used to show the performance of discriminant analysis method (Zhong and Suslick, 2015). It records information of chemical dyes after exposed to volatile chemical toxicants to identify their classes. It contains 147 observations in 21 classes. For each observation, the predictor is a 36×3 matrix. We include two conditions in our dataset, but focus on the Immediately Dangerous to Life or Health (IDLH) condition.

Core functions

Function `dsda`

The following code shows an example of utilizing DSDA. Given the data set `dat.vec`, we fit DSDA on $\{\mathbf{X}, Y\}$ by specifying the tuning range of parameter λ to be a sequence between $[0.005, 0.3]$. Hence the function will generate a solution path. Next, we apply `predict` function on the model and obtain the prediction for each λ and error rate. In the example, we report the minimum error rate and corresponding parameter value.

```
obj <- dsda(dat.vec$x, y=dat.vec$y, lambda=seq(0.005, 0.3, length.out=20))
pred <- predict(obj, dat.vec$testx)
err<- apply(pred, 2, function(x){mean(x!=dat.vec$testy)})
print(min(err))
[1] 0.111
print(obj$lambda[which.min(err)])
[1] 0.02052632
```

If one wishes, `dsda` can also be used in a more automatic way. On one hand, it can be called without supplying a sequence of value for tuning parameter. The function will automatically generate a sequence based on data. On the other hand, the prediction can be performed along with model fitting if testing data is supplied. The function `dsda` will produce the prediction error on the testing data corresponding to each tuning parameter. See the following example.

```
obj <- dsda(dat.vec$x, y=dat.vec$y,testx=dat.vec$testx)
err <- apply(obj$pred, 2, function(x){mean(x!=dat.vec$testy)})
print(min(err))
[1] 0.107
print(obj$lambda[which.min(err)])
[1] 0.03180946
```

Figure 4 shows a solution path of DSDA model. As parameter λ increases, more coefficients will be shrunken towards 0. In addition, DSDA can also integrate the covariate adjustment, model fitting and prediction. The usage is similar to the function `catch`, and we do not give a separate example here to avoid redundancy.

Function `SeSDA`

Function `SeSDA` fits a semiparametric sparse discriminant analysis model on the input vector data. The simulated data `dat.vec` follows normal distribution within each class. We take an exponential transformation on it to violate the normality assumption. The following example shows that `SeSDA` achieves error rates 11%. However, if we directly apply DSDA on the data set, the minimum error rate is as high as 15.8%. Therefore, the preprocessing of SeSDA can indeed help to improve performance under this scenario.

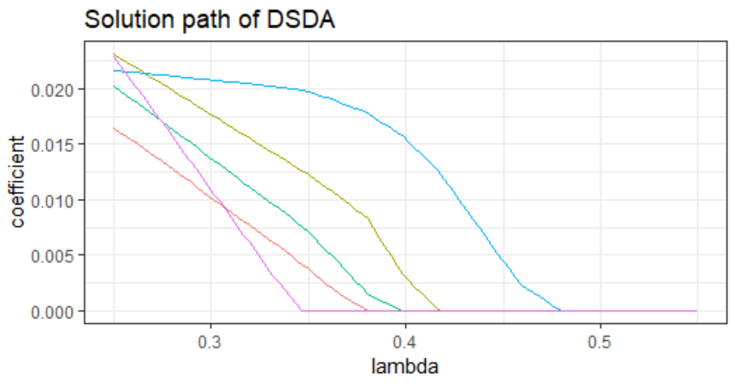


Figure 4: Solution path of five selected variables in a DSDA model. Five trends correspond to the parameter values of five elements given different parameter λ values.

```
x <- exp(dat.vec$x)
testx <- exp(dat.vec$testx)
obj.SeSDA <- SeSDA(x, y=dat.vec$y)
pred.SeSDA <- predict(obj.SeSDA, testx)
err <- apply(pred.SeSDA, 2, function(x){mean(x!=dat.vec$testy)})
min(err)
[1] 0.11
```

Further, Figure 5 shows how the distribution of the first variable changes after transformation. It is clear that both pooled and naïve transformatins result in approximately normal distribution.

Functions `ROAD` **and** `SOS`

Functions `ROAD` and `SOS` can generate equivalent solution paths as ROAD (Fan et al., 2012) and SOS (Clemmensen et al., 2011) methods on binary vector data, respectively. Both of the two models are fit by calling `dsda` function. Compared to the original package for SOS, **sparseLDA**, our implementation is usually faster, especially when a solution path or parameter tuning is needed. For example, to fit a solution path with 10 possible values of λs on a toy example with $p = 40$, our implementation reduces the computation time by half compared to **sparseLDA**. An example of fitting ROAD and SOS model is as follows. The `lambdas` passed into `ROAD` and `SOS` functions will be directly used by `dsda` function. The `lambdas` returned by the two functions are their corresponding parameters in ROAD and SOS model, respectively. Figure 6 shows the relationship between the λ's that generate the same solution.

```
obj.dsda <- dsda(dat.vec$x, y=dat.vec$y, lambda=seq(0.1, 0.5, length.out=20))
obj.road <- ROAD(dat.vec$x, y=dat.vec$y, lambda=seq(0.1, 0.5, length.out=20))
obj.sos <- SOS(dat.vec$x, y=dat.vec$y, lambda=seq(0.1, 0.5, length.out=20))
```

Function `msda`

The function `msda` provides an interface to fit MSDA. Similarly to `dsda`, without specification of possible values of λ, the function will automatically generate a sequence of λs. Function `msda` can also perform predictions when testing data is supplied and make adjustments on covariates when covariates exist. We apply `msda` on GDS1615 data set to as a demonstration. We report the minimum training error, its corresponding parameter value and the number of non-zero variables selected by the model.

```
data(GDS1615)
x <- GDS1615$x
y <- GDS1615$y
set.seed(123456)
teindex <- c(sample(which(y==1), sum(y==1)/3), sample (which(y==2),
   sum(y==2)/3), sample(which(y==3), sum(y==3)/3))
obj <- msda(x[-teindex, ], y=y[-teindex], testx=x[teindex, ])
err <- apply(obj$pred, 2, function(x){mean(x!=y[teindex])})
paste(min(err), obj$lambda[which.min(err)], obj$df[which.min(err)] )
[1] "0.04878049 1.446872 19"
```

If one wishes to visualize the discriminant effect, plots of projections on the discriminant coefficients is helpful. A principle component analysis is also optional to show the classification even more clearly. For illustration, we perform principle component analysis on $\mathbf{X}\boldsymbol{\beta}$ where $\boldsymbol{\beta} = \{\boldsymbol{\beta}_2, \boldsymbol{\beta}_3\}$ is the

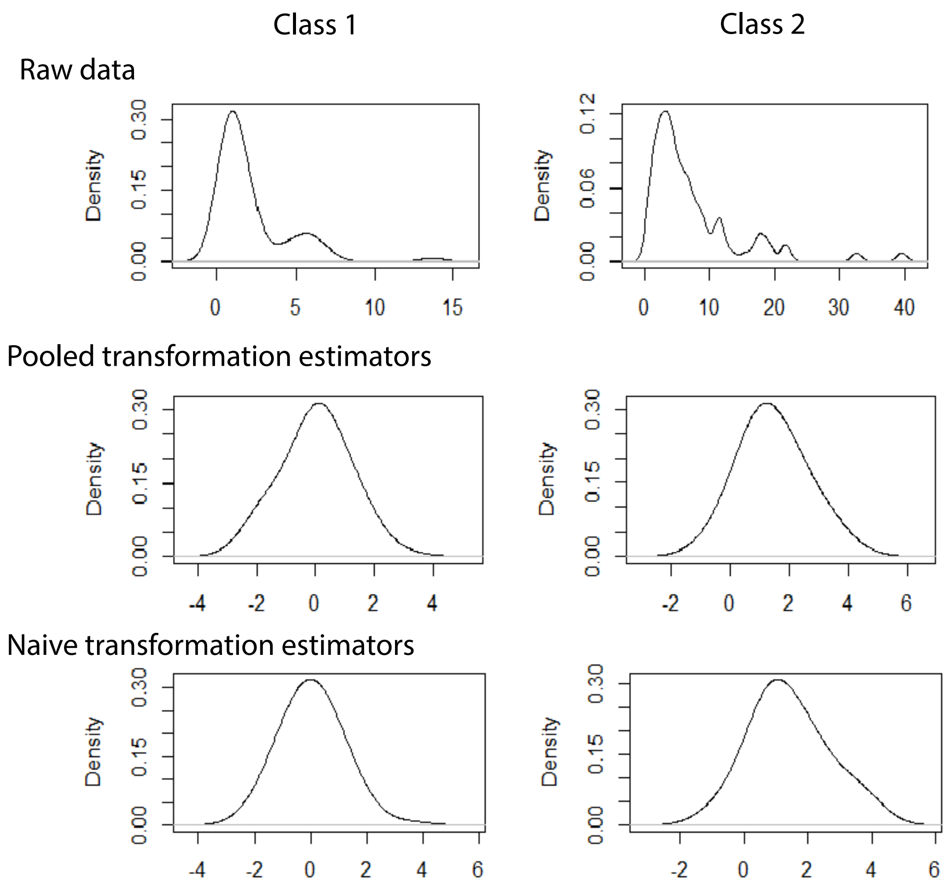


Figure 5: The distribution of the 1st variable in simulated data set among two classes before transformation and after transformation. The top row is before transformation. The second row is after pooled transformation. The bottom row is after naïve transformation.

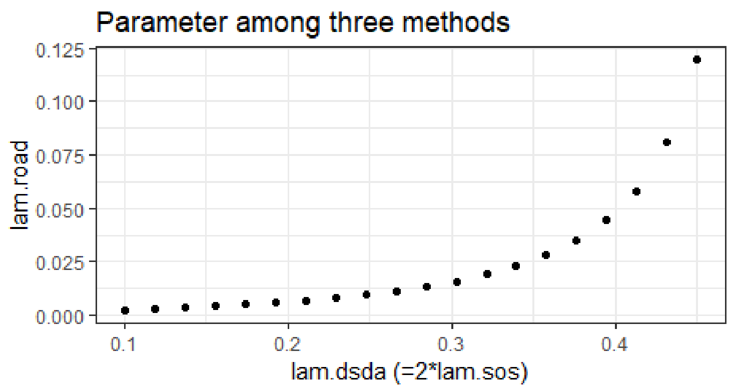


Figure 6: Parameters in ROAD vs. Parameters in DSDA. Notice that the parameters in DSDA are double those of SOS.

discriminant coefficient. The scatter plot on the two principle components is shown in Figure 7. It is clear to see that the three classes are separated well.

We also note that `msda` has an argument `model` that can be specified by users to use different algorithms in MSDA. The options for `model` include `binary`, `multi.original` and `multi.modified`.

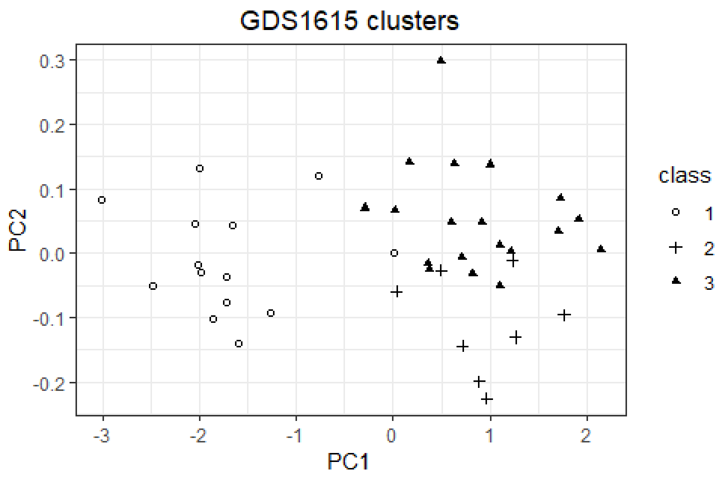


Figure 7: The GDS data projected onto the two principle components of $\mathbf{X}\boldsymbol{\beta}$. Three classes are separated.

Variable	Type	Dimension
x	list	150. Each element is a $10 \times 10 \times 10$ array.
y	vector	150
z	matrix	150×2
vec_x	matrix	1000×150
testx	list	1000. Each element is a $10 \times 10 \times 10$ array.
testy	vector	1000
testz	matrix	150×2
vec_testx	matrix	1000×1000

Table 4: Data set dat.ten. Variables type and dimension are listed.

The option `binary` can only be used in binary problems. If selected, MSDA is solved by DSDA, which gives the same solution with usually less time. However, using this option in multi-class problems will result in an error. The option `multi.original` indicates that MSDA is solved by the original algorithm, which requires the calculation of the full covariance matrix. When the dimension is low, `multi.original` is often efficient. The option `multi.modified`, on the other hand, solves MSDA with the modified algorithm, where only part of the covariance matrix is calculated in each iteration. This option allows MSDA to be applicable in much higher dimensions. Also, when `multi.modified` is selected, we suggest using relatively larger tuning parameters to account for the high dimensionality. If unspecified, `model` is set to be `binary` in binary problems. If the response variable is multi-class, the function will call multi.original implementation for $p \leq 2000$ and multi.modified implementation for $p > 2000$.

Function `catch`

To illustrate usage of function `catch`, we first simulate a data set named `dat.ten` with tensor predictors $\mathbf{X}_i \in \mathbb{R}^{10\times10\times10}$ and covariates $\mathbf{U}_i \in \mathbb{R}^2$. The data is simulated from model $\mathbf{X}_i \mid (Y_i = k) \sim TN(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_1, \boldsymbol{\Sigma}_2, \boldsymbol{\Sigma}_3)$ where $\boldsymbol{\mu}_1 = 0$, $\boldsymbol{\mu}_2 = [\![\boldsymbol{\beta}; \boldsymbol{\Sigma}_1, \boldsymbol{\Sigma}_2, \boldsymbol{\Sigma}_3]\!]$, $\boldsymbol{\Sigma}_j = \mathbf{I}$ for $j = 1,2,3$, $\boldsymbol{\beta}_{[1:2,1:2,1:2]} = 0.8$ and 0 otherwise. Let $\mathbf{U}_i \mid (Y_i = k) \sim N(\boldsymbol{\phi}_k, \boldsymbol{\psi})$ where $\boldsymbol{\phi}_1 = 0$, $\boldsymbol{\phi}_2 = (0.3, 0.3)$ and $\boldsymbol{\psi} = \mathbf{I}$. The connection between $\mathbf{X}$ and $\mathbf{U}$ is measured by $\boldsymbol{\alpha} \in \mathbb{R}^{10\times10\times10\times2}$ and $\boldsymbol{\alpha}_{[1:5,1:5,1:5,1]} = 1$ and 0 otherwise. Variables in `dat.ten` are summarized in Table 4.

Data set `dat.vec` can be simulated by code

```
dat.ten<-sim.tensor.cov(1000)
```

Function `catch` fits a CATCH model on the input tensor data. Covariates are optional for the function and the function will fit a TDA model when there is no covariate. Function `catch` has already integrated the adjustment step and model fitting step, hence it will automatically adjust for covariates when covariates exist. If one prefers to seperate the adjustment step, he/she can call `adjten` function to make adjustments and then supply the adjusted predictors into `catch`, which we will discuss in supportive functions.

Similar to the two functions above, function catch can generate a solution path on default or user specified potential values of the parameter. It will also perform prediction when testing data is specified. To make predictions on CATCH model, user can directly apply catch function or separating adjustment and model fitting step and then call the predict function. The following example shows how to fit the model and make prediction when covariates exist. As mentioned above, functions dsda and msda shares the same arguments name for covariates.

```
obj <- catch(dat.ten$x, dat.ten$z, dat.ten$y, dat.ten$testx, dat.ten$testz)
pred <- obj$pred
err <- apply(pred, 2, function(x){mean(x!=dat.ten$testy)})
min(err)
[1] 0.167
obj$lambda[which.min(err)]
[1] 0.4270712
```

An example of applying CATCH to fit model and perform prediction on CSA data is as follows. catch function takes list of multi-dimensional array as input. In the dataset, x is a list of length 148, where each element is a matrix of dimension 36 × 3; y is a vector whose value ranges between 1 and 21. We use default parameter sequence of length 100 and the prediction for each value of parameter is generated.

```
data(csa)
x <- csa$IDLH
y <- csa$y
teindex <- seq(1,147,7)
obj <- catch(x[-teindex, ], y=y[-teindex], testx=x[teindex, ], nlambda=10)
err <- apply(obj$pred, 2, function(x){mean(x!=y[teindex])})
print(err)
[1] 0.9523819 0.1904762 0.0952381 0.0000000 0.0952381 0.0000000 0.0000000
 0.0000000 0.0000000 0.0000000
```

Other functions

Two special cases

We provide two more functions for two common problems in practice, binary classification and matrix classification, respectively. First, the function catch_matrix fits CATCH model on matrix data (2-way tensor), which is a special case of catch. The usage of catch_matrix is exactly the same as that of catch, with the only exception that the predictor has to be a matrix instead of higher-order tensor.

Second, our package includes the function dsda.all that integrates cross validation, model fitting and prediction. It requires the input of the training set and testing set. Then the optimal tuning parameter is chosen by cross validation on the training set, and the corresponding testing error is reported. See the following example.

```
obj <- dsda.all(dat.vec$x, dat.vec$y, dat.vec$testx, dat.vec$testy, nfolds = 10)
print(obj$err)
[1] 0.116
```

Supportive functions

The package provides functions cv.dsda, cv.msda, cv.SeSDA and cv.catch to perform cross validation. For all of these functions, user can give a sequence of potential values to tune parameter. Otherwise, the function will first fit a model on the entire data set and then perform cross validation on the automatically generated λs from the entire data set. Similar as msda, user can specify which model to use in cv.msda or let the function determine by input data.

Users can also specify the number of folds by the argument nfolds. Another argument lambda.opt has two options "min" and "max". When multiple λs lead to same error rate, "min" will return the smallest tuning parameter with the lowest error rate while "max" will return the largest one. We take cv.dsda and cv.catch as two examples.

```
obj.dsda <- cv.dsda(dat.vec$x, dat.vec$y, nfolds = 10)
obj.catch <- cv.catch(dat.ten$x, dat.ten$z, dat.ten$y, lambda.opt="min")
```

Function adjten and adjvec implement the adjustment step for tensor and vector data, respectively. It takes training tensor/vector, covariate and response as input, and outputs the adjusted tensor/vector and adjustment coefficients $\boldsymbol{\alpha}$. The adjustement step has already been incorporated into the modeling

Method	Binary			Multi-class		
Error rate (%) / Time (seconds)	Mean	SE	Time	Mean	SE	Time
DSDA/multi.modified	23.58	0.23	36.30	35.85	0.24	88.8
SeSDA	23.68	0.24	731.28	NA	NA	NA
CATCH	22.79	0.24	78.6	35.22	0.25	101.4
ℓ_1-GLM	23.99	0.16	36.23	35.66	0.21	134.4
SOS	23.87	0.26	100.8	37.07	0.29	1114.8

Table 5: ADHD classification. Average error rates based on 100 replicates and running time of 20 replicates are reported.

fitting functions dsda, msda and catch. When user input covariates along with tensor/vector, the model fitting functions will automatically make the adjustment. But if user do not want to use the automatic prediction in model fitting function and prefer to predict via predict, user need to first make the adjustment to obtain adjustment coefficient gamma, and pass it into function predict. Notice that making adjustment and fitting a model on the adjusted tensor and response without covariate is equivalent as fitting a model by inputting the original tensor, covariate and response labels. Examples of two approaches are given as follows.

```
obj <- catch(dat.ten$x, dat.ten$z, dat.ten$y, dat.ten$testx, dat.ten$testz)
obj.adj <- adjten(dat.ten$x, dat.ten$z, dat.ten$y, dat.ten$testx,
   dat.ten$testz)
obj.fit <- catch(dat.ten$x, dat.ten$z, dat.ten$y)
pred <- predict(obj.fit, obj.adj$testxres, dat.ten$z, dat.ten$testz,
   obj.adj$gamma)
```

There are three prediction functions corresponding to dsda, msda and catch, respectively. All of them can be directly called by predict and the function will recognize which function to use based on the input fitted model object. When covariate exists, user needs to pass the adjustment coefficient obtained from function adjten, the fitted model and testing data altogether to make predictions. Therefore, we encourage user to direct use model fitting functions msda and catch to fit model and predict categorical responses.

Real data example

In this section, we will show the performance of the models by a real data set. We considered the attention deficit hyperactivity disorder (ADHD) data set. The dataset is available on NITRC (http://fcon_1000.projects.nitrc.org/indi/adhd200) (Bellec et al., 2017). It contains three parts of information: s-MRI data which is a 3-D tensor, covariate information including age, gender and handedness which is a vector, and response label. Among all 930 individuals, there are four types of categorical labels: Typically Developing Childrem (TDC), ADHD Combined, ADHD Hyperactive and ADHD Inattentive.

We downsize the tensor to dimension $24 \times 27 \times 24$ and consider two classification scenarios. One is to combine ADHD Hyperactive with the ADHD Combined since there are only 13 subjects in class ADHD Hyperactive. This results in a multi-class problem with three classes. The second one is to further combine TDC and ADHD Inattentive since none of these two categories have hyperactivity symptoms. This give us a binary problem. We split the dataset into a training set and testing set by ratio 8 : 2.

Given the tensor structure and existence of covariates, the most suitable approach is to apply CATCH on that. We also vectorize the tensor into vectors and stack covariates along with the long vector to apply vector methods. For binary case, DSDA is applied. For multi-class case, MSDA model with multi.modified is applied since the dimension is too large to employ multi.original. We also compared with SOS (Clemmensen et al., 2011) by its own package **sparseLDA** and ℓ_1-GLM (Friedman et al., 2010) by package **glmnet**.

For each replicate, we perform cross validation on training data and record the classification error on testing data. The entire process was repeated for 100 times and we report the mean and standard error of the error rates. The performance is shown in Table 5.

Discussion

Package **TULIP** provides a toolbox to fit various sparse discriminant analysis models, including parametric models DSDA, ROAD, SOS for binary vector data, semiparametric model SeSDA for binary vector data, MSDA for multiclass vector data, and CATCH for multiclass tensor data. As a comprehensive toolbox, the package provides prediction and cross validation functions as well.

Meanwhile, the package propose an approach to handle cases when both predictor and covariates are supplied. The predictor can be vector and tensor, while the covariates are usually low-dimensional vectors. Covariates may have an effect on both response and the predictor. Therefore making adjustment and excluding the influence of covariates from predictor is important. The package includes functions to make the adjustments and can be called easily by supplying covariates in the model fitting function.

Appendices

Tensor notation

On each dimension, which is named mode, a tensor is composed by vectors of length $(p_k \times 1)$ called mode-k fiber, defined as $A_{i_1 \cdots i_{k-1} I_k i_{k+1} \cdots i_M}$, $I_k = 1, \dots p_k$. Stacking the mode-1 fiber by row gives the vectorization of a tensor $\text{vec}(\mathbf{A})$, which is a $(\prod_m p_m \times 1)$ column vector. If we unfold the tensor along the k-th mode, we obtain a matrix $\mathbf{A}_{(k)} \in \mathbb{R}^{p_k \times \prod_{l \neq k} p_l}$.

Denote the *mode-k product* of a tensor $\mathbf{A}$ and a matrix $\boldsymbol{\alpha} \in \mathbb{R}^{d \times p_k}$ by $\mathbf{A} \times_k \boldsymbol{\alpha} \in \mathbb{R}$, which results in a tensor of dimension $p_1 \times \cdots \times p_{k-1} \times d \times p_{k+1} \times \cdots \times p_M$. Each element of the product is the product of a mode-k fiber of $\mathbf{A}$ and a row vector of $\boldsymbol{\alpha}$. In particular, the *mode-k vector product* of a tensor $\mathbf{A}$ and a vector $\mathbf{c} \in \mathbb{R}^{p_k}$ is a $(M-1)$-way tensor as a special case when $d = 1$. The *Tucker decomposition* of a tensor is defined as $\mathbf{A} = \mathbf{C} \times_1 \mathbf{G}_1 \times_2 \cdots \times_M \mathbf{G}_M$, in short of $[\![\mathbf{C}; \mathbf{G}_1, \dots, \mathbf{G}_m]\!]$. In particular, the vectorization of tucker decomposition has the fact that $\text{vec}([\![\mathbf{C}; \mathbf{G}_1, \dots, \mathbf{G}_M]\!]) = (\mathbf{G}_M \otimes \cdots \otimes \mathbf{G}_1)\,\text{vec}(\mathbf{C})$, where $\otimes$ denotes Kronecker product. If $\mathbf{X} = \boldsymbol{\mu} + [\![\mathbf{Z}; \boldsymbol{\Sigma}_1^{1/2}, \dots, \boldsymbol{\Sigma}_M^{1/2}]\!]$, where $\mathbf{Z} \in \mathbb{R}^{p_1 \times \cdots \times p_M}$ and all elements of $\mathbf{Z}$ independently follow the univariate standard normal distribution, we say $\mathbf{X}$ follows a tensor normal distribution $\mathbf{X} \sim TN(\boldsymbol{\mu}, \boldsymbol{\Sigma}_1, \dots, \boldsymbol{\Sigma}_M)$. The dependence structure on the j-th mode is measured by $\boldsymbol{\Sigma}_j > 0$. Hence, $\text{vec}(\mathbf{X}) = \text{vec}(\boldsymbol{\mu}) + \boldsymbol{\Sigma}^{1/2}\text{vec}(\mathbf{Z})$, where $\boldsymbol{\Sigma} = \boldsymbol{\Sigma}_M \otimes \cdots \otimes \boldsymbol{\Sigma}_1$.

Simulation code

Data sets dat.vec and dat.ten are used to illustrate usage of the functions. Detailed model settings are described in Section 2.4. Here are the code to simulate the two data sets.

Code to simulate data set dat.vec:

```
set.seed(123456)
sigma <- matrix(0.3, 500, 500)
diag(sigma) <- 1
dsigma <- t(chol(sigma))
#define beta and mean
beta <- matrix(0, nrow = 500, ncol = 1)
beta[1:10,1] <- 0.5
M <- matrix(0, nrow = 2, ncol = 500)
M[2,] <- sigma%*%beta
y <- c(rep(1, 75), rep(2, 75))
#generate test data
telabel <- ceiling(runif(1000)*2)
x <- matrix(rnorm(150*500),ncol = 500)%*%t(dsigma)
x[y==2, ] <- x[y==2, ] + M[2,]
testx <- matrix(rnorm(1000*500), ncol = 500) %*% t(dsigma)
testx[telabel==2, ] <- testx[telabel==2, ] + M[2, ]
dat.vec <- list(x = x, y = y, testx = testx, testy = telabel)
```

Code to simulate data set dat.ten:

```
set.seed(123456)
sigma <- array(list(), 3) #define covariance matrices
dsigma <- array(list(), 3)
```

```
for (i in 1:3){
   sigma[[i]] <- diag(10)
   dsigma[[i]] <- t(chol(sigma[[i]]))
}
B2 <- array(0, dim=c(10,10,10)) #define B and mean
B2[1:2, 1:2, 1:2] <- 0.8
M <- array(list(), 2)
M[[1]] <- array(0, dim=c(10,10,10))
M[[2]] <- atrans(B2, sigma)
y <- c(rep(1,75), rep(2,75))
coef <- array(0, dim=c(10,10,10,2)) #define alpha
coef[1:5, 1:5, 1:5, 1] <- 1
telabel <- ceiling(runif(1000)*2)
z <- matrix(rnorm(2*150), nrow=150, ncol=2) #generate covariates
z[y==2,] <- z[y==2,] + 0.3
testz <- matrix(rnorm(2*1000), nrow=1000, ncol=2)
testz[telabel==2, ] <- testz[telabel==2, ] + 0.3
vec_x <- matrix(rnorm(1000*150), ncol=150) #generate tensor
x <- array(list(),150)
for (i in 1:150){
   x[[i]] <- array(vec_x[,i], c(10,10,10)) + amprod(coef, t(z[i,]), 4)[,,,1]
   x[[i]] <- M[[y[i]]] + atrans(x[[i]], dsigma)
}
vec_testx <- matrix(rnorm(1000*1000), ncol=1000)
testx <- array(list(), 1000)
for (i in 1:1000){
   testx[[i]] <- array(vec_testx[,i], c(10,10,10)) + amprod(coef, t(testz[i,]),
 4)[,,,1]
   testx[[i]] <- M[[telabel[i]]] + atrans(testx[[i]], dsigma)
}
dat.ten <- list(x=x, z=z, testx=testx, testz=testz, vec_x=t(vec_x),
   vec_testx=t(vec_testx), y=y, testy=telabel)
```

Estimation of covariance matrices in the TDA/CATCH model

Denote the sample mean of Class k by $\overline{\mathbf{X}}_k$. We first center $\mathbf{X}_i$ within class to obtain the residuals:

$$\widehat{\mathbf{E}}_i = \mathbf{X}_i - \widehat{\boldsymbol{\mu}}_k = \mathbf{X}_i - \overline{\mathbf{X}}_k.$$

Further unfold $\widehat{\mathbf{E}}_i$ along the j-th mode to obtain $\mathbf{W}_{i(j)}$ and find $\widetilde{\mathbf{S}}_j = (n\prod_{l\neq j}^M p_l)^{-1}\sum_{i=1}^n \mathbf{W}_{i(j)}(\mathbf{W}_{i(j)})^T$. Then our estimator for $\boldsymbol{\Sigma}_j$ is defined as

$$\widehat{\boldsymbol{\Sigma}}_j = \widetilde{s}_{j,11}^{-1}\widetilde{\mathbf{S}}_j \text{ for } j = 1,\dots,M-1; \widehat{\boldsymbol{\Sigma}}_M = \frac{\widehat{\text{var}}(X_{1\cdots 1})}{\prod_{j=1}^M \widetilde{s}_{j,11}}\widetilde{\mathbf{S}}_M. \tag{30}$$

Definition of F-test statistic

The F-test statistic used to preprocess GDS1615 data is defined as

$$f_j = \frac{\sum_{k=1}^K n_k(\widehat{\boldsymbol{\mu}}_{kj} - \widehat{\boldsymbol{\mu}}_j)^2/(K-1)}{\sum_{i=1}^n (\mathbf{X}_j^i - \widehat{\boldsymbol{\mu}}_{Y^i,j})^2/(n-K)}, \tag{31}$$

where $\mathbf{X}_j^i$ is the j-th variable of i-th observation and $\widehat{\boldsymbol{\mu}}$ is the grand mean.

Bibliography

D. Bates and M. Maechler. *Matrix: Sparse and Dense Matrix Classes and Methods*, 2016. URL https://CRAN.R-project.org/package=Matrix. R package version 1.2-6. [p143]

P. Bellec, C. Chu, F. Chouinard-Decorte, Y. Benhajali, D. S. Margulies, and R. C. Craddock. The neuro bureau adhd-200 preprocessed repository. *NeuroImage*, 144:275 – 286, 2017. URL https://doi.org/10.1016/j.neuroimage.2016.06.034. [p149]

P. J. Bickel and E. Levina. Covariance regularization by thresholding. *Ann. Statist.*, 36(6):2577–2604, 12 2008. doi: 10.1214/08-AOS600. URL https://doi.org/10.1214/08-aos600. [p134]

M. Burczynski, R. Peterson, N. Twine, K. A Zuberek, B. J Brodeur, L. Casciotti, V. Maganti, P. S Reddy, A. Strahs, F. Immermann, W. Spinelli, U. Schwertschlag, A. M Slager, M. M Cotreau, and A. J Dorner. Molecular classification of crohn's disease and ulcerative colitis patients using transcriptional profiles in peripheral blood mononuclear cells. *The Journal of Molecular Diagnostics*, 8:51–61, 03 2006. URL https://doi.org/10.2353/jmoldx.2006.050079. [p144]

T. T. Cai, W. Liu, and X. Luo. A constrained ℓ_1 minimization approach to sparse precision matrix estimation. *J. Amer. Statist. Assoc.*, 106(494):594–607, 2011. URL https://doi.org/10.1198/jasa.2011.tm10155. [p135]

E. C. Chi and T. G. Kolda. On tensors, sparsity, and nonnegative factorizations. *SIAM Journal on Matrix Analysis and Applications*, 33(4):1272–1299, 2012. URL https://doi.org/10.1137/110859063. [p135]

L. Clemmensen and M. Kuhn. *sparseLDA: Sparse Discriminant Analysis*, 2016. URL https://CRAN.R-project.org/package=sparseLDA. R package version 0.1-9. [p135]

L. Clemmensen, T. Hastie, D. Witten, and B. Ersbøll. Sparse discriminant analysis. *Technometrics*, 53(4): 406–413, 2011. URL ttps://doi.org/10.1198/TECH.2011.08118. [p134, 135, 140, 145, 149]

M. Dettling. Bagboosting for tumor classification with gene expression data. *Bioinformatics*, 20(18): 3583–3593, 2004. URL https://doi.org/10.1093/bioinformatics/bth447. [p134]

J. Fan and Y. Fan. High dimensional classification using features annealed independence rules. *Annals of statistics*, 36(6):2605, 2008. URL https://doi.org/10.1214/07-aos504. [p134, 135]

J. Fan, Y. Feng, and X. Tong. A road to classification in high dimensional space: the regularized optimal affine discriminant. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 74(4): 745–771, 2012. URL https://doi.org/10.1111/j.1467-9868.2012.01029.x. [p134, 135, 139, 145]

J. Fan, Z. T. Ke, H. Liu, and L. Xia. Quadro: A supervised dimension reduction method via rayleigh quotient optimization. *Annals of statistics*, 43(4):1498, 2015. URL https://doi.org/10.1214/14-aos1307. [p136]

J. Friedman, T. Hastie, and R. Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics Springer, Berlin, 2001. URL https://doi.org/10.1007/b94608. [p136]

J. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software, Articles*, 33(1):1–22, 2010. URL https://doi.org/10.18637/jss.v033.i01. [p143, 149]

D. Gerard and P. Hoff. *tensr: Covariance Inference and Decompositions for Tensor Datasets*, 2016. URL https://CRAN.R-project.org/package=tensr. R package version 1.0.0. [p143]

D. J. Hand. Classifier technology and the illusion of progress. *Statistical science*, 21(1):1–14, 2006. URL https://doi.org/10.1214/088342306000000024. [p134]

T. Hastie, R. Tibshirani, and A. Buja. Flexible discriminant analysis by optimal scoring. *Journal of the American statistical association*, 89(428):1255–1270, 1994. URL https://doi.org/10.1080/01621459.1994.10476866. [p134]

P. D. Hoff, X. Niu, and J. A. Wellner. Information bounds for Gaussian copulas. *Bernoulli*, 20:604–622, 2014. URL https://doi.org/10.3150/12-bej499. [p138]

B. Jiang, X. Wang, and C. Leng. Quda: A direct approach for sparse quadratic discriminant analysis. *arXiv preprint arXiv:1510.00084*, 2015. [p136]

C. Klaassen and J. Wellner. Efficient estimation in the bivariate normal copula model: normal margins are least favourable. *Bernoulli*, 3:55–77, 1997. URL https://doi.org/10.2307/3318652. [p138]

T. G. Kolda and B. W. Bader. Tensor decompositions and applications. *SIAM Review*, 51(3):455–500, 2009. ISSN 0036-1445. URL https://doi.org/10.1137/07070111x. [p135, 138]

Z. Lai, Y. Xu, J. Yang, J. Tang, and D. Zhang. Sparse tensor discriminant analysis. *IEEE Transactions on Image Processing*, 22(10):3904–3915, Oct 2013. ISSN 1057-7149. doi: 10.1109/TIP.2013.2264678. URL https://doi.org/10.1109/TIP.2013.2264678. [p135]

Q. Li and D. Schonfeld. Multilinear discriminant analysis for higher-order tensor data classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(12):2524–2537, Dec 2014. ISSN 0162-8828. doi: 10.1109/TPAMI.2014.2342214. URL `https://doi.org/10.1109/tpami.2014.2342214`. [p135]

Q. Li and J. Shao. Sparse quadratic discriminant analysis for high dimensional data. *Statistica Sinica*, 25(2):457–473, 4 2015. ISSN 1017-0405. URL `https://doi.org/10.5705/ss.2013.150`. [p136]

T.-S. Lim, W.-Y. Loh, and Y.-S. Shih. A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms. *Machine learning*, 40(3):203–228, 2000. [p134]

Y. Lin and Y. Jeon. Discriminant analysis through a semiparametric model. *Biometrika*, 90(2):379–392, 2003. ISSN 00063444. URL `https://doi.org/10.1093/biomet/90.2.379`. [p137]

H. Liu, J. Lafferty, and L. Wasserman. The nonparanormal: Semiparametric estimation of high dimensional undirected graphs. *J. Mach. Learn. Res.*, 10:2295–2328, 2009. [p138]

T. Liu, M. Yuan, and H. Zhao. Characterizing spatiotemporal transcriptome of human brain via low rank tensor decomposition. *arXiv preprint arXiv:1702.07449*, 2017. [p135]

Q. Mai and H. Zou. A note on the connection and equivalence of three sparse linear discriminant analysis methods. *Technometrics*, 55(2):243–246, 2013. URL `https://doi.org/10.1080/00401706.2012.746208`. [p140]

Q. Mai and H. Zou. Sparse semiparametric discriminant analysis. *Journal of Multivariate Analysis*, 135: 175 – 188, 2015. ISSN 0047-259X. URL `https://doi.org/10.1016/j.jmva.2014.12.009`. [p135, 140]

Q. Mai, H. Zou, and M. Yuan. A direct approach to sparse discriminant analysis in ultra-high dimensions. *Biometrika*, 99:29–42, 2012. URL `https://doi.org/10.1093/biomet/asr066`. [p134]

Q. Mai, Y. Yang, and H. Zou. Multiclass sparse discriminant analysis. *Statistica Sinica*, 29, 04 2015. URL `https://doi.org/10.5705/ss.202016.0117`. [p135, 141, 144]

D. Michie, D. J. Spiegelhalter, and C. C. Taylor. *Machine learning, neural and statistical classification*. Ellis Horwood, 1994. URL `https://doi.org/10.2307/1269742`. [p134]

Y. Niu, N. Hao, and B. Dong. A new reduced-rank linear discriminant analysis method and its applications. *Statistica Sinica*, 28, 11 2015. URL `https://doi.org/10.5705/ss.202015.0387`. [p135]

Y. Pan, Q. Mai, and X. Zhang. Covariate-adjusted tensor classification in high dimensions. *Journal of the American Statistical Association*, 114(527):1305–1319, 2019. URL `https://doi.org/10.1080/01621459.2018.1497500`. [p134, 135, 137, 138, 142]

Y. Pan, Q. Mai, and X. Zhang. *TULIP: A Toolbox for Linear Discriminant Analysis with Penalties*, 2021. URL `https://CRAN.R-project.org/package=TULIP`. R package version 1.0.2. [p135]

J. Shao, Y. Wang, X. Deng, and S. Wang. Sparse linear discriminant analysis by thresholding for high dimensional data. *The Annals of Statistics*, 39(2):1241–1265, 2011. URL `https://doi.org/10.1214/10-aos870`. [p135]

J. Sun and H. Zhao. The application of sparse estimation of covariance matrix to quadratic discriminant analysis. *BMC Bioinformatics*, 16, 12 2015. URL `https://doi.org/10.1186/s12859-014-0443-6`. [p136]

R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, Series B*, 58:267–288, 1996. URL `https://doi.org/10.1111/j.2517-6161.1996.tb02080.x`. [p139]

R. Tibshirani, T. Hastie, B. Narasimhan, and G. Chu. Diagnosis of multiple cancer types by shrunken centroids of gene expression. *Proceedings of National Academic Science (PNAS)*, 99:6567–6572, 01 2002. URL `https://doi.org/10.1073/pnas.082099299`. [p135]

N. T. Trendafilov and I. T. Jolliffe. Dalass: Variable selection in discriminant analysis via the lasso. *Computational Statistics and Data Analysis*, 51(8):3718–3736, May 2007. URL `https://doi.org/10.1016/j.csda.2006.12.046`. [p135]

W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S*. Springer, New York, fourth edition, 2002. URL `http://doi.org/10.1007/978-0-387-21706-2`. ISBN 0-387-95457-0. [p143]

D. M. Witten and R. Tibshirani. Penalized classification using fisher's linear discriminant. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 73(5):753–772, 2011. URL https://doi.org/10.1111/j.1467-9868.2011.00783.x. [p135]

M. C. Wu, L. Zhang, Z. Wang, D. C. Christiani, and X. Lin. Sparse linear discriminant analysis for simultaneous testing for the significance of a gene set/pathway and gene selection. *Bioinformatics*, 25(9):1145–1151, 2009. URL https://doi.org/10.1093/bioinformatics/btp019. [p135, 139]

P. Xu, J. Zhu, L. Zhu, and Y. Li. Covariance-enhanced discriminant analysis. *Biometrica*, 102(1):33–45, 2015. URL https://doi.org/10.1093/biomet/asu049. [p135]

M. Yuan and Y. Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(1):49–67, 2006. URL https://doi.org/10.1111/j.1467-9868.2005.00532.x. [p135, 141]

R. Zeng, J. Wu, L. Senhadji, and H. Shu. Tensor object classification via multilinear discriminant analysis network. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1971–1975, April 2015. URL https://doi.org/10.1109/icassp.2015.7178315. [p135]

W. Zhong and K. S. Suslick. Matrix discriminant analysis with application to colorimetric sensor array data. *Technometrics*, 57(4):524–534, 2015. URL https://doi.org/10.1080/00401706.2014.965347. [p135, 144]

H. Zhou, L. Li, and H. Zhu. Tensor regression with applications in neuroimaging data analysis. *Journal of the American Statistical Association*, 108(502):540–552, 2013. ISSN 0162-1459. URL https://doi.org/10.1080/01621459.2013.776499. [p135]

Yuqing Pan
Florida State University
117 N. Woodward Ave., Tallahassee, FL 32306
USA
yuqing.pan@stat.fsu.edu

Qing Mai
Florida State University
117 N. Woodward Ave., Tallahassee, FL 32306
USA
mai@stat.fsu.edu

Xin Zhang
Florida State University
117 N. Woodward Ave., Tallahassee, FL 32306
USA
henry@stat.fsu.edu

fitzRoy - An R Package to Encourage Reproducible Sports Analysis

by Robert Nguyen, James Day, David Warton and Oscar Lane

Abstract The importance of reproducibility, and the related issue of open access to data, has received a lot of recent attention. Momentum on these issues is gathering in the sports analytics community. While Australian Rules football (AFL) is the leading commercial sport in Australia, unlike popular international sports, there has been no mechanism for the public to access comprehensive statistics on players and teams. Expert commentary currently relies heavily on data that isn't made readily accessible and this produces an unnecessary barrier for the development of an inclusive sports analytics community. We present the R package **fitzRoy** to provide easy access to AFL statistics.

Introduction

Access to data is the key enabling tool for any sports analytics community. Most major international sports have a mechanism to provide free access to match statistics, for example, ballR for NBA, Lahman for baseball and deuce for tennis. Access to sports data can be used by fans, clubs and researchers to better predict match outcomes, to inform decisions and better understand the sport. For example (Romer, 2006) helped change the way teams evaluate 4th down decisions in the NFL, and the way the NBA is played has changed becoming more three point focused (Goldman and Rao, 2013). Sports analytics has also proved a popular avenue for modern data journalism for example, Fivethirtyeight is a popular culture website with a strong analytics following, which publishes models daily across a variety of sports. This sort of product can only be constructed given a publicly available source of sports data.

The Australian Football League (AFL) is the national league of Australia's national winter sport, Australian Rules Football. This is the largest commercial support in Australia with over 1 million club members, a 2.5 billion dollar broadcast rights deal and a participation level of 1.649 million. No current AFL statistics website provides easy access to data for a growing analytical fan base. The Australian Football League (AFL) has an official data provider, Champion Data, which is 49% owned by the AFL. Champion Data have the licence to collect the data for all AFL games and then charge clubs and media organisations fees to access the data. There are two leading websites of publicly available data, afltables and footywire, but data are not available in an easy-to-use form. For example, match statistics are listed on separate web pages for different matches, so hours of time would be required to compile data from over 200 different webpages in order to do an analysis across a single season. Hence, unfortunately, there are significant financial and logistical barriers to prospective analysts and fans studying the game, which stagnates progress advancing our understanding of AFL.

This paper describes the **fitzRoy** package, the first package to provide free and easy access to data on the AFL, with match and player data for the men's competition[1]. Web scraping tools have been developed that provide easy, up-to-date access to AFL match and player box statistics across the history of the game, since 1897, using open source data. The package also provides tools to link match and player data to expert tips from popular websites. For the first time, fans can evaluate the performance of tipsters themselves andcompare them to the betting market.

What is fitzRoy?

We developed **fitzRoy**, an R package that allows users to access Australian Rules Football statistics from various websites easily with R. The **fitzRoy** package allows users access to popular AFL statistics websites such as afltables and footywire. These are the two most widely used data repositories in the AFL, which have existed since the late 1990s, and while they are not official repositories, the AFL has not tried to take them offline. However, the data on these websites is not available in an easy-to-use form, e.g. match statistics are stored across different web pages for each match, so compiling season statistics would involve harvesting data from hundreds of webpages. The **fitzRoy** package compiles match or player data into a single frame, and also allows users to access popular bloggers' AFL models via the squiggle website.

A popular website called afltables contains AFL-VFL match, player and coaching stats, records

[1]fitzRoy used to contain access to the AFLW (AFL Womens) data, but unfortunately the data was removed from official AFL media, we are committed to adding AFLW data again, once it comes back

and lists from 1897[2]. The website afltables has been used in research for topics such as umpire racism (Lenten, 2017), umpires assessment of players (Lenten et al., 2019), modelling of the AFL game (Kiley et al., 2016), fixture difficulty (Lenor et al., 2016) and drafting (Lenten et al., 2018). The umpire studies would not have been possible with footywire data as umpire information isn't contained on the game pages.

The footywire website has data back to 1965, but while it does not have as many seasons of data, it has additional game variables not included in afltables. One example is Super Coach score, sometimes used as a proxy for player value (Marshall, 2017). Other examples of variables contained within footywire are tackles inside 50, intercepts and marks inside 50 to name a few.

Squiggle is a unique website in the AFL sporting landscape. It contains game analyses but it also aggregates popular AFL bloggers' tips each week. From squiggle users are able to get each models probability of win, margin prediction and a leaderboard based on bits which has been made popular by the Monash probability footy tipping competition[3].

Building fitzRoy

The name **fitzRoy** comes from the Old Fitzroy hotel in Sydney where the idea for the package was first conceived. It is also the name of one of the foundation clubs of the Australian Football League, which since merged with another club, and is now called the Brisbane Lions.

We used the R packages **Rvest** (Wickham and Wickham, 2016), **dplyr** (Wickham et al., 2015), **purrr** (Henry and Wickham, 2019) and **XML** (Temple Lang, 2020) to construct our web-scraper functions that collate data from afltables or footywire into a single data frame. These websites update immediately on completion of each round, hence so does data accessed via **fitzRoy** scraper functions. The key functions accessing afltables player and match statistics are `get_afltables_stats` and `get_afl_match_data`, respectively, and footywire data are accessed via the `get_footywire_stats` function. These functions form the backbone of the package.

Applications of fitzRoy

Match Data (Scores)

The `get_match_results` function can be used to obtain match data for any season(s) or team(s). For example to get the game scores for Fitzroy's last AFL season as follows.[4]

```
library(fitzRoy)
library(tidyverse)
library(lubridate)
fitzRoy::get_match_results()%>%
mutate(Season=lubridate::year(Date))%>%
filter(Season==1996)%>%
filter(Home.Team=="Fitzroy" | Away.Team=="Fitzroy")
```

Match Data (Players)

Fans of Australian Football like many major sports like to keep up to date with leaders of statistical categories. One statistic often of interest is goals scored. Users can come up with the leading goalkicker list for Fitzroy in 1996 as follows.

```
library(fitzRoy)
library(tidyverse)
fitzRoy::get_afltables_stats(start_date="1996-01-01",
end_date="1997-01-01")%>%
filter(Playing.for=="Fitzroy")%>%
group_by(ID, First.name, Surname)%>%
summarise(Total_Goals=sum(Goals))%>%
arrange(desc(Total_Goals))
```

[2]The first year of VFL

[3]http://probabilistic-footy.monash.edu/ footy/

[4]fans of Fitzroy Lions might want to avoid this as it only contains one win (Round 8 vs Fremantle Dockers)

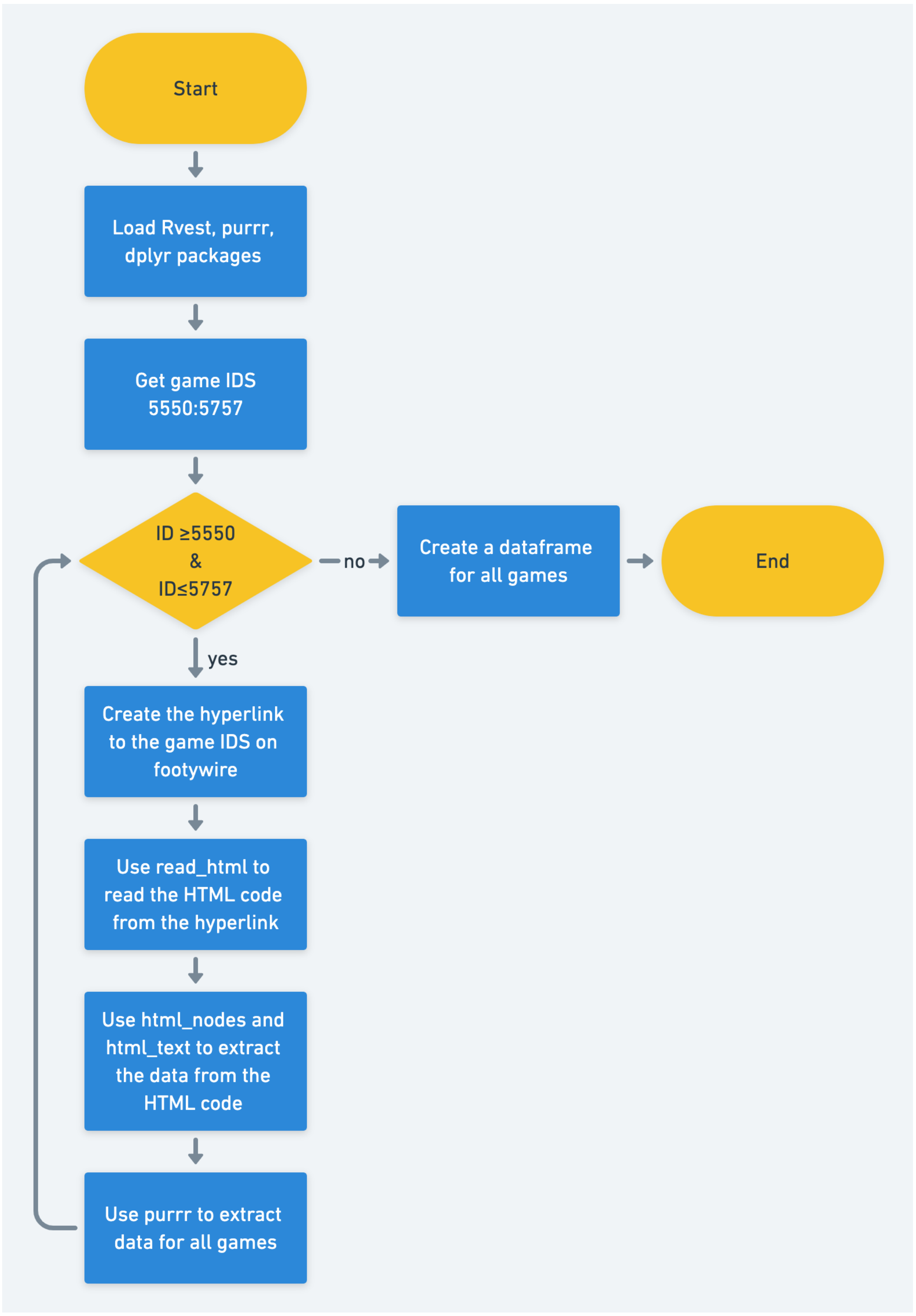

Start
Load Rvest, purrr, dplyr packages
Get game IDS 5550:5757
ID ≥5550 & ID≤5757
no
Create a dataframe for all games
End
yes
Create the hyperlink to the game IDS on footywire
Use read_html to read the HTML code from the hyperlink
Use html_nodes and html_text to extract the data from the HTML code
Use purrr to extract data for all games

Figure 1: Work flow for **fitzRoy** web-scrapers game IDS 5550 to 5757 refer to the 2013 AFLM Season.

We can see that Anthony Mellington won the goalkicking award for Fitzroy with a modest total of 22 goals for the 1996 season.

Building Sports Models

Sports models are commonly derived using an Elo system which only needs scores (Ryall and Bedford, 2010). The **fitzRoy** package readily provides a data frame of match scores, from which it is straightforward to construct an Elo to predict future match outcomes, as below.

```
library(fitzRoy)
library(tidyverse)
library(elo)
library(lubridate)

# Get data
results <- fitzRoy::get_match_results()
results <- results %>%
mutate(seas_rnd = paste0(Season, ".", Round.Number),
First.Game = ifelse(Round.Number == 1, TRUE, FALSE)
)

fixture <- fitzRoy::get_fixture()
fixture <- fixture %>%
filter(Date > max(results$Date)) %>%
mutate(Date = ymd(format(Date, "%Y-%m-%d"))) %>%
rename(Round.Number = Round)

# Simple ELO
# Set parameters (these should be optimised!)
HGA <- 30
carryOver <- 0.5
B <- 0.03
k_val <- 20

# Create margin function to ensure result is between 0 and 1
map_margin_to_outcome <- function(margin, B) {
1 / (1 + (exp(-B * margin)))
}

# Run ELO
elo.data <- elo.run(
map_margin_to_outcome(Home.Points - Away.Points, B = B) ~
adjust(Home.Team, HGA) +
Away.Team +
group(seas_rnd) +
regress(First.Game, 1500, carryOver),
k = k_val,
data = results
)

as.data.frame(elo.data)
as.matrix(elo.data)
final.elos(elo.data)

# Do predictions
fixture <- fixture %>%
mutate(Prob = predict(elo.data, newdata = fixture))

head(fixture)
```

Building Player Models

Box-score statistics, summary statistics of the involvement of each player in each match, contain a rich history of information. Box score statistics led, for example, to the concept of Value Over Replacement Player (VORP) (Woolner, 2001)[5]. Fantasy teams have gained a lot of interest in recent years, and fantasy scores for players tend to be constructed from box-score statistics.

The AFL runs a fantasy sport competition, and **fitzRoy** could be used to recreate its fantasy points formula, since it is a linear function of box-score statistics.

Box-score AFL data are made readily accessible through **fitzRoy** using the `player_stats` function.

```
library(fitzRoy)
library(tidyverse)

df<-fitzRoy::get_footywire_stats(9721:9927)

eq1<-lm(AF ~ K + HB + M + `T` + FF + FA + HO + G + B, data=df)
summary(eq1)
```

While this might seem like a trivial application, footywire only has fantasy scores going back to 2007, however the statistics used for fantasy go all the way back to 1965, with Tackles being first recorded in 1987.

```
cbPalette <- c("#999999", "#E69F00", "#56B4E9", "#009E73",
"#F0E442", "#0072B2", "#D55E00", "#CC79A7")

fitzRoy::get_afltables_stats(start_date = "1897-01-01",
end_date = "2018-10-10")%>%

group_by(Season)%>%
summarise(
meankicks=mean(Kicks),
meanmarks=mean(Marks),
meantackles=mean(Tackles),
meanfreesfor=mean(Frees.For),
meansfreesagainst=mean(Frees.Against),
meanhitouts=mean(Hit.Outs),
meangoals=mean(Goals),
meanbehinds=mean(Behinds))%>%
gather("variable", "value",-Season) %>%
ggplot(aes(x=Season, y=value, group=variable, colour=variable))+
geom_line()+
scale_colour_manual(values=cbPalette)
```

Goals have been recorded at a player level throughout the history of the game, and the most recent variable that is used in fantasy (tackles) started being recorded in 1987.

The box-score also contains time on ground so users are readily able to compute points per minute which has been a leading indicator for 2 time winner of fantasy sports Moreira Magic.[6]

Champion Data publish Super Coach scores, valued by clubs to inform recruitment decisions and fantasy sport competitions. However the formula for Super Coach scores is propriety and not in the public domain. Following the release of fitzRoy, one well-known blogger attempted to re-create it with a linear model, and managed an R-squared of 91.6[7][8]

Able to Compare Popular Models

Blogging has taken off around the world with popular websites such as fansided, fivethirtyeight and the ringer proving popular among the overseas sporting community. To help promote other people

[5]https://www.theringer.com/mlb/2018/2/20/17030428/sherri-nichols-baseball-sabermetric-movement

[6]https://player.whooshkaa.com/shows/chilling-with-charlie

[7]http://www.matterofstats.com/mafl-stats-journal/2018/10/7/a-first-attempt-at-combining-afl-team-and-player-data-in-a-predictive-model

[8]This is impressive as Champion data uses data not available within **fitzRoy** and its weighted by time and game margin.

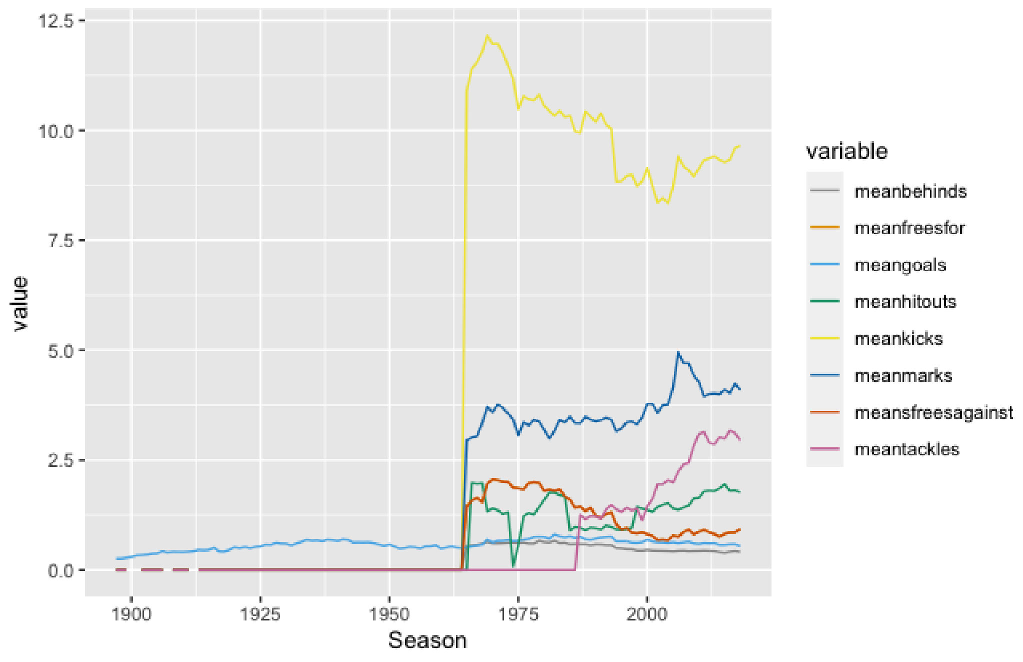


Figure 2: Line graph of mean values of AFLM statistics - By seeing when the line jumps from zero is a quick way to see when a statistic was first collected.

who do modeling work and make their work available online, we provide access to squiggle, the most popular aggregator website in the AFL. This means that the behaviour of different tipsters' models can be analysed easily.

```
library(fitzRoy)
fitzRoy::get_squiggle_data("tips")
```

The above command will enable a user to get the tips from popular blogging sites such as squiggle, matterofstats and liveladders among many. This means that different tipsters models behaviours can be analysed easily. For example studying how they take into account home ground advantages.

Future Developments

The developers of **fitzRoy** is committed to giving users the data to analyse the game. In the future this means updating the Womens AFLW data once it becomes available online, updating the scrapers to include the AFL website.

Summary

The **fitzRoy** package offers a springboard for sports analytics in the AFL community. It provides easy access to publicly available AFL data, and we have illustrated how this can be used to predict match outcomes and to rank players. In future work we plan to build a statistical model for AFL match outcomes and its key predictors, along the lines of (Yurko et al., 2018; Deshpande and Jensen, 2016). There are endless possibilities: clubs might use it to inform on player recruitment (O'Shaughnessy, 2010) and team style (Greenham et al., 2017); researchers and enthusiasts can use it to better understand the game; there are obvious betting implications (Bailey, 2000); and educators can use it as a teaching tool.

The **fitzRoy** package was only released in 2018 but has already been used by AFL club analysts and bloggers who are now able to access data and develop content they weren't previously able to do (*e.g.* VORP). The AFL analytics community is develops rapidly, and it is exciting to see where it will go over the coming seasons.

Bibliography

M. Bailey. Identifying arbitrage opportunities in afl betting markets through mathematical modelling. In *Proceedings of the Fifth Australian Conference in Mathematics and Computers in Sport*, pages 37–42, 2000. [p160]

S. K. Deshpande and S. T. Jensen. Estimating an nba player's impact on his team's chances of winning. *Journal of Quantitative Analysis in Sports*, 12(2):51–72, 2016. [p160]

M. Goldman and J. M. Rao. Live by the three, die by the three? the price of risk in the nba. In *Submission to the MIT Sloan Sports Analytics Conference*, 2013. [p155]

G. Greenham, A. Hewitt, and K. Norton. A pilot study to measure game style within australian football. *International Journal of Performance Analysis in Sport*, 17(4):576–585, 2017. URL https://doi.org/10.1080/24748668.2017.1372163. [p160]

L. Henry and H. Wickham. *purrr: Functional Programming Tools*, 2019. URL https://CRAN.R-project.org/package=purrr. R package version 0.3.2. [p156]

D. P. Kiley, A. J. Reagan, L. Mitchell, C. M. Danforth, and P. S. Dodds. Game story space of professional sports: Australian rules football. *Physical Review E*, 93(5):052314, 2016. URL https://doi.org/10.1103/PhysRevE.93.052314. [p156]

S. Lenor, L. J. Lenten, and J. McKenzie. Rivalry effects and unbalanced schedule optimisation in the australian football league. *Review of Industrial Organization*, 49(1):43–69, 2016. doi: https://doi.org/10.1007/s11151-015-9495-7. [p156]

L. J. Lenten. Racial discrimination in umpire voting: an (arguably) unexpected result. *Applied Economics*, 49(37):3751–3757, 2017. URL https://doi.org/10.1080/00036846.2016.1267848. [p156]

L. J. Lenten, A. C. Smith, and N. Boys. Evaluating an alternative draft pick allocation policy to reduce 'tanking'in the australian football league. *European Journal of Operational Research*, 267(1):315–320, 2018. URL https://doi.org/10.1016/j.ejor.2017.11.029. [p156]

L. J. Lenten, P. Crosby, and J. McKenzie. Sentiment and bias in performance evaluation by impartial arbitrators. *Economic Modelling*, 76:128–134, 2019. URL https://doi.org/10.1016/j.econmod.2018.07.026. [p156]

K. Marshall. The effect of leadership on afl team performance. In *Proceedings of MathSport International 2017 Conference*, page 255, 2017. [p156]

D. O'Shaughnessy. On the value of afl player draft picks. In *10th MathSport Conference, Darwin, Australia*, 2010. [p160]

D. Romer. Do firms maximize? evidence from professional football. *Journal of Political Economy*, 114(2):340–365, 2006. URL https://doi.org/10.1086/501171. [p155]

R. Ryall and A. Bedford. An optimized ratings-based model for forecasting australian rules football. *International Journal of Forecasting*, 26(3):511–517, 2010. URL https://doi.org/10.1016/j.ijforecast.2010.01.001. [p158]

D. Temple Lang. *XML: Tools for Parsing and Generating XML Within R and S-Plus*, 2020. URL https://CRAN.R-project.org/package=XML. R package version 3.99-0.5. [p156]

H. Wickham and M. H. Wickham. Package 'rvest'. *URL: https://cran. r-project. org/web/packages/rvest/rvest. pdf*, 2016. [p156]

H. Wickham, R. Francois, L. Henry, K. Müller, et al. dplyr: A grammar of data manipulation. *R package version 0.4*, 3, 2015. [p156]

K. Woolner. Introduction to vorp: Value over replacement player. *Retrieved from Stathead. com: https://web. archive. org/web/20070928064958/http://www. stathead. com/bbeng/woolner/v orpdescnew. htm*, 2001. [p159]

R. Yurko, S. Ventura, and M. Horowitz. nflwar: A reproducible method for offensive player evaluation in football. *arXiv preprint arXiv:1802.00998*, 2018. URL https://doi.org/10.1515/jqas-2018-0010. [p160]

Robert N. Nguyen
School of Mathematics and Statistics
University of New South Wales
Sydney,NSW 2052 Australia
robert.nguyen@unsw.edu.au

James T. Day
Fusion Sport
Australia
jamesthomasday@gmail.com

David I. Warton
School of Mathematics and Statistics and
Evolution & Ecology Research Centre
University of New South Wales
Sydney,NSW 2052 Australia
David.Warton@unsw.edu.au

Oscar Lane
lane.oscar@gmail.com

Assembling Pharmacometric Datasets in R - The puzzle Package

by Mario González-Sales, Olivier Barrière*, Pierre Olivier Tremblay, Guillaume Bonnefois, Julie Desrochers and Fahima Nekka*

Abstract

Pharmacometric analyses are integral components of the drug development process. The core of each pharmacometric analysis is a dataset. The time required to construct a pharmacometrics dataset can sometimes be higher than the effort required for the modeling *per se*. To simplify the process, the puzzle R package has been developed aimed at simplifying and facilitating the time consuming and error prone task of assembling pharmacometrics datasets.

Puzzle consist of a series of functions written in R. These functions create, from tabulated files, datasets that are compatible with the formatting requirements of the gold standard non-linear mixed effects modeling software.

With only one function, `puzzle()`, complex pharmacometrics databases can easily be assembled. Users are able to select from different absorption processes such as zero- and first-order, or a combination of both. Furthermore, datasets containing data from one or more analytes, and/or one or more responses, and/or time dependent and/or independent covariates, and/or urine data can be simultaneously assembled.

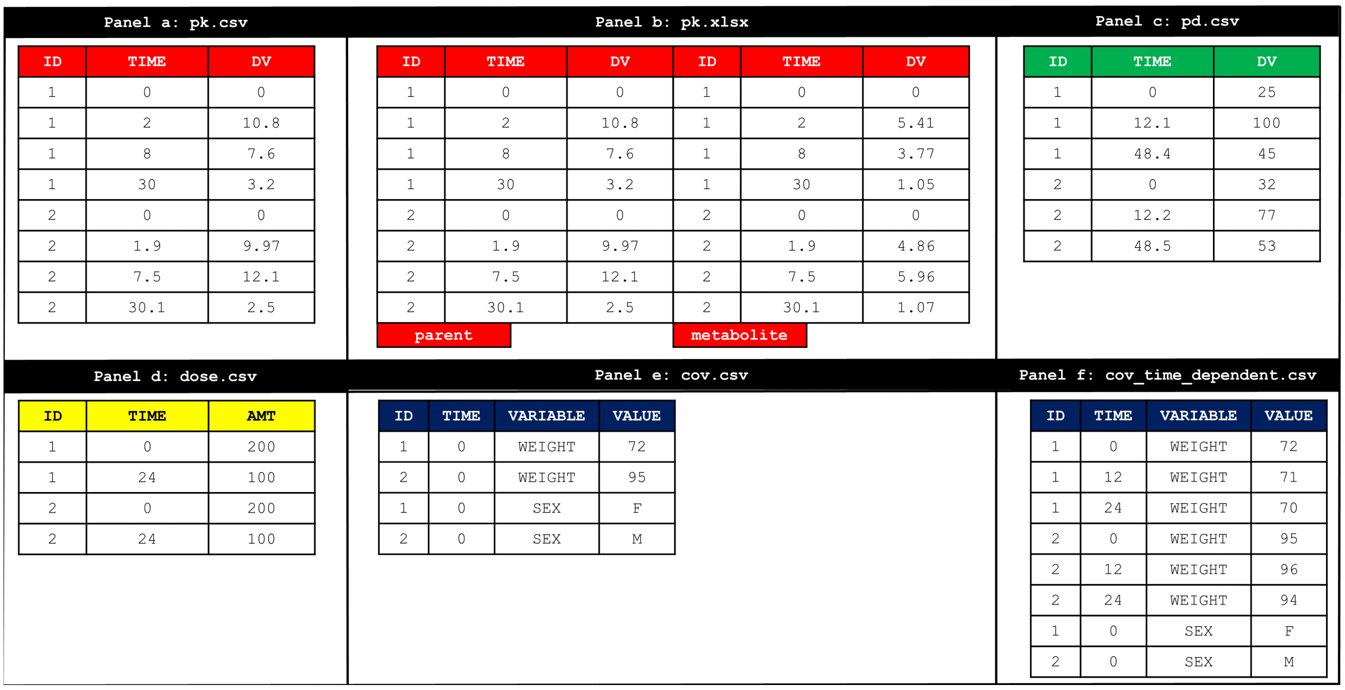
The **puzzle** package is a powerful and efficient tool that helps modelers, programmers and pharmacometricians through the challenging process of assembling pharmacometrics datasets.

Introduction

The pharmacometrics workflow has routine steps: 1) assemble the dataset, 2) explore, 3) model the data, 4) evaluate, 5) validate the model, and 6) communicate the findings. The automation of these steps saves time and money, reduces the risk of errors, and increases reproducibility. Currently, a number of excellent tools are available to enhance steps 2-6 (Jonsson and Karlsson, 1999; Lindbom et al., 2005; Keizer et al., 2011; Keizer, 2015; Mouksassi, 2016; Wang et al., 2016; Keizer, 2018; Xie et al., 2018; Mouksassi, 2019; Baron, 2019; RStudio team). However, to the best of our knowledge, there is no open source tool to support step 1). Considerable challenges exist when working in data management with an industrial setting that must comply with rules and regulations under the scrutiny and control of regulatory agencies, such as the U.S. Food and Drug Administration (FDA), Health Canada, the Japanese Pharmaceutical and Medical Device Agency (PMDA), or the European Medicines Agency (EMA). The task of dataset building should not be underrated, since the amount of time required to construct a pharmacometrics dataset can be sometimes higher than the effort required for the modeling *per se*. In fact, it is often claimed that the process of cleaning and preparing the data could take up to 80% of data analysis (Dasu and Johnson, 2003). In addition, data preparation does not amount to a single step, since it usually has to be repeated over the course of analysis as new data become available.

Understanding the structure of pharmacometrics datasets is essential for an efficient data assembling. These datasets are two-dimensional arrangements of data in rows and columns. Each row represents a record or an event, while each column represents an item or a variable. Pharmacokinetic (PK) datasets are generally time-ordered arrangements of records representing the time course of plasma concentrations related to the dose of drug administered. Depending on the complexity of the system being modeled, pharmacodynamics (PD) datasets might or not include dose and/or time information. Furthermore, pharmacometrics datasets normally imply multiple doses, several routes of administration, different treated arms and/or sequences, time dependent and/or independent covariates, metabolite, and/or urine data, and/or information regarding one or multiple PD endpoints. These additional components tend to skyrocket the complexity of the data assembling as it makes the process hard to script up-front. This is especially true when working with data that are not in tidy shape (Wickham et al., 2014). Due to this entanglement, complications are likely to happen throughout the process. In this regard, inconsistencies between exact dates, times of dose and blood sampling, different identifiers on the same requisition forms, imputation of missing and time-varying covariates, and missing timing of concomitant medications are common issues (Grasela et al., 2007).

In this tutorial, we present the **puzzle** package to the pharmacometrics community, an open source tool for assembling pharmacometrics datasets in R. The **puzzle** package consists in a group of functions developed to simplify and facilitate the time consuming and error prone task of assembling



Panel a: pk.csv

ID	TIME	DV
1	0	0
1	2	10.8
1	8	7.6
1	30	3.2
2	0	0
2	1.9	9.97
2	7.5	12.1
2	30.1	2.5

Panel b: pk.xlsx

ID	TIME	DV	ID	TIME	DV
1	0	0	1	0	0
1	2	10.8	1	2	5.41
1	8	7.6	1	8	3.77
1	30	3.2	1	30	1.05
2	0	0	2	0	0
2	1.9	9.97	2	1.9	4.86
2	7.5	12.1	2	7.5	5.96
2	30.1	2.5	2	30.1	1.07

parent metabolite

Panel c: pd.csv

ID	TIME	DV
1	0	25
1	12.1	100
1	48.4	45
2	0	32
2	12.2	77
2	48.5	53

Panel d: dose.csv

ID	TIME	AMT
1	0	200
1	24	100
2	0	200
2	24	100

Panel e: cov.csv

ID	TIME	VARIABLE	VALUE
1	0	WEIGHT	72
2	0	WEIGHT	95
1	0	SEX	F
2	0	SEX	M

Panel f: cov_time_dependent.csv

ID	TIME	VARIABLE	VALUE
1	0	WEIGHT	72
1	12	WEIGHT	71
1	24	WEIGHT	70
2	0	WEIGHT	95
2	12	WEIGHT	96
2	24	WEIGHT	94
1	0	SEX	F
2	0	SEX	M

Figure 1: Structure of the four different files to input data into `puzzle()` function. *Panels a and b illustrate the items required to input PK information from one and two analytes, respectively. Panel c and d depict the items to input PD and dosing information. Panel e and f presents the pre-formatting requirements to input covariate information assuming time in- and dependent covariates, respectively.*

pharmacometrics datasets. The datasets created are compatible with the formatting requirements of the NONMEM® software (Beal et al., 2009). Moreover, as NONMEM® data structures are the gold standard for non-linear mixed effects modeling, the datasets created with the **puzzle** package are mostly compatible with other non-linear mixed effects softwares such as MONOLIX (Lixoft), **saemix** (Comets et al., 2011), and **nlmixr** (Fidler et al., 2019). The **puzzle** package is also available on CRAN and can be installed with the following R code: `install.packages("puzzle")`.

This tutorial proceeds as follows. First, we illustrate two common case studies of data assembling of pharmacometrics datasets using the **puzzle** package. Second, we dive into the arguments of the main function of the **puzzle** package, the `puzzle()` function. Third, we present the pre-formatting requirements of `puzzle()`, and finally, we conclude with a discussion regarding the limitations and inconveniences of this framework, and what are the other approaches or efforts that might be advantageous to pursue.

Use of the puzzle function

The `puzzle()` function is flexible enough to build all types of pharmacometrics datasets in the appropriate format for the pharmacometrics analysis. The data presented in Figure 1 will be used to show the readers the flexibility of the `puzzle()` function and how powerful it is. In the first example, panels b, d, and e of Figure 1 will be used to assemble a PK dataset containing parent and metabolite plasma concentrations, several covariates, and multiple administrations. The second example will imply panels a, c, d and f of Figure 1. The output dataset will be a PK/PD data set with time dependent covariates.

Example 1: PK data set

The syntax to create the NM_PK.csv depicted in Table 1 is as follows:

```
library(puzzle)
directory = "C:/projects/puzzle/data"
puzzle(directory=file.path(directory),
       order=c(1,1),
       pk=list(name="pk.xlsx"),
       dose=list(name="dose.csv"),
       cov=list(name="cov.csv"),
       nm=list(name="NM_PK.csv"),
       username="Mario Gonzales Sales")
```

C	ID	TIME	TAD	DOSETIME	PDOSETIME	NUMDOSE	AMT	CMT	EVID	DV	LDV	MDV	SEX	WEIGHT
	1	0	0	0	0	1	200	1	1	.	.	1	0	72
	1	0	0	0	0	1	200	2	1	.	.	1	0	72
	1	0	0	0	0	1	.	3	0	0	.	0	0	72
	1	0	0	0	0	1	.	4	0	0	.	0	0	72
	1	2	2	0	0	1	.	3	0	10.8	2.37954613	0	0	72
	1	2	2	0	0	1	.	4	0	5.41	1.68824909	0	0	72
	1	8	8	0	0	1	.	3	0	7.6	2.02814825	0	0	72
	1	8	8	0	0	1	.	4	0	3.77	1.327075	0	0	72
	1	24	0	24	0	2	100	1	1	.	.	1	0	72
	1	24	0	24	24	2	100	2	1	.	.	1	0	72
	1	30	6	24	24	2	.	3	0	3.2	1.16315081	0	0	72
	1	30	6	24	24	2	.	4	0	1.05	0.04879016	0	0	72
	2	0	0	0	0	1	200	1	1	.	.	1	1	95
	2	0	0	0	0	1	200	2	1	.	.	1	1	95
	2	0	0	0	0	1	.	3	0	0	.	0	1	95
	2	0	0	0	0	1	.	4	0	0	.	0	1	95
	2	1.9	1.9	0	0	1	.	3	0	9.97	2.29958058	0	1	95
	2	1.9	1.9	0	0	1	.	4	0	4.86	1.58103844	0	1	95
	2	7.5	7.5	0	0	1	.	3	0	12.1	2.49320545	0	1	95
	2	7.5	7.5	0	0	1	.	4	0	5.96	1.78507048	0	1	95
	2	24	0	24	0	2	100	1	1	.	.	1	1	95
	2	24	0	24	24	2	100	2	1	.	.	1	1	95
	2	30.1	6.1	24	24	2	.	3	0	2.5	0.91629073	0	1	95
	2	30.1	6.1	24	24	2	.	4	0	1.07	0.06765865	0	1	95

Table 1: Returned output from the `puzzle()` function after running example 1

After running the `puzzle()` function the following message will be printed in the R console:

```
Automatic coercion to numeric for CMT
3=parent
4=metabolite
Automatic coercion to numeric for SEX
0=F
1=M
Assembling date and time: 2019-10-29 12:12:47
Time zone: Europe/Paris
Number of individuals: 2
Number of observations: 16
Dose levels: “100”, “200”
This data set was assembled by Mario Gonzalez Sales
```

It informs the user that parent and metabolite records are located in CMT 3 and 4, respectively. Moreover, it also indicates that the categorical variable SEX has been coerced into a numeric variable. Female (F) and male (M) categories have been assigned to the values of 0 and 1, respectively. The user should be aware that factors are coerced following alphabetical order. The date and time at which the dataset was assembled, the timezone where the assembling was performed, the number of individuals and observations in the dataset, the different dose levels administered, and if requested, the person who performed the data assembling will be automatically displayed.

The `puzzle()` function simultaneously assembles the PK (*i.e.* parent and metabolite), the dose and the covariate information returning a NONMEM® ready dataset stored in a .csv file (*i.e.* "NM_PK.csv").

Because order has been set to `order = c(1,1)`, the dose has been split into two compartments (*i.e.* CMT 1 and 2). Furthermore, because there are two analytes, the observations have been assigned to an independent compartment (*i.e.* CMT 3 and 4) for the parent and the metabolite, respectively. Additionally, the `puzzle()` function automatically appends useful items to the output. Specifically, a placeholder to ignore records (C), time after dose (TAD), dosing time (DOSETIME), prior dosing time (PDOSETIME), the compartment item (CMT), the event identification (EVID), the log of the DV (LDV) and the missing dependent variable item (MDV).

Example 2: PK/PD data set

The syntax to create the NM_PKPD.csv depicted in Table 2 is as follows:

```
puzzle(directory=file.path(directory),
       order=0,
       pk=list(name="pk.csv"),
       pd=list(name="pd.csv"),
       dose=list(name="dose.csv"),
       cov=list(name="cov_time_dependent.csv"),
       coercion=list(name="coercion.txt"),
       nm=list(name="NM_PKPD.csv"))
```

C	ID	TIME	TAD	DOSETIME	PDOSETIME	NUMDOSE	AMT	TYPE	CMT	EVID	DV	LDV	MDV	SEX	WEIGHT
	1	0	0	0	0	1	200	0	1	1	.	.	1	0	72
	1	0	0	0	0	1	.	1	1	0	0	.	0	0	72
	1	0	0	0	0	1	.	2	2	0	25	3.21887582	0	0	72
	1	2	2	0	0	1	.	1	1	0	10.8	2.37954613	0	0	72
	1	8	8	0	0	1	.	1	1	0	7.6	2.02814825	0	0	72
	1	12.1	12.1	0	0	1	.	2	2	0	100	4.60517019	0	0	71
	1	24	0	24	0	2	100	0	1	1	.	.	1	0	70
	1	30	6	24	24	2	.	1	1	0	3.2	1.16315081	0	0	70
	1	48.4	24.4	24	24	2	.	2	2	0	45	3.80666249	0	0	70
	2	0	0	0	0	1	200	0	1	1	.	.	1	1	95
	2	0	0	0	0	1	.	1	1	0	0	.	0	1	95
	2	0	0	0	0	1	.	2	2	0	32	3.4657359	0	1	95
	2	1.9	1.9	0	0	1	.	1	1	0	9.97	2.29958058	0	1	95
	2	7.5	7.5	0	0	1	.	1	1	0	12.1	2.49320545	0	1	95
	2	12.2	12.2	0	0	1	.	2	2	0	77	4.34380542	0	1	96
	2	24	0	24	0	2	100	0	1	1	.	.	1	1	94
	2	30.1	6.1	24	24	2	.	1	1	0	2.5	0.91629073	0	1	94
	2	48.5	24.5	24	24	2	.	2	2	0	53	3.97029191	0	1	94

Table 2: Returned output from the `puzzle()` function after running example 2

In this case, the variable type of observation (*i.e.* TYPE) has been appended to the "NM_PKPD.csv" file. Because a bolus administration has been specified with the argument `order` (*i.e.* `order = 0`), TYPE has a value of 0 for dosing events, and a value of 1 and 2 for PK and PD records, respectively. Moreover, the file "coercion.txt" has been generated. The content is printed below:

```
CMT: 1=pk, 2=pd
SEX: 0=F, 1=M
```

The argument of the puzzle function

The `puzzle()` function builds pharmacometrics ready datasets from a group of tabulated files. For convenience, it always returns a ".csv" file or an R object of class "data.frame" as output. The path to the location of the files to be assembled can be set with the `directory` argument. Depending on the pharmacometric analysis to be performed, the tabulated files may contain PK, dose, covariates and/or PD information.

The PK data may include drug concentration (*i.e.* parent and/or metabolite/s) and/or urine information. The `pk` argument is used to input this type of data into the `puzzle()` function. The general form is as follows:

```
pk = list(name = NULL, data = NULL)
```

If the PK data is stored in a ".csv" or an ".xlsx" file, the name of the file should be provided with name:

```
pk = list(name = "pk.csv", data = NULL)
```

Alternatively, if the data is within the R environment, the name of the R object may be defined with the parameter data:

```
pk = list(name = NULL, data = nm$pk)
```

where nm is a list containing the PK information pk. Excel files are useful to simultaneously store PK information from two or more analytes. The puzzle() function is smart enough to assemble all the PK information at once. To this end, each analyte data has to be contained in its own spreadsheet. Moreover, each sheet has to have the same items, and the items have to be in the same order.

The PD information may comprise the time course of the clinical endpoint of interest, time to event data, and/or a response outcome. This type of information is handled with the pd argument. Its behavior is similar to pk, and thus, it accepts the same type of tabulated files and it requires a similar syntax. For example, in case of multiple endpoints stored in an ".xlsx" file, the user should define:

```
pd = list(name = "pd.xlsx", data = NULL)
```

or simply

```
pd = list(name = "pd.xlsx")
```

Consistently, covariate and dose information are inputted to the puzzle() function with the arguments cov and dose, respectively. The reader should note that ".xlsx" files are not required, and the data should be contained within a ".csv" file or an R object. Finally, extratimes is another argument following this logic. This argument can be used to add a common pattern of additional times for each individual within the dataset. The user may be interested in this feature for prediction purposes. For example, if the data are very sparse (*i.e.* 2 observations per individual from time 1 to 4 hours post drug administration), it is possible to define a vector of times from time 0 to 4 hours post dose in order to obtain a smooth prediction (*e.g.* extratimes = list(name = "extratimes.csv",data = NULL)). Of note, EVID = 2 will be assigned to these extra times.

The cov and dose arguments may be used in combination with a number of additional arguments: optionalcolumns, which defines the optional columns to be appended in to the output. For instance, if more than one treatment is administered, the file storing the dose information may contain the variable treatment (*i.e.* "TRT"). If the user wants to include this variable in the pharmacometrics dataset, the following syntax is warranted: optionalcolumns = "TRT". If more than one variable are to be appended, the syntax should be: optionalcolumns = c("variable 1","variable 2"). Furthermore, the argument fillcolumns fills the columns appended with the argument optionalcolumns using the last observation carried forward method. If the argument is defined as fillcolumns = "TRT", the variable TRT will be added without missing values. Otherwise, the value of TRT only will be available for dosing records, and non-dosing records will be outputted as ".", the default for missing values.

The puzzle() function is able to coerce numeric variables from character variables. If a character variable is introduced with the cov argument, this variable will be automatically coerced into a numeric variable. For example, let's assume we have a character variable accounting for renal impairment with the following possible values: "mild", "moderate" and "severe". The puzzle() function will convert the variable from character to factor. Each string will be a different level. Then, the factor variable will be coerced into numeric. If there is no numeric order within the factor, the levels will be alphabetically assigned. The argument initialindex defines the initial value of the coerced numeric variable. The default value is initialindex = 0. Therefore, under the default settings, the "mild", "moderate" and "severe" levels will have numeric values of 0, 1 and 2, respectively. Moreover, a string for missing values can be defined with the argument na.strings. Thus, if a variable has a missing value, this can be labeled as not available as follows: na.strings = "N/A". For convenience, a ".txt" file may be created with the argument coercion. This argument creates a record of the categories of each coerced variable. If the user defines coercion = list(name = "my_coercion_records.txt"), a file with the name "my_coercion_records.txt" will be generated in the working directory after running the puzzle() function. The user can also control the variables to be included in the records with the argument nocoercioncolumns.

When performing population PK analysis in NONMEM®, the structure of the data set depends on the route of administration. This implies that the data structure is different for bolus, infusion and oral administrations (Owen and Fiedler-Kelly, 2014). The argument order is used to handle this situation. It can take four different values: order = 0 indicates a zero-order absorption process. It may be used for bolus or infusion administrations. For the later, the RATE has to be given in the dosing file; order = 1 serves to generate datasets to model first-order absorption processes (*e.g.* oral administration); order = c(1,1) may be used to model the absorption of drugs with complex formulations. An example may be a drug formulated as immediate and extended release. In this situation, two different absorption rates may be warranted, and c(1,1) means that the dataset is built assuming two first-order absorption processes. Finally, `order = c(0,1)` allows the user to define a zero- and first-order absorption processes. By default, the `puzzle()` function assumes this process occurs in parallel, meaning that one fraction of the drug is absorbed thorough a zero-order process and the remaining amount of the drug is absorbed following a first-order process. However, it may also follow a sequential process where the drug is delivered thorough a zero-order process to the depot compartment, and then, absorbed into the bloodstream following a first-absorption process. The argument `parallel` can be set to false (*i.e.* `parallel = FALSE`) to control these absorption patterns. The user must be aware that if order is not set to `order = c(0,1)` and `parallel = FALSE`, `puzzle()` will return the following error message:

Error in puzzle(directory = file.path(getwd()), order = c(1, 1), parallel = F,: Would you like to use a sequential zero + first order absorption model? Please set order=c(0,1). Otherwise, please set parallel = T

A characteristic of the NONMEM® datasets is that characters are not allowed, and all the items, except DAT, must be numeric. Hence, the command IGNORE in the $INPUT block within a NONMEM® control stream is commonly used to ignore the label of the data items. The `puzzle()` function has the argument `ignore` to create a new item for this purpose. For example, if the user sets ignore = "C", the first column of the returned ".csv" file will contain an item called C. The argument `missingvalues` allows the user to specify a label for the missing values. The default value is `missingvalues = "."`.

The user can control how the records are arranged with the argument `arrange`. By default, records are arranged as follows: `arrange="ID,TIME,CMT,desc(EVID)"`. Moreover, complex date formatting is also supported. The argument `datetimeformat` defines the format for dates and times. By default, the format is `datetimeformat = "%Y-%m-%d %H:%M:%S"`. In addition to that, time units may be specified with the argument `timeunits`. For example, `timeunits = "hours"`. It should be highlighted that the timezone will affect how times are computed from dates. Thus, depending on your location, times may be slightly different because the default value is `timezone = Sys.timezone()`. The user can also identify the person assembling the dataset with the argument `username`. The last argument of the `puzzle()` function is `nm`. It is used to name the output from the function, in other words, the returned ".csv" file. The syntax of the arguments of the `puzzle()` function is presented in Table 3.

Pre-formatting requirements

At the beginning of a new project, the modeler or the programmer usually faces a series of large and structurally diverse datasets where the required information for the analysis is stored. It is a common practice that data are collected using automatic informatics methods. Unfortunately, the systems may storage the information in multiple and non-tidy pre-defined files. This situation is not ideal to assemble the pharmacometrics datasets because the programmer has to spend valuable time puzzling out and assembling all the heterogeneous information. The `puzzle()` function generates NONMEM® formatted datasets from standard tabulated files. Consequently, before calling the `puzzle()` function, it may be necessary to perform some data manipulation (Wickham and Grolemund, 2016; Chen et al., 2017). The degree of data manipulation will depend on the structure of the stored information. Given the high combination of possible data structures, detailing how the data should be manipulated is out of the scope of this manuscript. The reader is referred to the documentation of the **tidyverse** package for more information regarding how to efficiently perform this step (Wickham, 2017).

Pharmacometrics datasets usually contain PK, PD, dose and/or covariate information. The `puzzle()` function requires each type of information to be inputted from a separate tabulated file. Each file has to contain pre-defined variables with specific labels.

Pre-formatting requirements for the tabulated file containing the PK information

This file has to have at least three columns: i) a column used for subject identification "ID"; ii) a column containing the sampling times at which the PK samples were collected "TIME" or "DATETIME"; and iii) the observed value of the dependent variable "DV". The tabulated file can be an R object of class list, or a ".csv" file in case of assembling data from one analyte (Figure 1: panel a). If information from more than one analyte is to be assembled (Figure 1: panel b), the data should be stored in an ".xlsx", or in an R object of class list. For example, if parent and metabolite information are available, the `puzzle()` function requires an ".xlsx" file with one sheet containing the PK information of the parent drug, and another sheet storing the PK information of the metabolite. There is no limit in the number of sheets that the `puzzle()` function can handle. However, it is mandatory that each sheet includes the three pre-defined columns mentioned above.

Argument	Example of syntax
directory	`directory = file.path(getwd())`
pk	`pk = list(name="pk.csv")`
pd	`pd = list(name="pd.csv")`
cov	`cov = list(name="cov.csv")`
dose	`dose = list(name="dose.csv")`
extratimes	`extratimes = list(name="extratimes.csv")`
optionalcolumns	`optionalcolumns = "TRT"`
fillcolumns	`fillcolumns = "TRT"`
initialindex	`initialindex = 0`
na.strings	`na.strings = "N/A"`
coercion	`coercion = list(name = "coercion.txt")`
nocoercioncolumns	`nocoercioncolumns = NULL`
order	`order = c(0,1)`
parallel	`parallel = FALSE`
ignore	`ignore = "C"`
missingvalues	`missingvalues = "."`
arrange	`arrange = "ID,TIME,CMT,desc(EVID)"`
datetimeformat	`datetimeformat="%Y-%m-%d %H:%M:%S"`
timeunits	`timeunits="hours"`
timezone	`timezone=Sys.timezone()`
username	`Username="User"`
nm	`nm=list(name="NONMEM.csv")`

Table 3: Syntax of the arguments of the `puzzle()` function

Pre-formatting requirements for the tabulated file containing the PD information

The `puzzle()` function has the same requirements for the PD than for the PK information. Therefore, the same three items (*i.e.* ID, TIME or DATETIME, and DV) are mandatory and depending on the number of PD endpoints available, the data can be stored in an R object of class list, a ".csv" or an ".xlsx" file (Figure 1: panel c).

Pre-formatting requirements for the tabulated file containing the dosing records

At least three columns are required: i) a column used as subject identification "ID": ii) a column containing the times at which the doses were administered "TIME" or "DATETIME"; and iii) the given dose "AMT" (Figure 1: panel d). In addition to that, and depending on the study design, other columns may be present as well. Later on, these columns may be passed to the returned ".csv" file using the `optionalcolumns` argument.

Pre-formatting requirements for the tabulated file containing the covariate information

If this file is used, it has to include the following three items: i) a column with subject identification "ID"; ii) the name of the variable "VAR", and iii) the value of the variable "VALUE" (Figure 1: panel e). Of note, the `puzzle()` function is able to handle time independent (*e.g.* sex) or time dependent covariates (Figure 1: panel f). In the specific case of time dependent covariates, a fourth item is mandatory: iv) a column containing the times at which the covariates were collected "TIME" or "DATETIME". This is required so that the `puzzle()` function can know the times at which the value of the covariate changes within a subject.

Discussion

The **puzzle** package has been designed to be used as simple as possible. In fact, with a few lines of R code, modelers, programmers and overall pharmacometricians have now an open source tool to assemble complex pharmacometrics datasets. In order to facilitate its use and to decrease the slope of the learning curve, users are only required to learn the behavior and syntax of one function, `puzzle()`. Nevertheless, the **puzzle** package involves additional functions intentionally working "under the hood" to enhance the user experience. These functions are borrowed from several R libraries including: utils, lubridate, stats, readxl, reshape2, sqldf, plyr, and dplyr. The **puzzle** package started to be coded more than six years ago. At the time, reshape2, plyr, readxl and sqldf were useful tools from a data assembling perspective. However, if the **puzzle** package was started to be coded from scratch nowadays, the authors would probably use the **tidyverse** package as reference.

At first, the main inconvenience of `puzzle()` may be the requirement of specific pre-defined formatting for its inputs. Nevertheless, it is indeed this feature that drastically increases the flexibility and productivity of `puzzle()`, allowing the assembling of all types of pharmacometrics datasets. The examples provided herein only scratches the surface of what the package is able to do. The reader is referred to the supplementary material for additional examples on how assembling pharmacometrics datasets using the **puzzle** package.

The data assembling workflow starts with data collection. The second step is data storage. Unfortunately, most of the heterogeneity and difficulty during the data assembling arises in this step. Effectively, each company or hospital may use its own recipe to store the data. Even more complexity is usually added to the process if, within a company or hospital, different scientists or nurses are involved in the data collection and/or storage. This is due to inter-individual (*i.e.* "two persons may label the same item differently") and/or inter-occasion (*i.e.* "the same person may label the same item differently") variability. This fact makes the principle of reproducibility from a data preparation point of view quite challenging.

The use of Study Data Tabulation Model (SDTM) has provided a standard for organizing and formatting data to streamline processes in collection, management, analysis and reporting, and it is one of the required standards for data submission to FDA and PMDA. The submission data standards team of Clinical Data Interchange Standards Consortium (CDISC) defines SDTM (Clinical Data Interchange Standards Consortium documentation). On July 21, 2004, SDTM was selected as the standard specification for submitting tabulation data to the FDA for clinical trials and on July 5, 2011 for nonclinical studies.

Unfortunately, inconsistencies in the data are not uncommon even if strict rules are supposed to be followed. Because of the heterogeneity of the data being assembled, the main limitation of the **puzzle** package is the requirement of pre-data manipulation. Specifically, the inconsistencies from SDTM have to be cleared before its use. Thus, at this stage of the development, the **puzzle** package is not fully compatible with SDTM datasets. Nevertheless, it is planned to implement this feature in the next version of the package. Alternatively, maybe, it is time within the pharmacometrics community to go one step further and develop new tools and methods allowing the establishment of strategies and/or protocols where data are more homogeneously stored. If this objective is accomplished and the degree of pre-data manipulation is reduced or even completely eliminated, the often-painful process of data assembling may become a breeze.

Conclusion

The **puzzle** package is a very flexible, powerful and efficient tool that helps modelers, programmers and/or pharmacometricians through the complex model building process. Specifically, it is the first open source tool supporting pharmacometricians during the difficult, error prone, and time consuming process of data assembling. Therefore, the **puzzle** package fills an unmet condition within the pharmacometrics workflow as it offers an opportunity for a unification of the code for data assembly improving reproducibility and traceability. Thus, we do believe that the **puzzle** package will be of high interest for the pharmacometrics community. It is the authors hope that this manuscript serves as detonating to set the foundations for a more efficient and pleasant data assembling in our field.

Bibliography

K. T. Baron. The mrgsolve package. `https://mrgsolve.github.io`, 2019. Accessed 2019-07-08. [p163]

S. Beal, L. Sheiner, A. Boeckmann, and R. Bauer. Nonmem user's guide. (1989-2009), 2009. Icon Development Solutions, Ellicott City, MD, USA. [p164]

S.-Y. Chen, Q. Liu, and Z. Feng. Common data manipulations with r in biological researches. *Journal of thoracic disease*, 9(7):2209–2213, 2017. URL https://doi.org/10.21037/jtd.2017.06.48. [p168]

Clinical Data Interchange Standards Consortium documentation. https://www.cdisc.org/standards/foundational/sdtm. Accessed 2019-11-29. [p170]

E. Comets, A. Lavenu, and M. Lavielle. saemix: Stochastic Approximation Expectation Maximization (SAEM) algorithm. https://rdrr.io/cran/saemix/man/saemix.html, 2011. Accessed 2019-11-29. [p164]

T. Dasu and T. Johnson. *Exploratory Data Mining and Data Cleaning*. Wiley-IEEE, 2003. [p163]

M. Fidler, J. J. Wilkins, R. Hooijmaijers, T. M. Post, R. Schoemaker, M. N. Trame, Y. Xiong, and W. Wang. Nonlinear mixed-effects model development and simulation using nlmixr and related r open-source packages. *CPT: pharmacometrics & systems pharmacology*, 8:621–633, 2019. URL https://doi.org/10.1002/psp4.12445. [p164]

T. H. Grasela, J. Fiedler-Kelly, B. Cirincione, D. Hitchcock, K. Reitz, S. Sardella, and B. Smith. Informatics: the fuel for pharmacometric analysis. *The AAPS journal*, 9(1):E84–E91, 2007. URL https://doi.org/10.1208/aapsj0901008. [p163]

E. N. Jonsson and M. O. Karlsson. Xpose–an S-PLUS based population pharmacokinetic/pharmacodynamic model building aid for nonmem. *Computer methods and programs in biomedicine*, 58(1):51–64, 1999. [p163]

R. J. Keizer. R library for simulation of PKPD models defined as ODE systems. The PKPDsim package. https://github.com/ronkeizer/PKPDsim, 2015. Accessed 2019-07-08. [p163]

R. J. Keizer. R library to create visual predictive checks. The vpc package. https://github.com/ronkeizer/vpc, 2018. Accessed 2019-07-08. [p163]

R. J. Keizer, M. van Benten, J. H. Beijnen, J. H. M. Schellens, and A. D. R. Huitema. Piraña and pcluster: a modeling environment and cluster infrastructure for nonmem. *Computer methods and programs in biomedicine*, 101(1):72–79, 2011. URL https://doi.org/10.1016/j.cmpb.2010.04.018. [p163]

L. Lindbom, P. Pihlgren, E. N. Jonsson, and N. Jonsson. Psn-toolkit–a collection of computer intensive statistical methods for non-linear mixed effect modeling using nonmem. *Computer methods and programs in biomedicine*, 79(3):241–257, 2005. URL https://doi.org/10.1016/j.cmpb.2005.04.005. [p163]

Lixoft. MONOLIX 2019 version documentation. http://monolix.lixoft.com/. Accessed 2019-10-28. [p164]

S. Mouksassi. Rshiny app as interface to ggplot2. The ggplotwithyourdata package. https://github.com/smouksassi/ggplotwithyourdata, 2016. Accessed 2019-07-08. [p163]

S. Mouksassi. Ggplot and summary statistics quick exploration of data. The ggquickeda package. https://github.com/smouksassi/ggquickeda, 2019. Accessed: 2019-07-08. [p163]

J. Owen and J. Fiedler-Kelly. *Introduction to population pharmacokinetic and pharmacodynamic analysis with nonlinear mixed effects models*. Wiley, 2014. [p167]

RStudio team. Rmarkdown: dynamic documents for R. The rmardown package. https://rmarkdown.rstudio.com/. Accessed 2019-10-28. [p163]

W. Wang, K. M. Hallow, and D. A. James. A Tutorial on RxODE: Simulating Differential Equation Pharmacometric Models in R. *CPT: pharmacometrics & systems pharmacology*, 5(1):3–10, 2016. URL https://doi.org/110.1002/psp4.12052. [p163]

H. Wickham. The tidyverse package. https://www.tidyverse.org/, Nov. 2017. Accessed 2019-07-08. [p168]

H. Wickham and G. Grolemund. *R for Data Science: Import, Tidy, Transform, Visualize, and Model Data*. O'Really Media, 2016. [p168]

H. Wickham et al. Tidy data. *Journal of Statistical Software*, 59(10):1–23, 2014. [p163]

Y. Xie, J. J. Allaire, and G. Grolemund. *R Markdown: The Definitive Guide*. Chapman & Hall/CRC, 2018. [p163]

*Mario González Sales**
Modeling Great Solutions
Crta. Engolasters s/n, Escaldes-Engordany, AD700
Andorra
mario@modelinggreatsolutions.com

*Olivier Barrière**
Certera
2000 Peel Street, Suite 570, Montréal, H3A 2W5
Canada
olivier.barriere@certara.com

*Both authors share the first authorship.

Pierre Olivier Tremblay
Syneos Health
2500 Rue Einstein, Québec, G1P 0A2
Canada
pierreolivier.tremblay@syneoshealth.com

Guillaume Bonnefois
Syneos Health
5160, boulevard Décarie, Montréal, H3X 2H9
Canada
guillaume.bonnefois@syneoshealth.com

Julie Desrochers
Syneos Health, Québec, Canada
2500, Rue Einstein, Québec, G1P 0A2
Canada
julie.desrochers@syneoshealth.com

Fahima Nekka
Université de Montréal - Faculté de Pharmacie
2940, Chemin de Polytechnique, Montréal, H3T 1J4
Canada
fahima.nekka@umontreal.ca

RNGforGPD: An R Package for Generation of Univariate and Multivariate Generalized Poisson Data

by Hesen Li, Hakan Demirtas, and Ruizhe Chen

Abstract This article describes the R package **RNGforGPD**, which is designed for the generation of univariate and multivariate generalized Poisson data. Some illustrative examples are given, the utility and functionality of the package are demonstrated; and its performance is assessed via simulations that are devised around both artificial and real data.

Introduction and motivation

It is well known that the variance of a Poisson variable equals to its mean. However, over- and under-dispersion in count data could make this mean-variance equality assumption unrealistically simplistic. This situation is often caused by the heterogeneity in the population, while we implicitly assume that the weights assigned to each event are equal when we employ the regular Poisson distribution. This problem can be addressed by modeling count data using the generalized Poisson distribution (GPD), which enables us to assign varying weights to events (Satterthwaite, 1942). The GPD includes a dispersion parameter λ, which accommodates over- or under-dispersion relative to the Poisson distribution in addition to the rate parameter θ in the regular Poisson distribution.

Scientific background

As discussed in the book of Consul (1989), the GPD has two parameters, rate and dispersion. It can be regarded as a mixture of Poisson distributions according to Joe and Zhu (2005).

The GPD can be described mathematically as follows: Let X be a discrete random variable defined over non-negative integers, and let $P_x(\theta, \lambda)$ be its probability mass function (pmf). X is said to follow the GPD with rate parameter θ and dispersion parameter λ if

$$P_x(\theta,\lambda) = \begin{cases} \theta\,(\theta+\lambda x)^{x-1}\, e^{-\theta-\lambda x}/x!, & \text{for } x = 0,1,2...\\ 0, & \text{for } x > m \text{ if } \lambda < 0 \end{cases}$$

and zero otherwise, where $\theta > 0$, $\max(-1, -\theta/m) \leq \lambda < 1$, and $m(\geq 4)$ is the largest positive integer for which $\theta + m\lambda > 0$ when $\lambda < 0$. The parameters θ and λ are independent, but the lower limits on λ and m are imposed to ensure that there are at least five classes with non-zero probability when λ is negative. $\lambda = 0$ corresponds to the Poisson distribution, while $\lambda > 0$ and $\lambda < 0$ correspond to over- and under-dispersion relative to the regular Poisson, respectively.

Besides, if we regard the weights of each event in a time period as independent and formulate the summation of the weights as a characteristic function, the distribution function of the sum of weights has all the properties of the GPD after the application of the Fourier transformation (Satterthwaite, 1942).

According to Vernic (2000), the pmf for the multivariate GPD can be derived using the multivariate reduction method. In her derivation, an m-dimensional GPD (MGP) is obtained by taking $(m+1)$ independent univariate generalized Poisson random variables, $X_i \sim \text{GPD}(\theta_i, \lambda_i)$, for $i = 0, ..., m$, and let $Y_1 = X_1 + X_0, Y_2 = X_2 + X_0, ..., Y_m = X_m + X_0$, Then $(Y_1, ..., Y_m) \sim \text{MGP}(\Theta, \Lambda)$, where $\Theta = (\theta_0, ..., \theta_m)$ and $\Lambda = (\lambda_1, ..., \lambda_m)$. The joint pmf of $(Y_1, ..., Y_m)$ is

$$\begin{aligned} P(y_1, ..., y_m) &= P(Y_1 = y_1, ..., Y_m = y_m) \\ &= \sum_{k=0}^{\min(y_1,...,y_m)} p_1(y_1-k) \cdot ... \cdot p_m(y_m-k)\, p_0(k), \end{aligned}$$

where p_i is the pmf of the random variable X_i. Plugging in the pmf of X_i's, we get

$$P(y_1,...,y_m) = \left(\prod_{j=0}^{m}\theta_j\right)\exp\left\{-\theta-\sum_{j=1}^{m}y_j\lambda_j\right\}\cdot\sum_{k=0}^{\min(y_1,...,y_m)}\left(\prod_{j=1}^{m}\frac{[\theta_j+(y_j-k)\lambda_j]^{y_j-k-1}}{(y_j-k)!}\right)\cdot$$
$$\cdot\frac{(\theta_0+k\lambda_0)^{k-1}}{k!}\exp\left\{k\left(\sum_{j=1}^{m}\lambda_j-\lambda_0\right)\right\},$$

where $\theta = \sum_{j=0}^{m}\theta_j$ and $0! = 1$.

Application fields

The applications of the GPD in science and business vary in a wide range that spans life insurance, physics, genetic biology, and public health. Satterthwaite (1942) mentioned a case where insurance companies model the average financial cost per claim with the GPD, which allows the weights (costs of different claims) to be heterogeneous. Vernic (1997) modeled the joint distribution of the yearly frequencies of hurricanes affecting the first and the third zones of the north Atlantic coastal states in the USA as a bivariate GPD.

Consul and Famoye (2006) gave an example regarding the induction and restitution process of chromosomes. Chromosomes can be damaged in the induction process, and repaired in the restitution process. The dispersion parameter λ in the GPD represents an equilibrium constant which is the limit of the ratio of the rate of induction to the rate of restitution, and thus the GPD can be used to estimate the net free energy for the production of induced chromosome aberrations (damaged chromosomes).

Although the importance of generating the multivariate generalized Poisson data is evident, there has not been a comprehensive computational tool specifically targeted for this particular distribution. Demirtas (2017) compared a few random variate generation techniques for univariate GPD, and mentioned the potential for the generation of multivariate GPD variates via correlation mapping procedure in a similar fashion to the method of Yahav and Shmueli (2012), which is concerned with correlated regular Poisson data generation. The R package **RNGforGPD** (Li et al., 2020) is developed to provide the accommodating tools for the expanded versions of the methods that appear in Demirtas (2017) and Yahav and Shmueli (2012), where the augmentation is mostly about allowing over- and under-dispersion for count data in a correlated setting.

The rest of the article is organized as follows: In Section 2, we outline the algorithms for the generation of univariate and multivariate generalized Poisson data. In Section 3, we describe the technical details of the R package **RNGforGPD**. In Section 4, we present the results of simulation studies that are designed around both artificial and real data. Finally, we conclude the paper with a brief discussion in Section 5.

Algorithm

First, we describe the prerequisites for the generation of GPD data. Then, we discuss the five algorithms for generating univariate GPD data, and the algorithm for generating multivariate GPD data, which is based on an adaptation of Yahav and Shmueli (2012)'s method for generating multivariate regular Poisson data. In addition, we provide guidance on how to choose an appropriate data generation function for generating univariate GPD data at the end of this section.

Generating univariate GPD data

In general, an appropriate choice from the five algorithms in generating univariate GPD data depends on the values of the rate (θ) and dispersion (λ) parameters. Descriptions of each of the five algorithms (Demirtas, 2017) are given in Table 1:

Table 1: Table of algorithms.

Algorithms	Steps	Notes
Inversion	1. Set $\omega = e^{-\lambda}, X = 0, S = e^{-\theta}$ and $P = S$ 2. Generate $U \sim U(0,1)$ 3. While $U > S$, do (a) $X = X + 1$ (b) $C = \theta - \lambda + \lambda X$ (c) $P = \omega C(1 + \lambda/C)^{X-1} P/X$ (d) $S = S + P$ 4. Deliver X	This algorithm is a general purpose univariate random number generation method that depends on the recursive relationship between consecutive GPD probabilities: $P_x(\theta,\lambda) = \frac{\theta-\lambda+\lambda x}{x} \times \left(1 + \frac{\lambda}{\theta-\lambda+\lambda x}\right)^{x-1} \times e^{-\lambda} P_{x-1}(\theta,\lambda),$ for $x \geq 1$ with $P_0(\theta,\lambda) = e^{-\lambda}$.
Branching	1. Generate $Y \sim \text{Pois}(\theta)$ 2. Set $X = Y$, if $X = 0$, deliver X 3. Generate $Z \sim \text{Pois}(\lambda Y)$ 4. Set $X = X + Z$ and $Y = Z$, if $Y = 0$, deliver X, otherwise go to the previous step	This algorithm is a distribution specific algorithm and it only works for positive λ values. Consul and Shoukri (1988) showed that when $X_0 \sim \text{Pois}(\theta)$ and $X_j \sim \text{Pois}(\lambda)$, where $j = 1, 2, ...n, Y = \sum_{k=0}^{n} X_k$ follows the GPD with rate parameter θ and dispersion parameter λ.
Normal Approximation	1. Initialize $m = \theta(1-\lambda)^{-1}$ and $\nu = \sqrt{\theta(1-\lambda)^{-3}}$ 2. Generate Y from a standard normal distribution 3. $X = \max(0, \lfloor m + \nu Y + 0.5 \rfloor)$, where $\lfloor . \rfloor$ is the floor function 4. Deliver X	This algorithm uses the first two moments and a continuity correction in generating univariate GPD data.
Build-Up	1. Set $t = e^{-\theta}, X = 0, P_x = t$ and $S = P_x$ 2. Generate $U \sim U(0,1)$ 3. If $U \leq S$ then deliver X, otherwise set $X = X + 1$, compute P_x by the probability mass function, set $S = S + P_x$, and return to the previous step	The cumulative distribution function (cdf) is built up by the recursive computation of the mass probabilities (Kemp, 1981).
Chop-Down	1. Set $t = e^{-\theta}, X = 0$ and $P_x = t$ 2. Generate $U \sim U(0,1)$ 3. If $U \leq P_X$ then deliver X, otherwise set $U = U - P_x, X = X + 1$, and compute P_x by the probability mass function, and return to the previous step	The generated uniform variate is decreased by an amount equal to the cdf (Kemp, 1981).

Generating multivariate GPD data

Yahav and Shmueli (2012) developed an algorithm for generating multivariate Poisson data using an improved version of the NORTA method (NORmal To Anything). The NORTA method can be used for generating multivariate regular Poisson data by simulating a p-dimensional multivariate Normal distribution with a correlation structure R_N, and then transform it into a regular Poisson distribution using the inverse cumulative distribution function (Chen, 2001). However, Yahav and Shmueli (2012) realized a drawback of the NORTA method for generating multivariate Poisson variates. For lower values of rates (θ), the desired correlation matrix (ρ_{Pois}) deviates seriously from the normal approximating correlation matrix (ρ_N) under the NORTA method, and this problem still persists in generating multivariate GPD data. However, the bias can be approximately corrected through an exponential function:

$$\rho_{Pois} = a \times e^{b\rho_N} + c,$$

where

$$a = -\frac{\bar{\rho} \times \underline{\rho}}{\rho + \underline{\rho}},\ b = \log\left(\frac{\bar{\rho} + a}{a}\right),\ c = -a,$$

and $\bar{\rho}$ and $\underline{\rho}$ represent the upper and lower bounds of the pair correlation, which can be calculated using their defined equations, respectively:

$$\underline{\rho} = corr\left(\Xi_{\lambda_i}^{-1}(U), \Xi_{\lambda_j}^{-1}(1-U)\right),$$
$$\bar{\rho} = corr\left(\Xi_{\lambda_i}^{-1}(U), \Xi_{\lambda_j}^{-1}(U)\right).$$

In our package, we keep the corrections of correlation matrix and adapt the calculations of $\bar{\rho}$ and $\underline{\rho}$ using a simple but accurate sorting technique, which is described in Demirtas and Hedeker (2011). We adapt Yahav and Shmueli (2012)'s method and develop the algorithm for generating multivariate generalized Poisson data. Suppose we want to generate a p-dimensional generalized Poisson data with an arbitrary correlation matrix R_{Gpois} (which we refer to as the target correlation matrix), rate parameter vector $\overrightarrow{\Theta} = \{\theta_1, \theta_2, ..., \theta_p\}$ and dispersion parameter vector $\overrightarrow{\Lambda} = \{\lambda_1, \lambda_2, ..., \lambda_p\}$:

(1) Compute the intermediate correlation matrix R_N from the target correlation matrix using Equation 2.2.2.

(2) Generate a p-dimensional normal vector $\overrightarrow{X_N}$ with mean $\overrightarrow{\mu} = 0$, variance $\overrightarrow{\sigma^2} = 1$, and a correlation matrix R_N.

(3) For each value X_{N_i}, $i \in 1, 2, ..., p$, calculate the normal cdf:

$$\Phi\left(X_{N_i}\right).$$

(4) For each $\Phi(X_{N_i})$, calculate the Poisson inverse cdf (quantile) with rate parameter vector $\overrightarrow{\Theta}$ and dispersion parameter vector $\overrightarrow{\Lambda}$

$$X_{Gpois_i} = \Xi^{-1}\left(\Phi\left(X_{N_i}\right)\right),$$

where

$$\Phi(x) = \int_{-\infty}^{x} \frac{1}{\sqrt{2\sigma^2}} e^{\frac{-u^2}{2}} du,$$
$$\Xi(x) = \sum_{i=0}^{x} \theta\left(\theta + \lambda i\right)^{i-1} e^{-\theta-\lambda i}/i!.$$

The resulting vector $\overrightarrow{X}_{Gpois}$ is a p-dimensional generalized Poisson vector with correlation matrix R_{Gpois}, rate vector $\overrightarrow{\Theta}$ and dispersion vector $\overrightarrow{\Lambda}$.

Comparisons of five univariate methods

In designing our package, we give our users the freedom to choose their preferred methods for generating univariate GPD data. However, users who are not familiar with the mechanisms of generating univariate GPD might not know how to choose the best method for their simulation scenarios. Demirtas (2017) evaluated the relative advantages and disadvantages of the five methods for generating univariate GPD data in terms of unbiasedness, variability, and speed in simulation studies. We find the results he found are instructive on choosing the appropriate methods for package users, so we summarize his findings as follows:

- When the rate parameter θ is large, *Inversion* method and *Branching* method have the best accuracy.
- When the dispersion parameter λ is large, *Inversion* method and *Branching* method have better accuracy, while *Build-Up* method and *Chop-Down* method have better precision.
- When the population mean is large, *Normal Approximation* method has better precision.
- When the population variance is large, *Inversion* method and *Branching* method have better accuracy, *Build-Up* method, *Chop-Down* method and *Normal Approximation* method have better precision.
- When the population skewness is large, *Inversion* method and *Branching* method have better accuracy, *Build-Up* method and *Chop-Down* method have better precision.

The RNGforGPD package

The **RNGforGPD** package provides functions for generating univariate and multivariate data that follow the generalized Poisson distribution. The package is available via the Comprehensive R Archive Network (CRAN) at `https://CRAN.R-project.org/package=RNGforGPD`. Once the package has been appropriately installed on a local machine, the results presented in this paper can be reproduced. The R code used in this manuscript can be accessed at `https://demirtas.people.uic.edu/RNGforGPD_paper_LDC_R_Journal.R`.

This package includes two data generating functions: `GenUniGpois` and `GenMVGpois`. The data generating functions are supported by five core functions: `CmatStarGpois`, `ComputeCorrGpois`, `CorrNNGpois`, `QuantileGpois`, and `ValidCorrGpois`, that provide essential support to the two data generating functions. Throughout this article, we use the following input arguments as given in Table 2:

Table 2: Table of input arguments.

Input Argument	Description
`sample.size`	Number of rows to be generated in multivariate generalized Poisson data.
`no.gpois`	Dimension of the multivariate generalized Poisson distribution.
`n`	Number of data points to be generated in univariate generalized Poisson data.
`p`	Percentile of the generalized Poisson distribution.
`cmat.star`	Intermediate correlation matrix to be used in generating multivariate generalized Poisson data.
`corMat`	A positive definite target correlation matrix whose entries are within the valid limits.
`theta.vec`	A vector of rate parameters in the multivariate generalized Poisson distribution.
`lambda.vec`	A vector of dispersion parameters in the multivariate generalized Poisson distribution.
`theta`	Rate parameter in the univariate generalized Poisson distribution.
`lambda`	Dispersion parameter in the univariate generalized Poisson distribution.
`method`	Method to be used in generating univariate generalized Poisson data.
`details`	Boolean parameter for users to decide whether to display the specified and empirical values of parameters.
`verbose`	Boolean parameter for users to decide whether to display traces or not.

Table 3 summarizes each function in the **RNGforGPD** package:

Table 3: Table of functions.

Function type	**Function name**	**Description**
Data generating functions	`GenUniGpois`	Generates univariate GPD variables
	`GenMVGpois`	Generates univariate GPD variables
Core functions	`CmatStarGpois`	Computes the intermediate correlation matrix
	`ComputeCorrGpois`	Computes correlation bounds
	`CorrNNGpois`	Adjusts the target correlation
	`QuantileGpois`	Computes the quantile for GPD
	`ValidCorrGpois`	Validates the correlation matrix

Their functionality, in the context of generalized Poisson data generation, is described in the next several subsections:

`GenUniGpois`

`GenUniGpois` generates univariate data that follow the GPD with pre-specified rate and dispersion parameters using appropriate methods according to different values of θ and λ as described in the previous section. It takes `theta`, `lambda`, `n`, `details`, and `method` as input arguments. A warning will be displayed if the method chosen by user is inappropriate considering the characteristics of each method. For example, the *Normal Approximation* method does not work well for $\theta < 10$. In that case, a warning message shows up suggesting the user to choose a working method according to their specific θ and λ parameters. Also, *Branching* method only works for positive λ values.

`GenMVGpois`

`GenMVGpois`, also referred to as the "engine" function in our package, generates multivariate GPD data. It generates multivariate data that follow the GPD with pre-specified rate parameter vector, dispersion parameter vector, and an intermediate correlation matrix. Its functionality depends on all the other functions in the package (except for `GenUniGpois`). Besides, it requires the `rmvnorm` function from the **mvtnorm** (Genz et al., 2020) package, the `is.positive.definite` function from the **corpcor** (Schafer et al., 2017) package, and the `nearPD` function from the **Matrix** (Bates and Maechler, 2019) package. It takes `sample.size`, `no.gpois`, `cmat.star`, `theta.vec`, `lambda.vec`, and `details` as input arguments. The `cmat.star` argument is the intermediate correlation matrix, and is later used to obtain the target correlation matrix using the inverse cdf transformation method in `GenMVGpois`. This argument needs to be executed using the `CmatStarGpois` function before it can be used by the `GenMVGpois` function.

Generation of the multivariate GPD data is more complex than that of the univariate GPD data due to the restrictions on the correlation matrix. These requirements can also be verified by the core functions as explained below.

`CmatStarGpois` and `CorrNNGpois`

`CmatStarGpois` function computes an intermediate correlation matrix, that will be used to obtain the target correlation matrix, using the inverse cdf transformation method in `GenMVGpois`. Because the target correlation matrix has to be positive definite and its entries must be within the correlation bounds, therefore `CmatStarGpois` requires the functionality of both `ValidCorrGpois` and `CorrNNGpois`. `ValidCorrGpois` checks the validity of the values of pairwise correlations including positive definiteness, symmetry, and correctness of the dimensions. `CorrNNGpois` adjusts the realized correlation to the target correlation bounds.

The following example shows the use of `CorrNNGpois` for adjusting the realized correlation to the targeted correlation bounds, and `CmatStarGpois` for computing intermediate values of pairwise correlations between three GPD variates.

```
set.seed(3406)
CorrNNGpois(c(0.1, 10), c(0.1, 0.2), 0.5)
#> [1] 0.8016437
lambda.vec <- c(-0.2, 0.2, -0.3)
theta.vec <- c(1, 3, 4)
M <- c(0.352, 0.265, 0.342)
```

```
N <- diag(3)
N[lower.tri(N)] <- M
TV <- N + t(N)
diag(TV) <- 1
cstar <- CmatStarGpois(TV, theta.vec, lambda.vec, verbose = FALSE)
cstar
#>           [,1]      [,2]      [,3]
#> [1,] 1.0000000 0.3943785 0.2946171
#> [2,] 0.3943785 1.0000000 0.3601862
#> [3,] 0.2946171 0.3601862 1.0000000
```

If the intermediate correlation matrix is not positive definite, the nearest positive definite matrix will be used.

`QuantileGpois`

`QuantileGpois` function computes the quantile for the generalized Poisson distribution for specified values of percentile, θ and λ parameters. This function is of great importance because it realizes the NORTA method (Chen, 2001) by inversely transforming the normal cdf to GPD quantiles. The example below shows the use of `QuantileGpois` for computing the quantile of a generalized Poisson distribution given θ, λ, and the percentile of the variate.

```
QuantileGpois(0.98, 1, -0.2, details = TRUE)
#> x = 0, P(X = x) = 0.3678794, P(X <= x) = 0.3678794
#> x = 1, P(X = x) = 0.449329, P(X <= x) = 0.8172084
#> x = 2, P(X = x) = 0.1646435, P(X <= x) = 0.9818519
#> When lambda is negative, we need to account for truncation error
#> The adjusted CDF are: 0.3746792 0.8323133 1
#> [1] 2
```

The corresponding cdf are adjusted to account for truncation error when $\lambda < 0$ as the warning shows above. Besides, λ must be greater than or equal to $-\theta/4$ when $\lambda < 0$.

`ComputeCorrGpois` and `ValidCorrGpois`

Theoretically, correlation bounds (both Pearson and Spearman correlations) for pairwise random variables are between -1 and 1. However, correlation bounds in practice are often narrower than their theoretical limits due to the restrictions imposed by the marginal distributions. Given vectors of θ and λ values, `ComputeCorrGpois` computes the pairwise correlation bounds between any pair of generalized Poisson variables using the Generate, Sort, and Correlate (GSC) algorithm described in Demirtas and Hedeker (2011). It is also an integral part of `ValidCorrGpois` function that examines whether values of pairwise correlation matrix fall within the limits imposed by the marginal distributions. Besides, `ValidCorrGpois` checks positive definiteness, symmetry, correctness of the dimensions of the input correlation matrix. The following example demonstrates the use of both functions:

```
set.seed(86634)
ComputeCorrGpois(c(3, 2, 5,4), c(0.3, 0.2, 0.5, 0.6), verbose = FALSE)
#> $min
#>            [,1]         [,2]        [,3]        [,4]
#> [1,]         NA   -0.8441959  -0.8523301  -0.8040863
#> [2,] -0.8441959           NA  -0.8364747  -0.7861681
#> [3,] -0.8523301   -0.8364747          NA  -0.7966635
#> [4,] -0.8040863   -0.7861681  -0.7966635          NA

#> $max
#>           [,1]      [,2]      [,3]      [,4]
#> [1,]        NA 0.9838969 0.9937024 0.9869316
#> [2,] 0.9838969        NA 0.9872343 0.9819031
#> [3,] 0.9937024 0.9872343        NA 0.9941324
#> [4,] 0.9869316 0.9819031 0.9941324        NA

ValidCorrGpois(matrix(c(1, 0.9, 0.9, 1), byrow = TRUE, nrow = 2),
               c(0.5, 0.5), c(0.1, 0.105), verbose = FALSE)
#>[1] TRUE
```

The following diagram shows the dependencies of the functions in the **RNGforGPD** package, the arrows suggest that the function on the tail of each arrow depends on the function on its head:

$$\texttt{GenMVGpois} \longrightarrow \begin{cases} \texttt{GenUniGpois} \\ \texttt{ValidCorrGpois} \longrightarrow \begin{cases}\texttt{GenUniGpois}\\ \texttt{ComputeCorrGpois} \longrightarrow \texttt{GenUniGpois} \\ \texttt{is.positive.definite*}\end{cases} \\ \texttt{CorrNNGpois} \longrightarrow \texttt{GenUniGpois} \\ \texttt{CmatStarGpois} \longrightarrow \begin{cases} \texttt{ValidCorrGpois}\\ \texttt{CorrNNGpois}\\ \texttt{is.positive.definite*}\\ \texttt{nearPD*}\end{cases} \\ \texttt{QuantileGpois} \\ \texttt{rmvnorm*}\end{cases}$$

* *indicates that the function is from another R package.*

Simulation studies

In this section, we present three examples that hinge upon one artificial and two real data-based scenarios. We demonstrate the functionality of the package through simulating GPD data based on the parameter estimates (rate parameter θ and dispersion parameter λ) and compare the simulated empirical estimates with the specified parameters,

$$\theta = \sqrt{\frac{\mu^3}{\sigma^2}},\ \lambda = 1 - \frac{\sqrt{\mu}}{\sigma}. \tag{1}$$

The most intuitive way to check the legitimacy of the simulation results is to compare the estimated rate and dispersion parameters to the specified quantities. Moreover, to further verify the simulation results for multivariate GPD, we calculate the first four moments via simulated data and compare them with the theoretical (specified) moments since most real life distributions are typically characterized by their first four moments. The expressions for the mean (μ), variance (σ^2), skewness (ν_1) and excess kurtosis (ν_2) derived by Consul and Famoye (2006) are as follows:

$$\mu = \theta\,(1-\lambda)^{-1},\ \sigma^2 = \theta\,(1-\lambda)^{-3},\ \nu_1 = \frac{1+2\lambda}{(\theta\,(1-\lambda))^{1/2}},\ \nu_2 = \frac{1+8\lambda+6\lambda^2}{\theta\,(1-\lambda)}. \tag{2}$$

As can be seen from the variance expression, $\lambda = 0$ corresponds to the standard Poisson distribution, $\lambda > 0$ and $\lambda < 0$ signify over- and under-dispersed count data relative to the Poisson, respectively.

Artificial data modeled via multivariate GPD

In this example, we generate a four-dimensional Poisson data of size 2,000 based on 1,000 replications.

One of its marginal random variables is distributed as a regular Poisson distribution, and the other three follow the GPD with different rate and dispersion parameters. The specifications on the rate and dispersion parameters of the four Poisson distributions are listed below:

- Variable 1. Ordinary count data (regular Poisson data): mean 2.00, variance 2.00 with rate parameter 2
- Variable 2. Over-dispersed count data (GPD): mean 5.00, variance 13.89 with rate parameter 3, dispersion parameter 0.4
- Variable 3. Over-dispersed count data (GPD): mean 10.00, variance 40.00 with rate parameter 5, dispersion parameter 0.5
- Variable 4. Under-dispersed count data (GPD): mean 44.00, variance 28.16 with rate parameter 55, dispersion parameter -0.25

As we are generating multivariate GPD data, we need to specify a positive definite correlation matrix whose entries are within the feasible lower and upper bounds. The specified correlations and the empirical results that are obtained through averaging the correlation matrix across 1,000 replications are shown below:

Table 4: Artificial data: specified and empirical correlation matrices.

Specified	$Y1$	$Y2$	$Y3$	$Y4$
$Y1$	1.0000	0.1521	0.2652	0.2428
$Y2$	0.1521	1.0000	-0.6475	0.1645
$Y3$	0.2652	-0.6475	1.0000	-0.2522
$Y4$	0.2428	0.1645	-0.2522	1.0000
Empirical	$Y1$	$Y2$	$Y3$	$Y4$
$Y1$	1.0000	0.1520	0.2676	0.2445
$Y2$	0.1520	1.0000	-0.6499	0.1654
$Y3$	0.2676	-0.6499	1.0000	-0.2517
$Y4$	0.2445	0.1654	-0.2517	1.0000

Below is the table of empirical θ's and λ's for four marginals compared to the specified θ's and λ's across 1,000 replications:

Table 5: Specified and empirical θ's and λ's for four marginals.

Parameter	Comparison	Variable 1	Variable 2	Variable 3	Variable 4
Rate (θ)	Specified	2.0000	3.0000	5.0000	55.0000
	Empirical	1.9994	3.0041	4.9930	55.1155
Dispersion (λ)	Specified	0.0000	0.4000	0.5000	-0.2500
	Empirical	0.0004	0.3996	0.5004	-0.2527

We can see that the empirical θ's, λ's and correlation matrix of the data generated using the `GenMVGpois` function are very close to the specified true parameters. The table below compares their first four moments:

Table 6: Artificial data: specified and empirical moments.

Moments	Comparison	Variable 1	Variable 2	Variable 3	Variable 4
Mean(μ)	Specified	2.0000	5.0000	10.0000	44.0000
	Empirical	2.0001	5.0036	9.9949	43.9975
Variance(σ^2)	Specified	2.0000	13.8889	40.0000	28.1600
	Empirical	2.0016	13.8809	40.0505	28.0374
Skewness(ν_1)	Specified	0.7071	1.3416	1.2649	0.0603
	Empirical	0.7079	1.3397	1.2669	0.0595
Kurtosis(ν_2)	Specified	0.5000	2.8667	2.6000	-0.0091
	Empirical	0.5018	2.8581	2.6085	-0.0092

Tables 6 and 7 show that the empirical first four moments of the generated GPD data are very close to the specified ones, indicating the algorithm of generating the data properly captures the true parameter values with negligible deviations.

Table 7: Artificial data: per cent difference between empirical and specified moments.

Moments	Variable 1	Variable 2	Variable 3	Variable 4
Mean (μ)	0.01%	0.07%	0.05%	0.01%
Variance (σ^2)	0.08%	0.06%	0.13%	0.44%
Skewness (ν_1)	0.11%	0.14%	0.16%	1.29%
Kurtosis (ν_2)	0.36%	0.30%	0.33%	1.72%

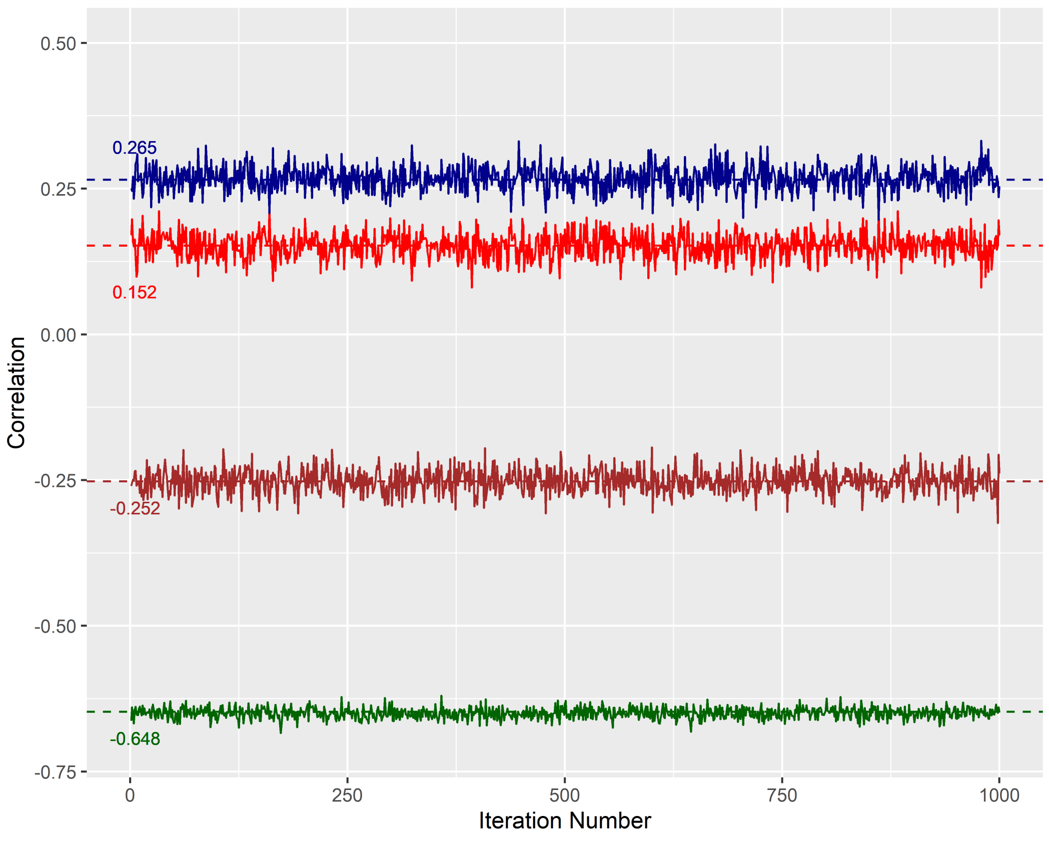


Figure 1: Artificial data example: Empirical values versus specified correlations across 1,000 replications (four out of six unique pairwise correlations). Trace plot of empirical correlations that appear to closely approximate the specified correlations across 1,000 replications.

To further illustrate the precision of the algorithm, we use the entries (1, 2), (1, 3), (2, 3), (3, 4) of the empirical correlation matrices to generate the plot as shown in Figure 1 using the R package **ggplot2** (Wickham, 2016). The dashed lines represent the specified correlation values, the figure shows that the empirical correlation values across 1,000 iterations fluctuate around the specified ones within reasonably small ranges.

Epilepsy rates modeled via univariate GPD

The epilepsy data (Thall and Vail, 1990) available from the R package **robustbase** (Maechler et al., 2020) were collected from a randomized clinical trial investigating the treatment effect of an anti-epileptic drug called Progabide, which was originally conducted by Leppik (1985). In this clinical trial study, 59 patients suffering from simple or complex partial seizures were randomized to groups receiving either the anti-epileptic drug Progabide or a placebo in addition to standard chemotherapy. The baseline number of seizures occurred was measured for each patient followed by four successive post-randomization clinic visits with the number of seizures occurring over the previous two weeks reported. Although all patients were crossed over to the other treatment, we are only interested in modeling the number of seizures at baseline and the four pre-crossover follow-up responses for each patient as a realization of GPD using our package. The number of seizures that occur on a patient follows a Poisson distribution by assuming that each patient is independent of each other, and we regard them as fixed unit "intervals". In this scenario, the GPD is more appropriate than the ordinary Poisson distribution in modeling the count data since the patients generally do not exhibit the characteristics of homogeneity in real life.

Univariate GPD simulation with small sample size

The data set has a relatively small sample size of 59, and we set the number of replications as 1,000. First, we simulate univariate GPD data based on the baseline seizure counts measured. The true rate (θ) and dispersion (λ) parameters used to generate univariate GPD data are calculated by the method of moments in which the functions of parameters are set to equal to the moment estimates of the data. The simulation results of the five univariate GPD generation algorithms are presented in the table below:

Table 8: Epilepsy data (baseline seizure counts): specified and empirical parameters using the original sample size of 59.

Parameters	Specified	Branching	Inversion	Build-Up	Chop-Down	Normal
Rate (θ)	6.4904	6.7701	6.7548	6.8478	6.8594	7.9017
Dispersion (λ)	0.7921	0.7817	0.7821	0.7773	0.7770	0.7597

The first column lists the true values of rate and dispersion parameters calculated by the method of moments. Overall, the empirical rate parameters overestimate the true rate parameters, and the empirical dispersion parameters underestimate the true dispersion parameters. We note that the *Normal Approximation* method performs badly, perhaps due to the limitation of its approximation to the Poisson distribution with a small rate parameter.

Estimation problems caused by simulating GPD data using a small sample size

We can infer from the above simulation results that $\hat{\theta}$ and $\hat{\lambda}$ calculated using each of the five univariate approaches overestimates and underestimates the true θ and true λ, respectively, under a small sample size scenario. This behavior of the estimators can be explained by a close examination of the Equation (1):

$$\theta = \sqrt{\frac{\mu^3}{\sigma^2}},\ \lambda = 1 - \frac{\sqrt{\mu}}{\sigma}.$$

Since under a small sample size, the sample variance $\hat{\sigma^2}$ underestimates the true variance σ^2. Assuming that $\hat{\mu}$ is a consistent estimator of μ, $\hat{\theta} = \sqrt{\frac{\hat{\mu}^3}{\hat{\sigma^2}}}$ overestimates the true θ, and $\hat{\lambda} = 1 - \frac{\sqrt{\hat{\mu}}}{\hat{\sigma}}$ underestimates the true λ.

Univariate GPD simulation with large sample size

To alleviate this over/under estimation problem and demonstrate the performance of our package, we use the true θ and λ parameters calculated from the epilepsy baseline seizure counts data to set up a new simulation scenario where the sample size is increased from 59 to 2,000 while maintaining the original distributional properties. The simulation results under this scenario are shown in the Table 9 below:

Table 9: Epilepsy data (baseline seizure counts): specified and empirical parameters using an augmented sample size of 2,000.

Parameters	Specified	Branching	Inversion	Build-Up	Chop-Down	Normal
Rate (θ)	6.4904	6.4973	6.4953	6.6879	6.7043	7.8181
Dispersion (λ)	0.7921	0.7919	0.7920	0.7845	0.7841	0.7619

We note that the *Build-up* and *Chop-Down* methods do not work as well as the *Branching* and *Inversion* methods, which capture the true rate and dispersion parameters of the data. The *Normal Approximation* method still does not perform well in the large sample scenario.

Physician visits modeled via multivariate GPD

Deb and Trivedi (1997) conducted research on 4,406 individuals, aged 66 and over, who are covered by Medicare (a public insurance program). The data are available as `DebTrivedi.rda` in the R package **MixAll** (Iovleff, 2019). For this data set, we consider the multivariate generation of two mutually exclusive measures of utilization variables, one pathologic variable, and one demographic variable: visits to a physician in an office setting (OFP), visits to a physician in a hospital outpatient setting (OPP) adjusted by adding 1 to avoid computational complexities, the number of chronic diseases and conditions (NUMCHRON), and the years of education received (SCHOOL). The simulation results based on 1,000 replications are shown below:

Table 10: DebTrivedi data: specified and empirical θ's and λ's for four marginals.

Parameter	Comparison	OFP	SCHOOL	OPP + 1	NUMCHRON
Rate (θ)	Specified	2.0529	8.8291	0.6342	1.4188
	Empirical	2.0545	8.8345	0.6357	1.4191
Dispersion (λ)	Specified	0.6445	0.1420	0.6378	0.0799
	Empirical	0.6442	0.1416	0.6373	0.0801

Table 11: DebTrivedi data: specified and empirical correlation matrices.

Specified	$Y1$	$Y2$	$Y3$	$Y4$
$Y1$	1.0000	0.0644	0.0681	0.2619
$Y2$	0.0644	1.0000	-0.0122	-0.0658
$Y3$	0.0681	-0.0122	1.0000	0.1008
$Y4$	0.2619	-0.0658	0.1008	1.0000
Empirical	$Y1$	$Y2$	$Y3$	$Y4$
$Y1$	1.0000	0.0646	0.0743	0.2688
$Y2$	0.0646	1.0000	-0.0120	-0.0660
$Y3$	0.0743	-0.0120	1.0000	0.1066
$Y4$	0.2688	-0.0660	0.1066	1.0000

Table 12: DebTrivedi data: specified and empirical moments.

Moments	Comparison	OFP	SCHOOL	OPP + 1	NUMCHRON
Mean(μ)	Specified	5.7744	10.2903	1.7508	1.5420
	Empirical	5.7745	10.2922	1.7528	1.5427
Variance(σ^2)	Specified	45.6871	13.9781	13.3426	1.8215
	Empirical	45.6190	13.9688	13.3244	1.8231
Skewness(ν_1)	Specified	2.6794	0.4665	4.7475	1.0152
	Empirical	2.6767	0.4660	4.7371	1.0155
Kurtosis(ν_2)	Specified	11.8495	0.2979	37.1841	1.2852
	Empirical	11.8255	0.2972	37.0193	1.2865

Table 13: DebTrivedi data: per cent difference between empirical and specified moments.

Moments	OFP	SCHOOL	OPP + 1	NUMCHRON
Mean (μ)	0.00%	0.02%	0.11%	0.05%
Variance (σ^2)	0.15%	0.07%	0.14%	0.09%
Skewness (ν_1)	0.10%	0.11%	0.22%	0.03%
Kurtosis (ν_2)	0.20%	0.26%	0.44%	0.09%

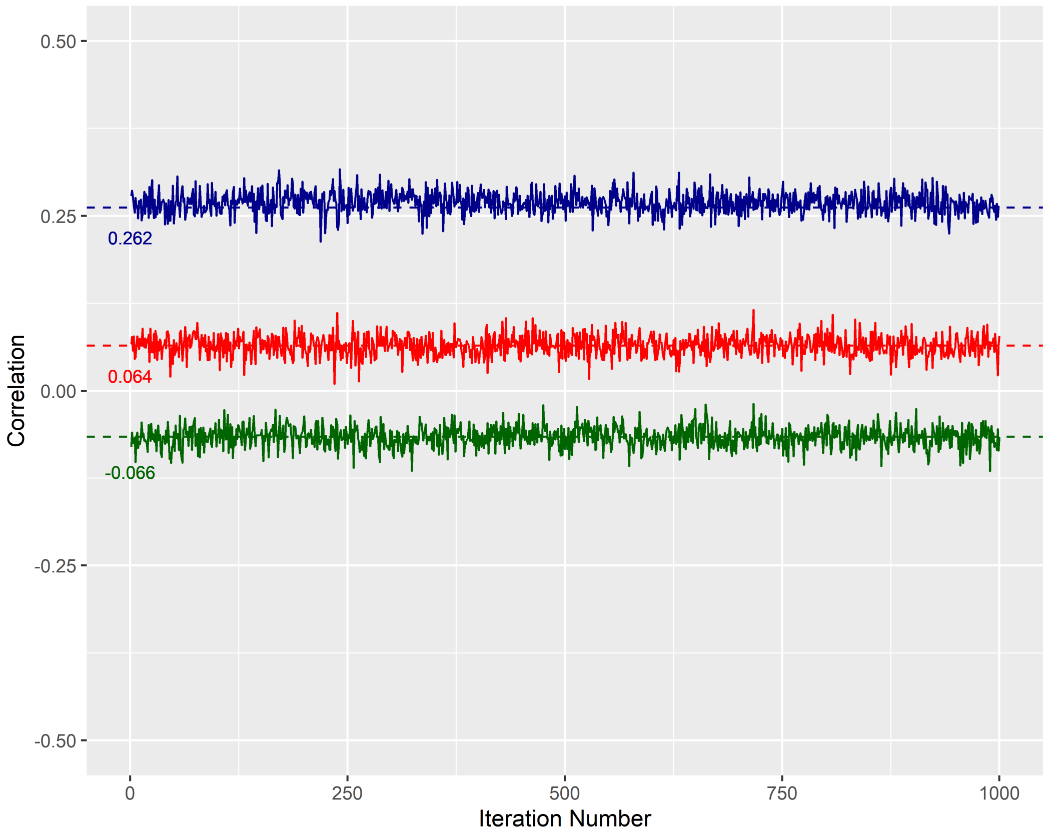


Figure 2: DebTrivedi data example: Empirical values versus specified correlations across 1,000 replications (three out of six unique pairwise correlations). Trace plot of empirical correlations that appear to closely approximate the specified correlations across 1,000 replications.

Graphical tools further verify the simulation results: In Figure 2, we see that a few randomly selected empirical correlations (entries (1, 2), (1, 4), (2, 4)) across 1,000 replications fluctuate around the specified quantities within reasonably small ranges, suggesting that the algorithm is consistent and efficient in capturing the desired values.

We evaluate the performance of the package through a comparison between the theoretical and empirical first four moments and parameters, as we reported earlier in the manuscript. As correctly indicated by a reviewer, an additional assessment that involves identifying the proximity between theoretical and empirical pmf's may provide further support for the software tool under consideration. In this spirit, we show two plots (Figure 3 and Figure 4) that visually validate the feasibility of our multivariate GPD generating algorithm. In Figure 3, we compare the theoretical and empirical marginal probabilities using each of the four random variables specified in the artificial data example. The green and red dots represent the theoretical and empirical probabilities, respectively. The theoretical probabilities are calculated based on the pmf of each univariate GPD when its rate and dispersion parameters are specified. The empirical probabilities are marginally extracted from the multivariate data simulated in the artificial data example. We observe that the empirical and theoretical probabilities align closely, which suggests that our algorithm can accurately simulate data that follow the specified GPD marginally. In Figure 4, we show a comparison plot between specified and empirical correlations in an attempt to examine if our multivariate GPD generating algorithm can reasonably simulate the specified correlations. We specify a bivariate GPD using the variable 2 and variable 4, as previously defined in the artificial data example. The specified correlations range from -0.89 to 0.86 with an increasing step size of 0.05, which are within the lower and upper correlation bounds, as verified using the `ComputeCorrGpois` function. For each specified correlation, we generate a sample of 100 observations with 50 replications, and calculate the average empirical correlation. The plot shows that empirical correlations closely approximate the specified correlations for all scenarios.

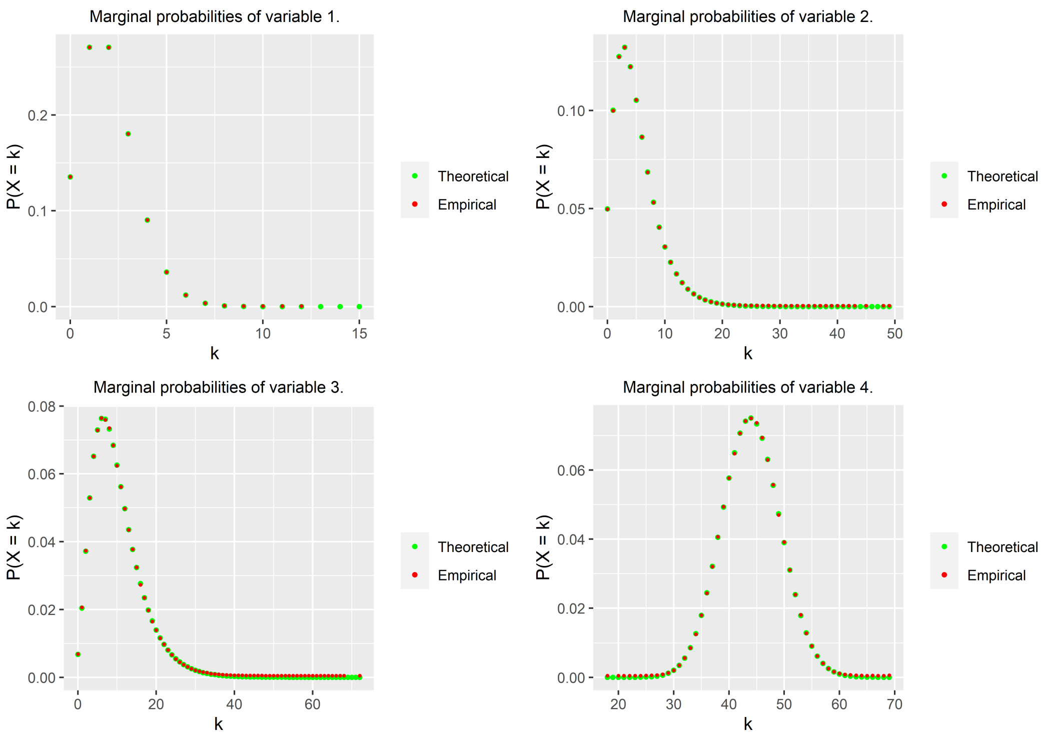


Figure 3: Comparison of theoretical and empirical marginal probabilities. The plot indicates that the empirical marginal probabilities closely approximate the theoretical marginal probabilities.

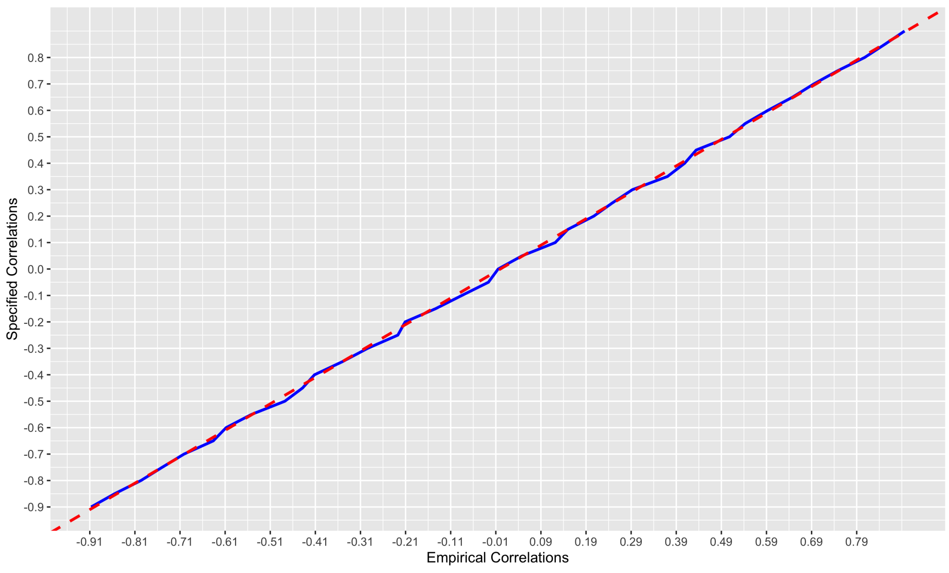


Figure 4: Comparison of specified and empirical correlations. The slope of the red dashed line is one, and is used for calibration. The blue solid line represents the empirical correlations plotted against the specified correlations at varying specifications.

Discussion

Throughout this paper, we have demonstrated the functionality and performance of the **RNGforGPD** package. This package is an accurate and computationally efficient tool in random generation of both univariate and multivariate data that follow the generalized Poisson distribution. Overall, the performance of the algorithm is decent. Deviations between the specified and average empirical parameter values are within tolerable limits in both the univariate and multivariate data generation cases. Our simulation studies suggest that this package successfully implements the algorithms in both artificial and real life scenarios as long as there are no specification errors and the correlations are within the feasible limits (Demirtas and Hedeker, 2011). In situations such complications occur, appropriate warning or error messages will be generated to alert the user. The simulation results we present can be regarded as a compelling evidence for capturing the characteristics of both rate and dispersion parameters (naturally the first four moments that are functions of these quantities) as well as the true association structure in the multivariate cases with only minor differences. In summary, the **RNGforGPD** package provides a valuable tool for investigators who need a generalized Poisson data generation mechanism in their research.

Bibliography

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. [p178]

H. Chen. Initialization for NORTA: Generation of random vectors with specified marginals and correlations. *INFORMS Journal on Computing*, 13(4):312–331, 2001. URL `https://doi.org/10.1287/ijoc.13.4.312.9736`. [p176, 179]

P. Consul and M. Shoukri. Some chance mechanisms related to a generalized Poisson probability model. *American Journal of Mathematical and Management Sciences*, 8(1-2):181–202, 1988. URL `http://doi.org/10.1080/01966324.1988.10737237`. [p175]

P. C. Consul. Generalized Poisson distribution: properties and applications. *Decker, New York*, 1989. [p173]

P. C. Consul and F. Famoye. *Lagrangian Probability Distributions*. Springer, 2006. URL `https://link.springer.com/book/10.1007/0-8176-4477-6`. [p174, 180]

P. Deb and P. K. Trivedi. Demand for medical care by the elderly: a finite mixture approach. *Journal of Applied Econometrics*, 12(3):313–336, 1997. URL `http://doi.org/10.1002/(SICI)1099-1255(199705)12:3<313::AID-JAE440>3.0.CO;2-G`. [p184]

H. Demirtas. On accurate and precise generation of generalized Poisson variates. *Communication in Statistics - Simulation and Computation*, 46:489–499, 2017. URL `https://doi.org/10.1080/03610918.2014.968725`. [p174, 177]

H. Demirtas and D. Hedeker. A practical way for computing approximate lower and upper correlation bounds. *The American Statistician*, 65(2):104–109, 2011. URL `http://doi.org/10.1198/tast.2011.10090`. [p176, 179, 187]

A. Genz, F. Bretz, T. Miwa, X. Mi, F. Leisch, F. Scheipl, and T. Hothorn. *mvtnorm: Multivariate Normal and t Distributions*, 2020. URL `https://CRAN.R-project.org/package=mvtnorm`. R package version 1.1-1. [p178]

S. Iovleff. *MixAll: Clustering and Classification using Model-Based Mixture Models*, 2019. URL `https://CRAN.R-project.org/package=MixAll`. R package version 1.5.1. [p184]

H. Joe and R. Zhu. Generalized Poisson distribution: the property of mixture of Poisson and comparison with negative Binomial distribution. *Biometrical Journal*, 47(2):219–229, 2005. URL `http://doi.org/10.1002/bimj.200410102`. [p173]

A. W. Kemp. *Frugal Methods of Generating Bivariate Discrete Random Variables*. Springer, 1981. URL `http://doi.org/10.1007/978-94-009-8549-0_28`. [p175]

I. Leppik. A double-blind crossover evaluation of progabide in partial seizures. *Neurology*, 35, 1985. [p182]

H. Li, R. Chen, H. Nguyen, Y.-C. Chung, R. Gao, and H. Demirtas. *RNGforGPD: Random Number Generation for Generalized Poisson Distribution*, 2020. R package version 1.1.0. [p174]

M. Maechler, P. Rousseeuw, C. Croux, V. Todorov, A. Ruckstuhl, M. Salibian-Barrera, T. Verbeke, M. Koller, E. L. T. Conceicao, and M. Anna di Palma. *robustbase: Basic Robust Statistics*, 2020. URL http://robustbase.r-forge.r-project.org/. R package version 0.93-6. [p182]

F. Satterthwaite. Generalized Poisson distribution. *The Annals of Mathematical Statistics*, 13(4):410–417, 1942. URL http://doi.org/10.1214/aoms/1177731538. [p173, 174]

J. Schafer, R. Opgen-Rhein, V. Zuber, M. Ahdesmaki, 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. [p178]

P. F. Thall and S. C. Vail. Some covariance models for longitudinal count data with overdispersion. *Biometrics*, 46(3):657–671, 1990. URL http://doi.org/10.2307/2532086. [p182]

R. Vernic. On the bivariate generalized Poisson distribution. *ASTIN Bulletin: The Journal of the International Actuarial Association*, 27(01):23–32, 1997. URL https://doi.org/10.2143/AST.27.1.542065. [p174]

R. Vernic. A multivariate generalization of the generalized Poisson distribution. *ASTIN Bulletin: The Journal of the International Actuarial Association*, 30(1):57–67, 2000. doi: 10.2143/AST.30.1.504626. URL https://doi.org/10.2143/AST.30.1.504626. [p173]

H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. ISBN 978-3-319-24277-4. URL https://ggplot2.tidyverse.org. [p182]

I. Yahav and G. Shmueli. On generating multivariate Poisson data in management science applications. *Applied Stochastic Models in Business and Industry*, 28(1):91–102, 2012. URL http://doi.org/10.1002/asmb.901. [p174, 176]

Hesen Li
Ph.D. Student
Division of Epidemiology and Biostatistics
University of Illinois at Chicago
Chicago, IL 60612, USA
ORCID: 0000-0003-1636-299X
hli226@uic.edu

Dr. Hakan Demirtas
Associate Professor of Biostatistics
Division of Epidemiology and Biostatistics
University of Illinois at Chicago
Chicago, IL 60612, USA
ORCID: 0000-0003-2482-703X
demirtas@uic.edu

Ruizhe Chen
Ph.D. Student
Division of Epidemiology and Biostatistics
University of Illinois at Chicago
Chicago, IL 60612, USA
ORCID: 0000-0003-3924-3328
rchen18@uic.edu

Testing the Equality of Normal Distributed and Independent Groups' Means Under Unequal Variances by doex Package

by Mustafa Cavus and Berna Yazıcı

Abstract In this paper, we present the **doex** package contains the tests for equality of normal distributed and independent group means under unequal variances such as Cochran F, Welch-Aspin, Welch, Box, Scott-Smith, Brown-Forsythe, Johansen F, Approximate F, Alexander-Govern, Generalized F, Modified Brown-Forsythe, Permutation F, Adjusted Welch, B2, Parametric Bootstrap, Fiducial Approach, and Alvandi Generalized F-test. Most of these tests are not available in any package. Thus, **doex** is easy to use for researchers in multidisciplinary studies. In this study, an extensive Monte-Carlo simulation study is conducted to investigate the performance of the the tests for equality of normal distributed group means under unequal variances in terms of Type I error probability and penalized power. In the case of Type I error probability of the compared tests are different, the penalized power is used which allows fair power comparisons. In this way, we conclude the performance of the tests by taking into account two possible errors in hypothesis testing.

Introduction

Testing equality of normal distributed and independent groups' means is a basic analysis in statistics and related fields. The Fisher's F-test is a powerful test to do this analysis with the assumptions of variance homogeneity, normality, and statistical independency. Violation of the variance homogeneity assumption is a commonly encountered statistical problem in a variety of application areas such as agriculture, pharmacy, and biostatistics. There is number of methods improved because of the negative effect of the violation of variance homogeneity assumption on the performance of Classical F-test in terms of Type I error probability and power. These tests are, Cochran F (CF), Welch-Aspin (WA), Welch (WE), Box (BX), Scott-Smith (SS), Brown-Forsythe (BF), Johansen (JF), Approximate F (AF), Alexander-Govern (AG), Generalized F (GF), Modified Brown-Forsythe (MBF), Permutation F (PF), Adjusted Welch (AW), B2, Parametric Bootstrap (PB), Fiducial Approach (FA) and Alvandi et al. Generalized F (AGF) test, chronologically. The fact that the high number of methods in the literature raises the problem of choosing the most appropriate method for researchers.

There are many articles to investigate the performance of the tests for equality of normal distributed and independent group means under unequal variances in the literature. However, only some of the tests are included in these studies. The results of these studies help researchers to solve the problem of choosing the appropriate method for their work. Gamage and Weerahandi (1998) compared the size performance of the GF test and four widely used procedures: CF, BF, and Welch test in case of deviation from normality. The highly skewed Gamma distributions and Gamma distributions with shapes close to being normal are considered. While the GF was found to have size not exceeding the intended level, as heteroscedasticity becomes severe the others were found to have poor size performance. Hartung et al. (2002) compared the CF, C, W, BF, MBF, AF, and AW tests under normal populations, balanced-unbalanced sample sizes and an increasing number of populations. None of the tests considered is uniformly dominating the others. The BF and the W test perform well over a wide range of parameter configurations, the MBF test by Mehrotra keeps generally the level, but other tests may also perform well, depending on the constellation of the parameters under study. The W test becomes liberal when the sample sizes are small and the number of populations is large. They propose a modified version of Welch's test that keeps the nominal level in these cases. With the understanding that methods are unacceptable if they have Type I error rates that are too high, only the testing procedure associated with the MBF test can be recommended, the modified Welch test can also be recommended. Argac (2004) constructed a systematic pattern in simulations of the tests for equality of normal distributed and independent group means under unequal variances. Classical F, Cochran, Welch, modified Welch, Brown-Forsythe, modified Brown-Forsythe, and approximate F test considered are divided into two groups, Cochran-Welch type tests and the Brown-Forsythe type tests. There seems to be considerably higher variability in the power of C-W type tests in the balanced case. In the unbalanced case, there does not appear to be a huge difference between the power of the different tests. Sadooghi-Alvandi et al. (2012) proposed a new GF test and compared it with GF, PB, Welch, and Cochran test in an extensive Monte-Carlo simulation study. According to results, it controls the Type I error probability better and its power closed to the others. Gokpinar and Gokpinar (2012) compared the Type I error probability and power of CF, BF, GF, PB, and W test under different variance heterogeneities and effect sizes for three and five groups. Their results indicate that PB is the best control Type I error probability and has the highest power. In addition to these articles, the scope of the other articles are not comprehensive in the literature (Hartung et al. (2002), Lee and Ahn (2003), Li et al. (2011), Mutlu et al. (2017)). A comprehensive Monte-Carlo simulation study is conducted under normal distribution in this article in order to fill this gap. Especially, the penalized power is used which allows fair power comparisons when the Type I error probabilities are different. In this way, we conclude the performance of the tests by taking into account two possible errors in hypothesis testing.

Another problem experienced by the researchers is most of these tests are not available in any R package. However, some R packages contain the tests for equality of normal distributed and independent group means under unequal variances, **asbio** by Aho (2018), **coin** by Hothorn et al. (2008), **lawstat** by Hui et al. (2008), **onewaytests** by Dag et al. (2018), **welchADF** by Villacorta (2017), **WRS2** by Mair and Wilcox (2018). These packages contain only the Brunner-Dette-Munk, Permutation F, Kruskal-Wallis, Brown-Forsythe, Alexander-Govern, James Second Order, Welch test. In particular, the performance of the tests such as the GF, PB, FA, and AGF test by Monte-Carlo simulations prevents the easy use of these tests. Clearly, a package should contain these tests. We propose the package **doex** provides the tests for equality of normal distributed group means under unequal variances which previously have not been implemented in any R package such as AF, AGF, B2, FA, JF, MBF, MW, PB, and PF. Also, it consists of the modified Generalized F-test (MGF) which is proposed by Cavus et al. (2017) to test the equality of group means under heteroscedasticity and non-normality caused by outliers. It is a useful procedure for non-normal distributed groups and Cavus et al. (2018) showed in a real data application.

The following sections detail the tests for equality of normal distributed and independent group means under unequal variances considered in **doex**. The performance of these tests is investigated in terms of penalized power and Type I error probability. Finally, we conclude with a brief summary and future works.

Tests for Testing Equality of Normal Distributed Groups' Means under Unequal Variance

The linear model within the context of a one-way independent group design for testing the equality of groups' means is given in (1) .

$$Y_{ij} = \mu_i + \epsilon_{ij} \tag{1}$$

where Y_{ij} is the dependent variable associated with the ith observation in the jth group for $i = 1, 2, ..., n_i$ and $j = 1, 2, ..., k$. μ_i is the group mean for the ith group, and ϵ_{ij} is the random error component associated with Y_{ij}. The null hypothesis $H_0 : \mu_1 = \mu_2 = ... = \mu_k$ is tested as the Classical F-test assumed that the ϵ_{ij}'s are independent, normally distributed, and have an equal variance σ^2 for each group of k. Type I error probability of Classical F-test inflates and its power decreases in case of the violation of variance homogeneity assumption. There are many procedures improved in the literature to solve this problem. In this section, the tests for equality of normal distributed and independent group means under unequal variances, considered in **doex** and discussed in the Monte-Carlo simulation study, are introduced. These tests are, Alexander-Govern, Alvandi et al. generalized F, Approximate F, Box F, Brown-Forsythe, B^2, Cochran F, Fiducial Approach, Generalized F, Johansen, Modified Brown-Forsythe, Adjusted Welch, Parametric Bootstrap, Permutation F, Scott-Smith, Welch, Welch-Aspin test.

Alexander-Govern (AG) test

Alexander and Govern (1994) improved a test using the Hill's normality transformation to the Student's t variables. Consider $X_{i1}, X_{i2}, ..., X_{in_i} \sim N(\mu_i, \sigma_i^2)$ and the standard deviations of normal groups computed as in (2).

$$S_{\bar{X}_i} = \frac{\sum_{i=1}^{k} \sum_{j=1}^{n_i} (X_{ij} - \bar{X}_i)^2}{n_i(n_i - 1)} \tag{2}$$

The weights are computed using the $S_{\bar{X}_i}$ as in (3).

$$wi = \frac{1/S^2_{\bar{X}_i}}{\sum_{i=1}^{k} 1/S^2_{\bar{X}_i}} \tag{3}$$

The weight mean is computed using the w_i in (4).

$$\bar{X}^* = \sum_{i=1}^{k} w_i \bar{X}_i \tag{4}$$

The values of $t_i = (\bar{X}_i - \bar{X}^*)/S_{\bar{X}_i} \sim t_{n_i-1}$ are transformed using the following transformation.

$$z_i = c + \frac{c^3 + 3c}{b} + \frac{4c^7 + 33c^5 + 240c^3 + 855c}{10b^2 + 8bc^4 + 1000b} \tag{5}$$

where $a = v_i - 0.5, c = \sqrt{aln(1 + \frac{t_i^2}{v_i})}$ and $b = 48a^2$. The test statistic of AG test is computed as in (6).

$$T_{AG} = \sum_{i=1}^{k} z_i^2 \tag{6}$$

The H_0 is rejected when $T_{AG} > \chi^2_{(k-1);\alpha}$.

Alvandi et. al. Generalized F (AGF) test

Sadooghi-Alvandi et al. (2012) proposed the test statistic in 8 as an alternative of Weerahandi's Generalized F-test.

$$T_G(S_1^2, S_2^2, ..., S_k^2) = \sum_{i=1}^{k} \frac{n_i}{S_i^2} \bar{X}_i - \frac{[\sum_{i=1}^{k} n_i \bar{X}_i / S_i^2]^2}{\sum_{i=1}^{k} n_i / S_i^2} \tag{7}$$

$$T_{AGF} = \sum_{i=1}^{k} \frac{n_i - 1}{U_i} (\bar{X}_i - q_i \tilde{X})^2 \tag{8}$$

where $q_i = \sqrt{\frac{n_i/s_i^2}{\sum_{i=1}^{k} n_i/s_i^2}}$ and $\tilde{X} = \sum_{i=1}^{k} q_i \bar{X}_i$. The p-value of AGF test computed using Monte-Carlo simulations with **Algorithm 1**.

Algorithm 1. Computation of Monte-Carlo estimate of the AGF test

```
1. Compute the vectors of $(\bar{x}_1, \bar{x}_2, ..., \bar{x}_k)$ and $(s_1^2, s_2^2, ..., s_k^2)$ for $k$ groups
2. Compute the $T_G$ using the vectors in **Step 1**
3. **for** $j$ in $\{1, ..., r\}$ **do**
       Generate $U_i \sim \chi^2_{n_i-1}$ random samples
       Compute the $T_{AGF}$ using generated random samples
       Set the counter $Q_j = 1$ when $T_{AGF} > T_G$
   **end for**
4. Compute the Monte-Carlo estimate of p-value as $\sum_{i=1}^{k} Q_j / r$
```

Approximate F (AF) test

Asiribo and Gurland (1990) proposed a modification to the F-test as in (9).

$$T_{AF} = N \frac{\sum_{i=1}^{k} n_i (\bar{X}_i - \bar{X}_{..})^2}{\sum_{i=1}^{k} (N - n_i) S_i^2} \tag{9}$$

where $\bar{X}_{..} = \sum_{i=1}^{k} \bar{X}_i$ and $N = \sum_{i=1}^{k} n_i$. The H_0 is rejected when $T_{AF} > F_{v1,v2;\alpha}$. The degrees of freedom of the AF test statistic is computed in (10).

$$v_1 = \frac{[\sum_{i=1}^{k} (1 - n_i/N) S_i^2]^2}{\sum_{i=1}^{k} S_i^4 + [\sum_{i=1}^{k} n_i S_i^2 / N]^2 - 2\sum_{i=1}^{k} n_i S_i^4 / N}, \qquad v_2 = \frac{[\sum_{i=1}^{k} (1 - n_i/N)^2 S_i^2]^2}{\sum_{i=1}^{k} (n_i - 1) S_i^4} \tag{10}$$

Box (BX) test

Box (1954) proposed the test statistic in (11).

$$T_{BO} = \frac{N_k}{N(k-1)\sum_{i=1}^{k} \frac{(N-n_i)S_i^2}{(n_i-1)S_i^2}} \tag{11}$$

The H_0 is rejected when $T_{BO} > F_{v_1,v_2;\alpha}$ where

$$v_1 = \frac{[\sum_{i=1}^{k}(N-n_i)S_i^2]^2}{[\sum_{i=1}^{k} n_i S_i^2]^2 + N\sum_{i=1}^{k}(N-2n_i)S_i^2}, \qquad v_2 = \frac{[\sum_{i=1}^{k}(n_i - 1S_i^2)]^2}{\sum_{i=1}^{k}(n_i-1)S_i^+} \tag{12}$$

Brown-Forsythe (BF) test

Brown and Forsythe (1974) proposes the following test statistic.

$$T_{BF} = \frac{\sum_{i=1}^{k} n_i(\bar{X}_i - X_{..})^2}{\sum_{i=1}^{k}(1-n_i/N)S_i^2} \tag{13}$$

where $X_{..} = \sum_{i=1}^{k}\bar{X}_i$ and $N = \sum_{i=1}^{k} n_i$. The H_0 is rejected when $T_{BF} > F_{(k-1),v;\alpha}$. The degrees of fredom of the test statistic computed as in (14).

$$v = \frac{[\sum_{i=1}^{k} n_i(\bar{X}_i - X_{..})]^2}{\sum_{i=1}^{k}\frac{(1-n_i/N)^2 S_i^4}{(n_i-1)}} \tag{14}$$

The B^2 test

Ozdemir and Kurt (2006) proposed the following procedure using the Bailey's normality transformation to the Student's t variables. Consider $X_{i1}, X_{i2}, ..., X_{in_i} \sim N(\mu_i, \sigma_i^2)$ and the standard deviations of normal groups computed as in (15).

$$S_{\bar{X}_i} = \frac{\sum_{i=1}^{k}\sum_{j=1}^{n_i}(X_{ij} - \bar{X}_i)^2}{n_i(n_i-1)} \tag{15}$$

The weights computed using the $S_{\bar{X}_i}$ as in (16).

$$w_i = \frac{1/S_{\bar{X}_i}^2}{\sum_{i=1}^{k} 1/S_{\bar{X}_i}^2} \tag{16}$$

The weighed mean computed using the w_i's as in (17).

$$\bar{X}^* = \sum_{i=1}^{k} w_i \bar{X}_i. \tag{17}$$

The values of $t_i = (\bar{X}_i - \bar{X}^*)/S_{\bar{X}_i} \sim t_{n_i-1}$ are transformed using Bailey's (1980) normality transformation.

$$z_i = \frac{4v_i^2 + \frac{5(2z_c^2+3)}{24}}{4v_i^2 + v_i + \frac{4z_c^2+9}{12}}\sqrt{v_i ln(1 + \frac{t_i^2}{v_i})} \sim N(0,1) \tag{18}$$

where $z_c = Z_{\alpha/2} \sim N(0,1)$ and the test statistic of B^2 test computed as in (19).

$$T_{BK} = \sum_{i=1}^{k} z_i^2 = \sum_{i=1}^{k}(\frac{4v_i^2 + \frac{5(2z_c^2+3)}{24}}{4v_i^2 + v_i + \frac{4z_c^2+9}{12}})^2. \tag{19}$$

The H_0 is rejected when $T_{BK} > \chi^2_{(k-1);\alpha}$.

Cochran (CF) test

Cochran (1937) proposes the test statistic in (20).

$$T_C = \sum_{i=1}^{k} w_i(\bar{X}_i - \sum_{j=1}^{k} h_j\bar{X}_j)^2 \tag{20}$$

where $w_i = n_i/s_i^2$ and $h_i = w_i/\sum_{i=1}^{k} w_i$. The H_0 is rejected when $T_C > \chi^2_{(k-1);\alpha}$.

Fiducial Approach (FA) test

Li et al. (2011) proposed the test statistic in (21).

$$T_{FA} = \sum_{i=1}^{k} t_i^2 - \frac{(\sum_{i=1}^{k} \frac{\sqrt{n_i}}{S_i} t_i)^2}{\sum_{i=1}^{k} \frac{n_i}{S_i^2}}. \tag{21}$$

The p-value of the FA test can be computed using Monte-Carlo simulations with **Algorithm 2**.

Algorithm 2. Computation of Monte-Carlo estimate of the FA test

```
1.  Compute the vectors of $(\bar{x}_1, \bar{x}_2, ..., \bar{x}_k)$ and $(s_1^2, s_2^2, ..., s_k^2)$ for $k$ groups
2.  Compute the $T_G$ using the vectors in Step 1
3.  for $j$ in $\{1, ..., r\}$ do
        Generate $Z_i \sim N(0,1)$ and $U_i \sim \chi^2_{n_i-1}$ random samples
        Compute the $T_{FA}$ using generated random samples
        Set the counter $Q_j = 1$ when $T_{FA} > T_G$
    end for
4.  Compute the Monte-Carlo estimate of p-value as $\sum_{i=1}^{k} Q_j/r$
```

Generalized F (GF) test

Weerahandi (1995) proposed the test statistic in (22) using the generalized p-value approach.

$$T_{GF} = \sum_{i=1}^{k} (n_iU_i/v_i^2)\bar{x}_i^2 - \frac{[\sum_{i=1}^{k}(n_iU_i/v_i^2)\bar{x}_i]^2}{\sum_{i=1}^{k} n_iU_i/v_i^2} \tag{22}$$

where $v_i^2 = (n_i - 1)S_i^2$. The p-value of GF test can be computed using Monte-Carlo simulations with **Algorithm 3**.

Algorithm 3. Computation of Monte-Carlo estimate of the GF test

```
1.  Compute the vectors of $(\bar{x}_1, \bar{x}_2, ..., \bar{x}_k)$ and $(s_1^2, s_2^2, ..., s_k^2)$ for $k$ groups
2.  Compute the $T_G$ using the vectors in Step 1
3.  for $j$ in $\{1, ..., r\}$ do
        Generate $U_i \sim \chi^2_{n_i-1}$ random samples
        Compute the $T_{GF}$ using generated random samples
        Set the counter $Q_j = 1$ when $T_{GF} > T_G$
    end for
4.  Compute the Monte-Carlo estimate of p-value as $\sum_{i=1}^{k} Q_j/r$
```

Johansen (JF) test

Johansen (1980) proposed an approximate solution to the W test as in (23).

$$T_J = \frac{\sum_{i=1}^{k} \frac{\bar{X}_i^2}{S_i^2} - \frac{[\sum_{i=1}^{k} \bar{X}_i/S_i^2]^2}{\sum_{i=1}^{k} 1/S_i^2}}{c} \tag{23}$$

where $c = (k-1) + 2A - 6A/(k+1)$, $v = (k-1)(k+1)/3A$ and $A = \sum_{i=1}^{k}(1 - w_i/w)^2/(n_i - 1)$ The H_0 is rejected when $T_J > F_{k-1,v;\alpha}$.

Modified Brown-Forsythe (MBF) test

Mehrotra (1997) proposed the test statistic in (24), which is a modification of BF, to well-performing in case of small sample size.

$$T_{MBF} = \frac{\sum_{i=1}^{k} n_i(\bar{X}_i - \bar{X}_{..})^2}{\sum_{i=1}^{k}(1 - n_i/N)S_i^2}. \tag{24}$$

where $\bar{X}_{..} = \sum_{i=1}^{k} \bar{X}_i$ and $N = \sum_{i=1}^{k} n_i$. The H_0 is rejected when $T_{MBF > F_{v_1,v_2;\alpha}}$. The degrees of freedom of the MBF test statistics is computed as in (25).

$$v_1 = \frac{[\sum_{i=1}^{k}(1 - n_i/N)S_i^2]^2}{\sum_{i=1}^{k} S_i^4 + (\sum_{i=1}^{k} n_i S_i^2/N)^2 - 2\sum_{i=1}^{k} n_i S_i^4/N}, \qquad v_2 = \frac{[\sum_{i=1}^{k}(1 - n_i/N)^2 S_i^2]^2}{\sum_{i=1}^{k} \frac{(1-n_i/N)^2 S_i^4}{n_i - 1}}. \tag{25}$$

Adjusted Welch (AW) test

Hartung et al. (2002) proposed an adjustment to the Welch test. The test statistic of adjusted Welch test is computed as in (26).

$$T_W = \frac{\sum_{i=1}^{k} w_i^*(\bar{x}_i - \sum_{j=1}^{k} h_j^* \bar{x}_j)^2}{(k-1) + 2\frac{k-2}{k+1}\sum_{i=1}^{k}\frac{1}{n_i - 1}(1 - h_j^*)^2}. \tag{26}$$

where $w_i^* = [\frac{n_i}{(n_i - 1/n_i - 3)s_i^2}]$ and $h_i^* = \frac{w_i^*}{\sum_{i=1}^{k} w_i^*}$. The H_0 is rejected when $T_W > F_{(k-1),v;\alpha}$. The degrees of freedom of the test statistic computed in (27).

$$v = \frac{\frac{k^2 - 1}{3}}{\sum_{i=1}^{k}\frac{(1-h_i^*)^2}{n_i - 1}}. \tag{27}$$

Parametric Bootstrap (PB) test

Krishnamoorthy et al. (2007) proposed a procedure to test the equality of group means under heteroscedasticity.

$$T_G(S_1^2, S_2^2, ..., S_k^2) = \sum_{i=1}^{k}\frac{n_i}{S_i^2}\bar{X}_i - \frac{[\sum_{i=1}^{k} n_i \bar{X}_i / S_i^2]^2}{\sum_{i=1}^{k} n_i / S_i^2} \tag{28}$$

Assume $Z_i \sim N(0,1)$ and $U_i \sim \chi^2_{n_i - 1}$ random samples, the test statistic of the PB test is computed as in (29).

$$T_{PB}(S_1^2, S_2^2, ..., S_k^2) = \sum_{i=1}^{k}\frac{Z_i^2(n_i - 1)}{U_i} - \frac{[\sum_{i=1}^{k}\sqrt{n_i} Z_i (n_i - 1)/S_i U_i]^2}{\sum_{i=1}^{k} n_i(n_i - 1)/S_i^2 U_i} \tag{29}$$

The H_0 is rejected when $T_{PB} > T_G$. The p-value of PB test is computed using Monte-Carlo simulations with **Algorithm 4**.

Algorithm 4. Computation of Monte-Carlo estimate of the PB test

```
1.  Compute the vectors of $(\bar{x}_1, \bar{x}_2, ..., \bar{x}_k)$ and $(s_1^2, s_2^2, ..., s_k^2)$ for $k$ groups
2.  Compute the $T_G$ using the vectors in Step 1
3.  for $j$ in $\{1, ..., r\}$ do
        Generate $Z_i \sim N(0,1)$ and $U_i \sim \chi^2_{n_i - 1}$ random samples
        Compute the $T_{PB}$ using generated random samples
        Set the counter $Q_j = 1$ when $T_{PB} > T_G$
    end for
4.  Compute the Monte-Carlo estimate of p-value as $\sum_{i=1}^{k} Q_j / r$
```

Permutation F (PF) test

Berry and Mielke (2002) proposed the test statistic in (30) as the permutational alternative of F-test.

$$T_{PF} = \frac{(T - N\bar{X}^*)/(k-1)}{(V-T)/(N-k)} \tag{30}$$

where $T = \sum_{i=1}^{k} n_i \sum x_i^2$, $\bar{X}^* = 1/N \sum n_i \bar{x}_i$ and $V = \sum_{i=1}^{k}\sum_{j=1}^{n_i} X_{ij}^2$. The H_0 is rejected when $T_{PF} > F_{k-1,N-k;\alpha}$.

Scott-Smith (SS) test

Scott and Smith (1971) proposed the test statistic in (31).

$$T_{SC} = \sum_{i=1}^{k} \frac{n_i(\bar{X}_i - \bar{X}_{..})^2}{S_i^{*2}} \tag{31}$$

where $S_i^{*2} = \frac{n_i-1}{n_i-3} S_i^2$. The H_0 is rejected when $T_{SC} > \chi^2_{k;\alpha}$.

Welch (WE) test

Welch (1951) improved the test statistic in 32 based on the weighted group variance as an alternative to the F-test under heteroscedasticity.

$$T_W = \frac{\sum_{i=1}^{k} w_i(\bar{x}_i - \sum_{i=1}^{k} h_j \bar{x}_j)^2}{(k-1) + 2\frac{k-2}{k+1}\sum_{i=1}^{k}\frac{1}{n_i-1}(1-h_i)^2} \tag{32}$$

where $w_i = n_i/s_i^2$ and $h_i = w_i/\sum_{i=1}^{k} w_i$. The H_0 is rejected when $T_W > F_{(k-1),v;\alpha}$. The degrees of freedom of the Welch test computed as in 33.

$$v = \frac{(k^2-1)/3}{\sum_{i=1}^{k}\frac{(1-h_i)^2}{n_i-1}} \tag{33}$$

Welch-Aspin (WA) test

Aspin (1948) proposed the test statistic in (34) with a modification to the degrees of freedom of Welch test.

$$T_{WA} = \frac{\sum_{i=1}^{k}(\bar{X}_i - \bar{X})^2/S_i^2}{(k-1)[1 + \frac{2k-2}{k^2-1}\lambda]} \tag{34}$$

where $\lambda = \sum_{i=1}^{k}[(1-w_i)^2/w_i]$, $v_1 = k-1$ and $v_2 = (k^2-1)/3\lambda$. The H_0 is rejected when $T_{WA} > F_{v_1;v_2;\alpha}$.

Using doex package

The **doex** package provides to perform several tests for equality of normal distributed and independent distributed group means under unequal variances. These tests are called a function with the initials of their name which are given in the previous sections. In particular, the following tests are not included in any R package or statistical package program: AF, AGF, B2, FA, JF, MBF, MW, PB, and PF. In this section, the examples are given how to use these tests by using doex. After the explanatory data analysis, the variance homogeneity assumption must be checked to move on to the next stage (Noguchi and Gel, 2010; Erps and Noguchi, 2019). The Levene Test is used to this, and we did not include it in the package is because it is included in many R package such as **car** by Fox and Weisberg (2019), **rstatix** by Kassambara (2020), **lawstat** by Gastwirth et al. (2020), **inferr** by Hebbali (2018). We want to stick with the idea of creating a package that includes tests not included in the CRAN.

Example 1: The data are inputted to the functions with two parts: observations and the group labels. As an example `hybrid` data from Weerahandi (1995) is given in the package. It consists of two parts: `data` are observations and `species` are the labels of species of the corn hybrids.

```
# Call the doex package
> library(doex)
# print hybrid data of Weerahandi (1995)
> hybrid

    data species
 1   7.4       A
 2   6.6       A
 3   6.7       A
 4   6.1       A
 5   6.5       A
 6   7.2       A
 7   7.1       B
 8   7.3       B
 9   6.8       B
 10  6.9       B
 11  7.0       B
 12  6.8       C
 13  6.3       C
 14  6.4       C
 15  6.7       C
 16  6.5       C
 17  6.8       C
 18  6.4       D
 19  6.9       D
 20  7.6       D
 21  6.8       D
 22  7.3       D

# observations of the hybrid data
> hybrid$data
 [1] 7.4 6.6 6.7 6.1 6.5 7.2 7.1 7.3 6.8 6.9 7.0 6.8 6.3 6.4 6.7 6.5 6.8 6.4 6.9 7.6 6.8 7.3

 # group labels of the hybrid data
> hybrid$species
 [1] A A A A A A B B B B B C C C C C C D D D D D
 Levels: A B C D

 # The ggplot2 package can be used to plot the box plot of the data in Figure 1.
> ggplot(hybrid, aes(x = species, y = data)) +
>  geom_boxplot() +
>  ylab("Yield") +
>  xlab("Corn Species")


# Look at the summary statistics of the data before using the tests.
# Use psych package to obtain the descriptive statistics of the hybrid data
> library(psych)

# Describe the hybrid data by species using describe.by(.) function
> describe.by(hybrid$data, hybrid$species)

#The output of the describe.by function as follows:

Descriptive statistics by group
group: A
   vars n mean   sd median trimmed  mad min max range skew kurtosis   se
X1    1 6 6.75 0.48   6.65    6.75 0.52 6.1 7.4   1.3 0.11     -1.7 0.19
-----------------------------------------------------------
group: B
   vars n mean   sd median trimmed  mad min max range skew kurtosis   se
X1    1 5 7.02 0.19      7    7.02 0.15 6.8 7.3   0.5 0.28    -1.72 0.09
-----------------------------------------------------------
group: C
   vars n mean   sd median trimmed mad min max range  skew kurtosis   se
```

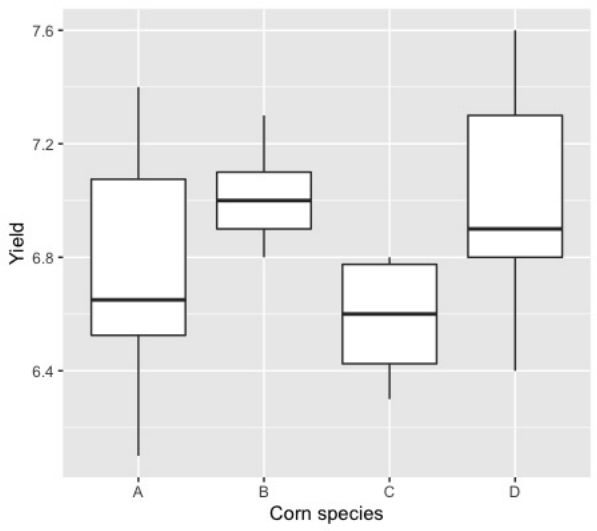


Figure 1: Boxplot of the data in Example 1.

```
X1    1 6 6.58 0.21    6.6    6.58 0.3 6.3 6.8   0.5 -0.13    -2.02 0.09
----------------------------------------------------------
group: D
   vars n mean   sd median trimmed  mad min max range skew kurtosis   se
X1    1 5    7 0.46    6.9       7 0.59 6.4 7.6   1.2 0.04    -1.84 0.21

# It is seen that the variances of the species are unequal
# Thus we need to use the tests for equality of the group means under unequal variances
#
# Examples of the use of the AF and GF tests on the hybrid data are given in the follows.
# The following code performs the Approximate F-test on the hybrid data.

> library(doex)
> AF(hybrid$data,hybrid$species)

# This function returns a result matrix consists of a test statistic, degrees of freedom,
and p-value of Approximate F-test as follows:

              Test Statistic df1 df2 p-value
Approximate F         1.8538   2  12  0.1943

# Following code performs the Generalized F-test.

> library(doex)
> GF(hybrid$data,hybrid$species)

# The p-value of the GF test is computed Monte-Carlo estimates and its size is
# controlled with the rept parameter in the function. It is implemented as
# default rept=10000
# This function returns the p-value of the Generalized F-test as follows:

              p-value
Generalized F  0.0492

# The results of the AF and GF tests are different at the nominal level 0.05.
# It is needed to investigate the performance of these tests in
# a Monte-Carlo simulation study.
```

Example 2: This example is provided an external data involves litter weights of mice born from mothers assigned to three different dosage groups and a control. For the low dose group the dose metameter is 5, for the medium dose group it is 50, and for the high dose group it is 500. In here, the problem is testing the equality of mean of litter weights of mice born according to the used dose. The dataset is available in the following repository: https://github.com/mcavs/doex_TheRJournal.

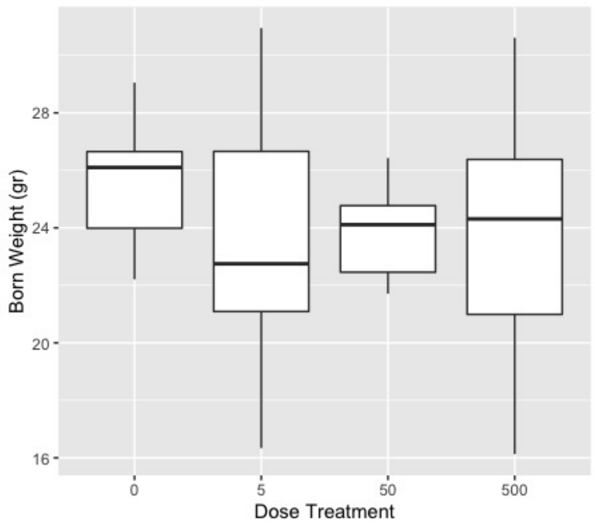


Figure 2: Boxplot of the data in Example 2.

```
# Print born weight data using the data is given in GitHub repository.

   weight_data dose
1        22.69    0
2        26.59    0
3        28.85    0
4        28.03    0
5        29.05    0
6        23.61    0
7        22.21    0
8        26.81    0
9        26.01    0
10       25.98    0
.        .        .
.        .        .
.        .        .
70       26.31  500
71       30.61  500
72       26.48  500
73       24.31  500
74       27.98  500

# The ggplot2 package can be used to plot the box plot of the data in Figure 2.
> ggplot(born_weight_data, aes(x = dose, y = weight_data)) +
>  geom_boxplot() +
>  ylab("Born Weight (gr)") +
>  xlab("Dose Treatment")



# Describe the born weight data by species using describe.by(.) function
> describe.by(born_weight_data$weight_data, born_weight_data$dose)

#The output of the describe.by function as follows:

 Descriptive statistics by group
group: 0
   vars  n  mean   sd median trimmed  mad   min   max range skew kurtosis   se
X1    1 20 25.73 2.02   26.1   25.74 2.43 22.21 29.05  6.84 -0.1    -1.16 0.45
-------------------------------
group: 5
   vars  n  mean  sd median trimmed  mad   min   max range skew kurtosis   se
X1    1 19 23.52 3.9  22.75   23.51 4.28 16.34 30.95 14.61 0.01    -0.97 0.89
```

```
-------------------------------
group: 50
   vars  n  mean   sd median trimmed  mad   min   max range  skew kurtosis   se
X1    1 18 23.79 2.83  24.11   23.84 1.92 17.54 29.21 11.67 -0.28     0.03 0.67
-------------------------------
group: 500
   vars  n  mean   sd median trimmed  mad   min   max range skew kurtosis   se
X1    1 17 23.72 4.08  24.31   23.76 3.84 16.13 30.61 14.48 -0.4    -0.91 0.99

# It is seen that the variances of the dose groups may be unequal
# To conclude whether the variance homogenity assumption is valid,
# Levene test is used.

> library(car)
> car::LeveneTest(weight_data ~ dose)

# LeveneTest(.) function returns the test statistic and
# p-value of Levene variance homogeneity test as follows:

Levene's Test for Homogeneity of Variance (center = median)
      Df F value Pr(>F)
group  3  3.3819 0.0229 *
---
Signif. codes:
  0 '***' 0.001 '**' 0.01 '*' 0.05 '.'

# The p-value of Levene test is lower than the nominal level 0.05,
# so it is concluded that the variance homogeneity assumption is violated.
# Thus we need to use the tests for equality of the group means under
# unequal variances in doex.

# The GF, AF, and PB are used to conclude there is a significance difference between
# the mean born weight of mice according to used dose group.

> doex::GF(weight_data, dose)

              p-value
Generalized F  0.0331

> doex::AF(weight_data, dose)

              Test Statistic df1 df2 p-value
Approximate F         1.9408   3  57  0.1484

> doex::PB(weight_data, dose)

                     p-value
Parametric Bootstrap  0.0366

# The results of the GF and PB tests indicate that there is a significant difference,
# while the result of the AF indicates that there is no significant difference between
# the mean born weight of mice according to used dose group.
# It is also needed to investigate the performance of these tests in
# a Monte-Carlo simulation study.
```

Monte-Carlo simulation study

In this section, the performance of the tests for equality of normal distributed and independent groups' means under unequal variances are investigated in terms of Type I error probability and penalized power of the test. We used the penalized power instead of the classical power of the test, because any comparison of the powers is invalid when Type I error probabilities are different in Monte-Carlo simulation studies. Zhang and Boos (1994) and Lloyd (2005) proposed alternatives for the power of the tests have some deficiencies. To overcome this problem, Cavus et al. (2019) proposed the penalized

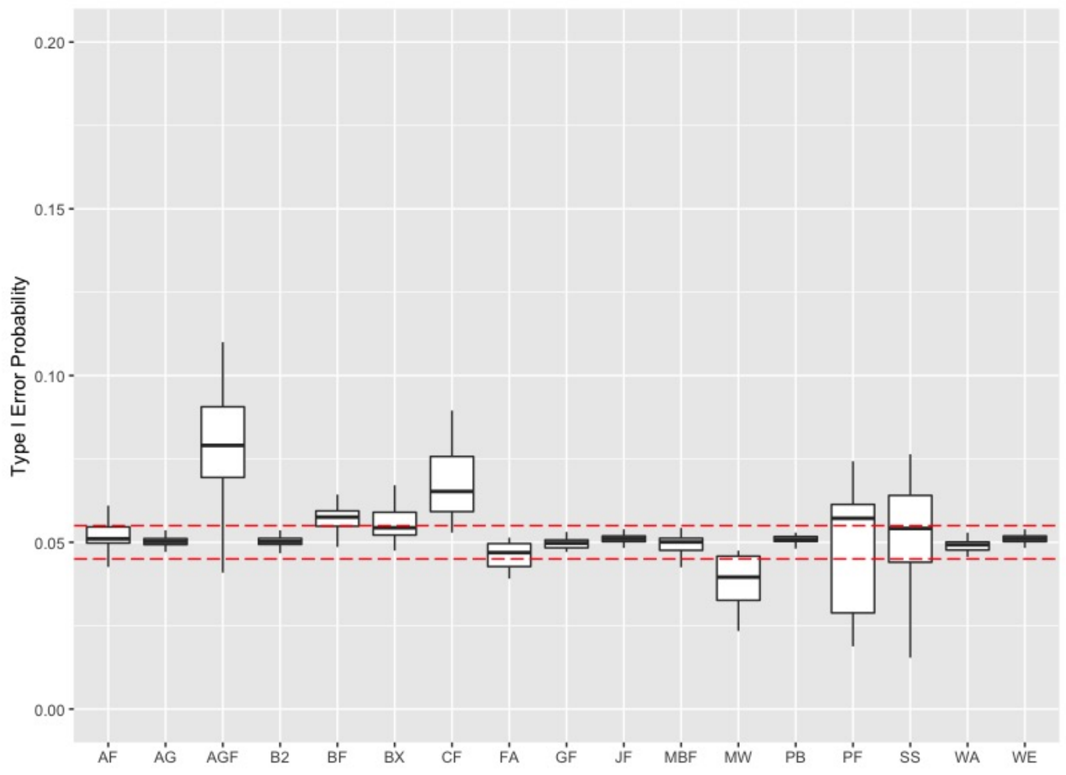


Figure 3: Type I error probability of the tests for $k = 3$.

power of the test in (35) to compare the power of the test even if Type I error probabilities are different.

$$\gamma = \frac{1-\beta}{\sqrt{1+\left|1-\frac{\alpha_i}{\alpha_0}\right|}} \tag{35}$$

where β is Type II error rate, α_i is Type I error of the test, and α_0 is the nominal level. Penalized power adjusts the power function with the square root of the percentile deviation between type I error probability and the nominal level. Thus, penalized power is used to compare the power of the tests in the simulation studies. An extensive Monte-Carlo simulation study is conducted to investigate the performance of the tests in terms of Type I error probability and penalized power. Firstly, the ability of the tests to control the Type I error probability is examined. Then, the penalized power of the test which controls the Type I error probability in the Bradley (1978)'s robustness limits are compared. In this way, we conclude the performance of the tests by taking into account two possible errors in hypothesis testing. The sample size, design type, variance heterogeneity, and effect sizes are used as configuration factors beyond this part of the study. The R code used in this simulation study is available in the following repository: `https://github.com/mcavs/doex_TheRJournal`.

The properties of the tests to control the Type I error probability

Type I error probabilities of the tests are investigated in an extensive Monte-Carlo simulation study under balanced and unbalanced design with small, moderate, and large sample sizes in this section. Also, the number of the groups is fixed as $k = 3, 5, 7$, and different heteroscedasticity setups are also used. Hereby, the properties of the tests to control the Type I error probability are revealed under various scenarios.

The boxplots in Figs. 3,4,5 are constructed for several heteroscedasticity scenarios. In this way, the ability of the tests to control the Type I error probability are obtained. According to the Fig.3, AF, AG, B2, GF, JF, MBF, PB, WA and WE test controls the Type I error probability in the Bradley (1978) limits which are shown with dashed red lines. However, the AGF, CF, PF, and SS test could not control the Type I error probability for $k = 3$. The GF test controls the Type I error probability unlike in the case of $k = 3$ in Fig.4. The AF, AG, B2, MBF, PB, and WA test control Type I error probability for $k = 7$. When the results are summarized, it is concluded that the AGF, BF, BX, CF, MW, PF, and SS test could not

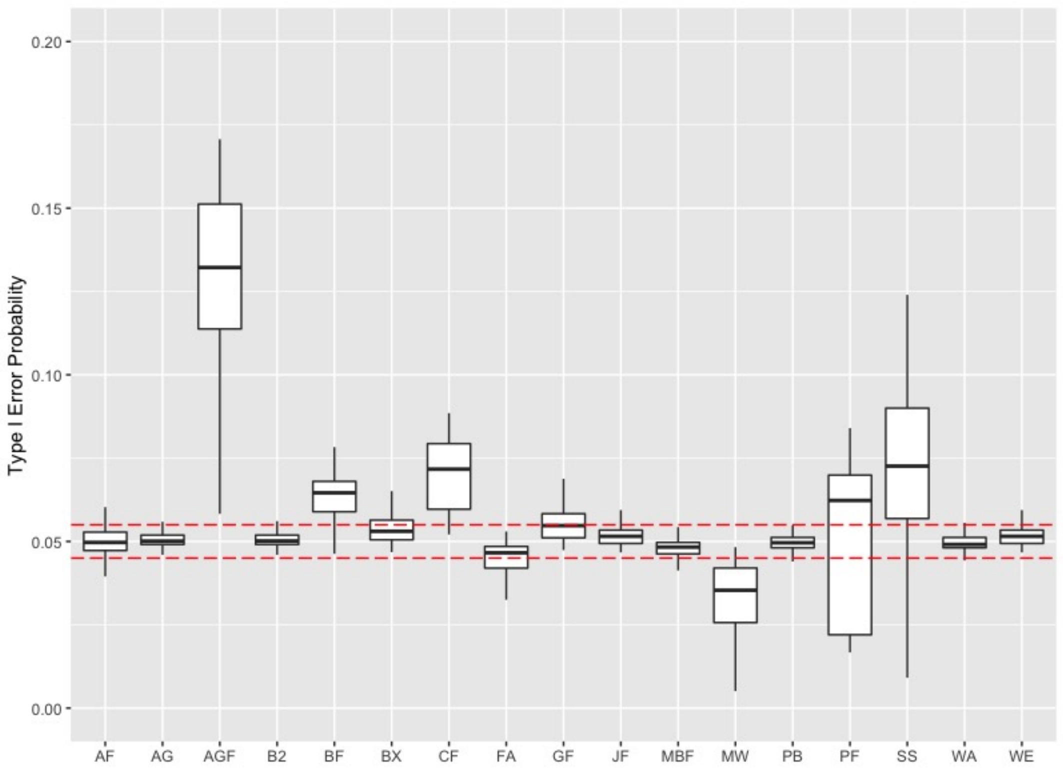


Figure 4: Type I error probability of the tests for $k = 5$.

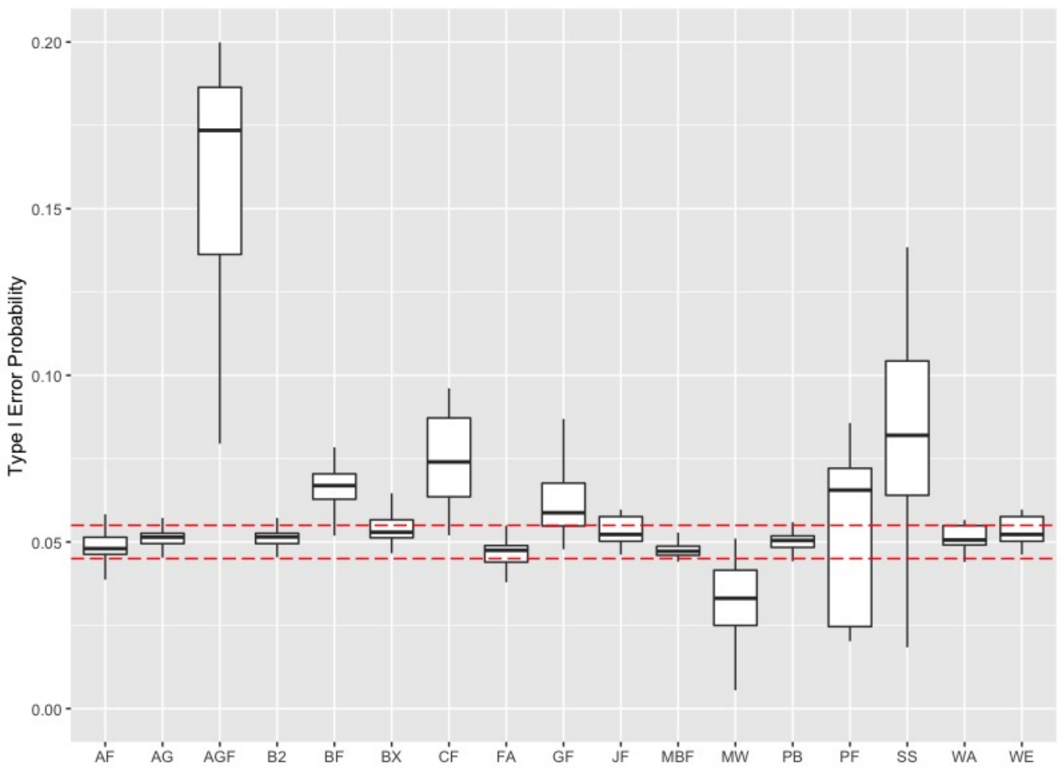


Figure 5: Type I error probability of the tests for $k = 7$.

control the Type I error probability for each of the k's. Thus, the tests only which control Type I error probability in the limits are considered in the next section for power comparisons to avoid making a wrong decision.

The results of the penalized powers

In this section, the penalized power results are given under four configuration factors are sample size, design type, effect size, heteroscedasticity level for $k = 3, 5, 7$. The samples follows normal distribution with the parameters (μ_i, σ_i^2) as given in Tables 1, 2, 3. The mean parameter of the samples are shown as the effect size Δ_i in each line. This means that the mean parameter of the samples are zero except the last sample is Δ_i.

When the effect of the configuration factors on the power are examined, it is observed that the larger sample size increase, the higher level of heteroscedasticity decrease, and the higher effect sizes increase the power of all tests as expected. Also, some interesting results are obtained such as the penalized power of all tests are higher in the unbalanced designs. The performances of the AF and MBF, the AG and B2, the JF and WE tests are very close to each other in terms of penalized power. Thus, these tests may be used interchangeably.

The AF and MBF test are superior than others in most of the scenarios. In the lower level of heteroscedasticity for all sample sizes, the penalized power of the tests are higher than 0.90 for $k = 3$. It is the same situation for $k = 5$ except for a small sample-unbalanced design. In this case, the penalized power of the AF and MGF is close to the 0.90, and the performance of the others is unacceptable. For $k = 7$, the penalized power of the tests is higher than 0.90 except for small sample-lower heteroscedasticity scenarios. In this case, the AF and MBF tests show acceptable performance in terms of penalized power in only small sample-lower level heteroscedasticity. As a result, it is clearly seen that the penalized power of the tests decreases dramatically in the higher level of heteroscedasticity for $k = 5, 7$.

Discussion

In this paper, an extensive Monte-Carlo simulation study is conducted to investigate the performance of the tests for equality of normal distributed and independent groups' means under unequal variances under several scenarios. It is rather rare to encounter normally distributed data and Bono et al. (2017) showed that the data obtained from health, educational, and social sciences research are often not normally distributed. Blanca et al. (2013) discussed the negative effect of non-normality on the power and Type I error probability of the parametric tests. It is reality that the normality assumption is crucial for the considered tests in this study. Here, it is focused on the performance of the considered tests under normality to fill the gap mentioned the introduction part. Firstly, the ability of the test to control the Type I error probability is examined and the boxplots in Figs.3, 4 and 5 are used to summarize the results. The tests which can control the Type I error probability are obtained as robust tests with respect to the Bradley (1978)'s limits. Then, the penalized power of the robust tests is calculated. The reason for using this method was to consider two possible types of error.

According to the results of the Monte-Carlo simulation study, the AF, AG, B2, MBF, PB, WA test control the Type I error probability for $k = 3, 5, 7$ in the interval $[0.0495, 0.0505]$. The GF can control only for $k = 3$ and the WE test can control only for $k = 3, 5$. Besides the controlling of the Type I error probability of these tests, the penalized power properties are also investigated under similar scenarios. The results are indicated that the AF and MBF tests are superior than others in the higher heteroscedasticity levels. Also, it is concluded that the penalized power of the other tests is quite close to the intended level.

As a result of this study, the robust tests are obtained and can be used in most of the situations except for a higher level of heteroscedasticity and small sample sizes. Using the results of the simulation study, researchers can use appropriate tests for their studies.

Summary and Future works

The **doex** package contains the several tests for testing equality of normally distributed groups' means under unequal variances. Most of these tests are not available in any R package. Thus, we fill this gap by implementing the package in the statistical software literature. The fact that the package contains tests such as the GF, PB, and FA with complex calculation steps provides a significant benefit to multidisciplinary researchers. Furthermore, the performance of the considered tests is investigated under normal distributions in detail in an extensive Monte-Carlo simulation study. Considering the number of methods discussed, this article is the most comprehensive performance investigation study in the literature. Recommendations were made to the researchers by using the interesting outputs from the simulation study.

It is always optimistic idea to encountered normal distribution in real life. The performance of the considered tests can be also investigated under the various distributions or to focus the tests are proposed for non-normal distributions. Thus, it is planned to expand the package by adding methods used to test the equality of the log-normal (Tian and Wu, 2007) and inverse-Gaussian (Tian, 2006; Ma and Tian, 2009) distributed and independent groups' means in further studies.

Acknowledgement

This study is supported by the Eskisehir Technical University Scientific Research Projects Commission under grant No. 20DRP047.

Bibliography

K. Aho. *asbio: A Collection of Statistical Tools for Biologists*, 2018. URL https://CRAN.R-project.org/package=asbio. R package version 1.5-3. [p190]

R. A. Alexander and D. M. Govern. A new and simplier approximation for anova under variance heterogeneity. *Journal of Educational Statistics*, 19(2):91–101, 1994. URL https://www.jstor.org/stable/1165140. [p190]

D. Argac. Testing for homogeneity in a general one-way classification with fixed effects: power simulations and comparative study. *Computational Statistics & Data Analysis*, 44:603–612, 2004. URL https://doi.org/10.1016/S0167-9473(02)00264-5. [p189]

O. Asiribo and J. Gurland. Coping with variance heterogeneity. *Communications in Statistics-Theory and Methods*, 19(11):4029–4048, 1990. URL https://www.tandfonline.com/doi/abs/10.1080/03610929008830427. [p191]

A. A. Aspin. Interval estimates for linear combinations of means. *Biometrika*, 35(1):276–285, 1948. URL https://www.jstor.org/stable/2332631. [p195]

K. J. Berry and P. W. Mielke. The fisher-pitman permutation test: an attractive alternative to the f test. *Psychological Reports*, 90(2):495–502, 2002. URL https://www.ncbi.nlm.nih.gov/pubmed/12061589. [p195]

M. Blanca, J. Arnau, D. Lopez-Montiel, R. Bono, and R. Bendayan. Skewness and kurtosis in real data samples. *Methodology*, 9:78–84, 2013. [p202]

R. Bono, M. Blanca, J. Arnau, and J. Gomez-Benito. Non-normal distributions commonly used in health, education, and social sciences: A systematic review. *Frontiers in Psychology*, 8, 2017. [p202]

G. E. P. Box. Some theorems on quadratic forms applied in the study of analysis of variance problems. *Technometrics*, 25(2):290–302, 1954. URL https://www.jstor.org/stable/pdf/2236731.pdf. [p192]

J. V. Bradley. Robustness. *British Journal of Mathematical and Statistical Psychology*, 31:144–152, 1978. [p200, 202]

M. B. Brown and A. B. Forsythe. The small sample behavior of some statistics which test the equality of several means. *Technometrics*, 16(1):81–90, 1974. URL https://www.jstor.org/stable/pdf/1267501.pdf. [p192]

M. Cavus, B. Yazici, and A. Sezer. Modified tests for comparison of group means under heteroskedasticity and non-normality caused by outliers. *Hacettepe Journal of Mathematics and Statistics*, 46 (3):493–510, 2017. URL http://www.hjms.hacettepe.edu.tr/uploads/f0746d40-4b55-41fb-9917-12f8f2f0a2e4.pdf. [p190]

M. Cavus, B. Yazici, and A. Sezer. Analysing regional export data by the modified generalized f-test. *International Journal of Economic and Administrative Studies*, pages 541–551, 2018. URL https://dergipark.org.tr/download/article-file/408447. [p190]

M. Cavus, B. Yazici, and A. Sezer. Penalized power approach to compare the power of the tests when type i error probabilities are different. *Communication in Statistics-Simulation and Computation*, 2019. URL https://doi.org/10.1080/03610918.2019.1588310. [p199]

W. G. Cochran. Problems arising in the analysis of a series of similar experiments. *Technometrics*, 4(1): 290–302, 1937. URL https://www.jstor.org/stable/pdf/2984123.pdf. [p193]

O. Dag, A. Dolgun, and N. M. Konar. onewaytests: an r package for one-way tests in independent groups designs. *R Journal*, 10(1):175–199, 2018. URL https://journal.r-project.org/archive/2018/RJ-2018-022/RJ-2018-022.pdf. [p190]

R. C. Erps and K. Noguchi. A robust test for checking the homogeneity of variability measures and its application to the analysis of implicit attitudes. *Journal of Educational and Behavioral Statistics*, 45(4): 403–425, 2019. [p195]

J. Fox and S. Weisberg. *An R companion to applied regression*. Sage, Thousand Oaks CA., third edition edition, 2019. [p195]

J. Gamage and S. Weerahandi. Size performance of some tests in one-way anova. *Communications in Statistics-Computation and Simulation*, 27(3):165–173, 1998. URL http://dx.doi.org/10.1080/03610919808813500. [p189]

J. L. Gastwirth, Y. R. Gel, W. L. W. Hui, V. Lyubchich, W. Miao, and K. Noguchi. *lawstat: Tools for Biostatistics, Public Policy, and Law*, 2020. URL https://CRAN.R-project.org/package=lawstat. R package version 3.4. [p195]

E. Gokpinar and F. Gokpinar. A test based on the computational approach for equality of means under the unequal variance assumption. *Hacettepe Journal of Mathematics and Statistics*, 41(4): 605–613, 2012. URL http://www.hjms.hacettepe.edu.tr/uploads/0871d39b-039b-45fb-9b31-483cdde56ef7.pdf. [p190]

J. Hartung, D. Argac, and K. H. Makambi. Small sample properties of tests on homogeneity in one-way anova and meta-analysis. *Statistical Papers*, 43(2):1139–1145, 2002. URL https://link.springer.com/article/10.1007/s00362-002-0097-8. [p189, 190, 194]

A. Hebbali. *inferr: Inferential Statistics*, 2018. URL https://CRAN.R-project.org/package=inferr. R package version 0.3.0. [p195]

T. Hothorn, K. Hornik, M. A. van de Wiel, and A. Zeileis. lawstat: an r package for law, public policy and biostatistics. *Journal of Statistical Software*, 28(8):1–23, 2008. URL https://doi.org/10.18637/jss.v028.i08. [p190]

W. Hui, Y. R. Gel, and G. G. Gastwirth. Implementing a class of permutation tests: The coin package. *Journal of Statistical Software*, 28(3):1–26, 2008. URL https://doi.org/10.18637/jss.v028.i03. [p190]

S. Johansen. The welch-james approximation to the distribution of the residual sum of squares in a weighted linear regression. *Biometrika*, 67(1):58–92, 1980. URL https://www.jstor.org/stable/2335320. [p193]

A. Kassambara. *rstatix: Pipe-Friendly Framework for Basic Statistical Tests*, 2020. URL https://CRAN.R-project.org/package=rstatix. R package version 0.6.0. [p195]

K. Krishnamoorthy, F. Lu, and T. Mathew. A parametric bootstrap approach for anova with unequal variances: fixed and random models. *Computational Statistics and Data Analysis*, 51(12):5731–5742, 2007. URL https://www.sciencedirect.com/science/article/pii/S016794730600363X. [p194]

S. Lee and C. H. Ahn. Modified anova for unequal variances. *Communication in Statistics-Simulation and Computation*, 32(4):987–1004, 2003. [p190]

X. Li, J. Wang, and H. Liang. Comparison of several means: a fiducial based approach. *Computational Statistics and Data Analysis*, 55(5):1993–2002, 2011. URL https://www.sciencedirect.com/science/article/pii/S0167947310004779. [p190, 193]

C. J. Lloyd. Estimating test power adjusted for size. *Journal of Statistical Computation and Simulation*, 75 (11):921–933, 2005. URL https://www.tandfonline.com/doi/abs/10.1080/00949650412331321160. [p199]

C. X. Ma and L. Tian. A parametric bootstrap approach for testing equality of inverse gaussian means under heterogeneity. *Communications in Statistics-Simulation and Computation*, 38:1153–1160, 2009. [p203]

P. Mair and R. Wilcox. *WRS2: Wilcox robust estimation and testing*, 2018. 0.10-0. [p190]

D. V. Mehrotra. Improving the brown-forsythe solution to the generalized behrens-fisher problem. *Communication in Statistics-Simulation and Computation*, 26(3):1139–1145, 1997. URL https://www.tandfonline.com/doi/abs/10.1080/03610919708813431. [p194]

H. T. Mutlu, F. Gokpinar, E. Gokpinar, H. H. Gul, and G. Guven. A new computational approach test for one-way anova under heteroscedasticity. *Communications in Statistics - Theory and Methods*, 46 (16):8236–8256, 2017. [p190]

K. Noguchi and Y. R. Gel. Combination of levene-type tests and a finite-intersection method for testing equality of variances against ordered alternatives. *Journal of Nonparametric Statistics*, 22(7):897–913, 2010. [p195]

A. F. Ozdemir and S. Kurt. One-way fixed effect analysis of variance under variance heterogeneity and a solution proposal. *Selcuk Journal of Applied Mathematics*, 7(2):81–90, 2006. URL http://sjam.selcuk.edu.tr/sjam/article/view/174. [p192]

S. M. Sadooghi-Alvandi, A. A. Jafari, and H. A. Mardani-Fard. One-way anova with unequal variances. *Communications in Statistics-Theory and Methods*, 41(22):4200–4221, 2012. URL https://www.tandfonline.com/doi/full/10.1080/03610926.2011.573160. [p189, 191]

A. J. Scott and T. M. Smith. Interval estimates for linear combinations of means. *Journal of the Royal Statistical Society*, 20(3):276–285, 1971. URL https://www.jstor.org/stable/2346757. [p195]

L. Tian. Testing equality of inverse gaussian means under heterogeneity based on generalized test variable. *Computational Statistics Data Analysis*, 51:1156:1162, 2006. [p203]

L. Tian and J. Wu. Inferences on the common mean of several log-normal populations: the generalized variable approach. *Biometrical Journal*, 49:944:951, 2007. [p203]

P. J. Villacorta. The welchadf package for robust hypothesis testing in unbalanced multivariate mixed models with heteroscedastic an non-normal data. *The R Journal*, 9(2):309–328, 2017. URL https://journal.r-project.org/archive/2017/RJ-2017-049/RJ-2017-049.pdf. [p190]

S. Weerahandi. Anova under unequal error variances. *Biometrics*, 51(2):589–599, 1995. URL http://www.jstor.org/stable/2532947. [p193]

B. L. Welch. On the comparison of several mean values. *Biometrika*, 38(1):330–336, 1951. URL https://www.jstor.org/stable/2332631. [p195]

J. Zhang and D. D. Boos. Adjusted power estimates in monte-carlo experiments. *Communications in Statistics-Computation and Simulation*, 23(1):165–173, 1994. URL https://www.tandfonline.com/doi/abs/10.1080/00949650412331321160. [p199]

Mustafa Cavus
Eskisehir Technical University
Department of Statistics
Eskisehir, Turkey
https://orcid.org/0000-0002-6172-5449
mustafacavus@eskisehir.edu.tr

Berna Yazıcı
Eskisehir Technical University
Department of Statistics
Eskisehir, Turkey
https://orcid.org/0000-0001-9843-7355
bbaloglu@eskisehir.edu.tr

Table 1: Penalized powers for $k = 3$

n_i	σ_i^2	Δ_i	AG	AF	B2	GF	JF	MBF	PB	WE	WA
(10, 10, 10)	(0.1, 0.2, 0.3)	0.3	0.2232	0.2630	0.2236	0.2133	0.2314	0.2628	0.2306	0.2314	0.2128
		0.8	0.9045	0.9391	0.9047	0.8915	0.9197	0.9390	0.9201	0.9197	0.8833
		1.5	0.9910	0.9815	0.9911	0.9768	0.9921	0.9815	0.9941	0.9921	0.9614
	(0.1, 0.4, 0.7)	0.3	0.1265	0.1491	0.1269	0.1277	0.1326	0.1490	0.1296	0.1326	0.1193
		0.8	0.5962	0.6858	0.5978	0.6004	0.6129	0.6855	0.6089	0.6129	0.5829
		1.5	0.9853	0.9880	0.9881	0.9893	0.9812	0.9879	0.9777	0.9812	0.9583
	(1, 2, 3)	0.3	0.0682	0.0656	0.0684	0.0649	0.0708	0.0655	0.0702	0.0708	0.0644
		0.8	0.1700	0.2006	0.1701	0.1653	0.1760	0.2004	0.1752	0.1760	0.1615
		1.5	0.4832	0.5536	0.4833	0.4716	0.4998	0.5534	0.4968	0.4998	0.4693
	(1, 4, 7)	0.3	0.0614	0.0566	0.0616	0.0622	0.0627	0.0565	0.0622	0.0627	0.0563
		0.8	0.1023	0.1200	0.1026	0.1050	0.1075	0.1200	0.1068	0.1075	0.0958
		1.5	0.2542	0.3022	0.2550	0.2556	0.2655	0.3021	0.2594	0.2655	0.2424
(30, 30, 30)	(0.1, 0.2, 0.3)	0.3	0.6299	0.7004	0.6293	0.6281	0.6327	0.7002	0.6342	0.6327	0.6375
		0.8	0.9882	0.9980	0.9872	0.9872	0.9815	0.9979	0.9853	0.9815	0.9970
		1.5	1	1	1	1	1	1	1	1	1
	(0.1, 0.4, 0.7)	0.3	0.3210	0.3791	0.3212	0.3210	0.3243	0.3789	0.3224	0.3243	0.3233
		0.8	0.9772	0.9924	0.9775	0.9779	0.9740	0.9922	0.9726	0.9740	0.9898
		1.5	0.9831	0.9960	0.9834	0.9834	0.9796	0.9960	0.9787	0.9796	0.9960
	(1, 2, 3)	0.3	0.0923	0.1072	0.0922	0.0934	0.0938	0.1070	0.0930	0.0938	0.0925
		0.8	0.4785	0.5411	0.4780	0.4733	0.4823	0.5410	0.4789	0.4823	0.4820
		1.5	0.9566	0.9798	0.9557	0.9545	0.9517	0.9797	0.9566	0.9517	0.9653
	(1, 4, 7)	0.3	0.0661	0.0715	0.0663	0.0663	0.0672	0.0714	0.0648	0.0672	0.0657
		0.8	0.2282	0.2809	0.2285	0.2284	0.2318	0.2808	0.2298	0.2318	0.2301
		1.5	0.6982	0.7659	0.6990	0.7012	0.7032	0.7655	0.7009	0.7032	0.7100
(50, 50, 50)	(0.1, 0.2, 0.3)	0.3	0.8573	0.9046	0.8577	0.8546	0.8547	0.9045	0.8604	0.8547	0.8593
		0.8	0.9811	0.9980	0.9815	0.9825	0.9759	0.9979	0.9853	0.9759	0.9834
		1.5	1	1	1	1	1	1	1	1	1
	(0.1, 0.4, 0.7)	0.3	0.5427	0.6028	0.5425	0.5443	0.5418	0.6026	0.5400	0.5418	0.5442
		0.8	0.9832	0.9980	0.9834	0.9880	0.9759	0.9980	0.9740	0.9759	0.9872
		1.5	1	1	1	1	0.9979	1	0.9940	0.9981	1
	(1, 2, 3)	0.3	0.1421	0.1671	0.1431	0.1432	0.1435	0.1671	0.1474	0.1435	0.1434
		0.8	0.7210	0.7900	0.7214	0.7203	0.7204	0.7900	0.7262	0.7204	0.7232
		1.5	0.9803	0.9974	0.9807	0.9817	0.9751	0.9974	0.9843	0.9751	0.9826
	(1, 4, 7)	0.3	0.0912	0.1040	0.0917	0.0907	0.0917	0.1040	0.0918	0.0917	0.0904
		0.8	0.4020	0.4659	0.4018	0.4066	0.4017	0.4659	0.4021	0.4017	0.4032
		1.5	0.9031	0.9405	0.9032	0.9070	0.8990	0.9405	0.8932	0.8990	0.9077
(5, 10, 15)	(0.1, 0.2, 0.3)	0.3	0.2790	0.3346	0.2791	0.2535	0.2573	0.2974	0.2605	0.2573	0.2371
		0.8	0.9598	0.9388	0.9570	0.9217	0.9432	0.9342	0.9484	0.9432	0.9406
		1.5	0.9844	0.9483	0.9815	0.9475	0.9731	0.9500	0.9787	0.9731	0.9759
	(0.1, 0.4, 0.7)	0.3	0.1522	0.1955	0.1526	0.1346	0.1494	0.1743	0.1490	0.1494	0.1339
		0.8	0.7601	0.8070	0.7609	0.7018	0.7535	0.7860	0.7532	0.7535	0.7063
		1.5	0.9910	0.9612	0.9913	0.9340	0.9962	0.9748	0.9992	0.9962	0.9567
	(1, 2, 3)	0.3	0.0702	0.0835	0.0709	0.0604	0.0650	0.0669	0.0652	0.0650	0.0607
		0.8	0.2018	0.2644	0.2014	0.1817	0.1900	0.2236	0.1912	0.1900	0.1741
		1.5	0.5940	0.6651	0.5928	0.5571	0.5611	0.6205	0.5649	0.5611	0.5412
	(1, 4, 7)	0.3	0.0562	0.0690	0.0567	0.0460	0.0568	0.0610	0.0580	0.0568	0.0496
		0.8	0.1202	0.1546	0.1210	0.1068	0.1190	0.1353	0.1190	0.1190	0.1036
		1.5	0.3310	0.3986	0.3312	0.3016	0.3252	0.3689	0.3250	0.3252	0.2971
(20, 30, 40)	(0.1, 0.2, 0.3)	0.3	0.7142	0.7419	0.7143	0.7108	0.7071	0.7609	0.7093	0.7071	0.7068
		0.8	0.9955	0.9509	0.9960	0.9980	0.9872	0.9970	0.9921	0.9872	0.9960
		1.5	1	1	1	1	1	1	1	1	1
	(0.1, 0.4, 0.7)	0.3	0.3951	0.4407	0.3953	0.3944	0.3994	0.4314	0.3953	0.3994	0.3872
		0.8	0.9892	0.9612	0.9895	0.9936	0.9984	0.9869	0.9899	0.9984	0.9857
		1.5	0.9910	0.9614	0.9911	0.9950	1	0.9872	0.9911	1	0.9872
	(1, 2, 3)	0.3	0.1062	0.1227	0.1066	0.1022	0.1060	0.1196	0.1058	0.1060	0.1044
		0.8	0.5572	0.5975	0.5576	0.5541	0.5521	0.6064	0.5540	0.5521	0.5530
		1.5	0.9843	0.9452	0.9847	0.9860	0.9758	0.9892	0.9806	0.9758	0.9845
	(1, 4, 7)	0.3	0.0743	0.0821	0.0747	0.0732	0.0756	0.0772	0.0747	0.0756	0.0735
		0.8	0.2872	0.3234	0.2874	0.2876	0.2904	0.3157	0.2908	0.2904	0.2843
		1.5	0.8011	0.8166	0.8020	0.8044	0.8100	0.8240	0.8004	0.8100	0.7957
(25, 50, 75)	(0.1, 0.2, 0.3)	0.3	0.9182	0.8505	0.9184	0.9294	0.9129	0.9361	0.9153	0.9129	0.9227
		0.8	0.9744	0.8811	0.9750	0.9872	0.9704	0.9796	0.9750	0.9704	0.9825
		1.5	1	1	1	1	1	1	1	1	1
	(0.1, 0.4, 0.7)	0.3	0.6392	0.6680	0.6398	0.6375	0.6365	0.6702	0.6384	0.6365	0.6380
		0.8	0.9682	0.9054	0.9685	0.9704	0.9649	0.9631	0.9750	0.9649	0.9722
		1.5	1	0.9654	1	1	1	1	1	1	1
	(1, 2, 3)	0.3	0.1631	0.2058	0.1636	0.1649	0.1630	0.1932	0.1652	0.1630	0.1627
		0.8	0.8022	0.7839	0.8030	0.8105	0.7976	0.8452	0.7991	0.7976	0.8058
		1.5	0.9745	0.8811	0.9750	0.9872	0.9704	0.9796	0.9750	0.9704	0.9825
	(1, 4, 7)	0.3	0.0970	0.1217	0.0972	0.0959	0.0969	0.1094	0.0989	0.0969	0.0961
		0.8	0.4922	0.5291	0.4924	0.4896	0.4902	0.5205	0.4941	0.4902	0.4900
		1.5	0.9466	0.8923	0.9470	0.9486	0.9433	0.9435	0.9531	0.9433	0.9500

Table 2: Penalized powers for $k = 5$

n_i	σ_i^2	Δ_i	AG	AF	B2	JF	MBF	PB	WE	WA
(10, 10, 10, 10, 10)	(0.1, 0.2, 0.3, 0.4, 0.5)	0.3	0.0713	0.0867	0.0716	0.0779	0.0863	0.0729	0.0779	0.0718
		0.8	0.2172	0.3403	0.2173	0.2424	0.3400	0.2350	0.2424	0.2271
		1.5	0.6316	0.8549	0.6315	0.6984	0.8542	0.6897	0.6984	0.6722
	(0.1, 0.4, 0.7, 1.1, 1.5)	0.3	0.0562	0.0586	0.0565	0.0543	0.0536	0.0540	0.0543	0.0515
		0.8	0.0917	0.1191	0.0915	0.0848	0.1074	0.0843	0.0848	0.0791
		1.5	0.2110	0.3094	0.2109	0.1840	0.2835	0.1850	0.1840	0.1747
	(1, 2, 3, 4, 5)	0.3	0.0522	0.0541	0.0523	0.0560	0.0540	0.0525	0.0560	0.0502
		0.8	0.0658	0.0788	0.0657	0.0713	0.0785	0.0671	0.0713	0.0647
		1.5	0.1066	0.5250	0.1065	0.1165	0.1479	0.1072	0.1165	0.1046
	(1, 4, 7, 11, 15)	0.3	0.0953	0.1154	0.0951	0.0923	0.1153	0.0964	0.0923	0.0940
		0.8	0.4644	0.5647	0.4635	0.4510	0.5645	0.4635	0.4510	0.4611
		1.5	0.9652	0.9788	0.9633	0.9312	0.9785	0.9611	0.9312	0.9563
(30, 30, 30, 30, 30)	(0.1, 0.2, 0.3, 0.4, 0.5)	0.3	0.3557	0.4704	0.3556	0.3598	0.4700	0.3573	0.3598	0.3601
		0.8	0.9824	0.9871	0.9824	0.9666	0.9870	0.9716	0.9666	0.9803
		1.5	0.9834	0.9872	0.9844	0.9676	0.9871	0.9731	0.9676	0.9815
	(0.1, 0.4, 0.7, 1.1, 1.5)	0.3	0.1442	0.1996	0.1443	0.1461	0.1994	0.1445	0.1461	0.1456
		0.8	0.7418	0.8700	0.7420	0.7478	0.8700	0.7439	0.7478	0.7586
		1.5	0.9843	0.9931	0.9848	0.9708	0.9930	0.9700	0.9708	0.9906
	(1, 2, 3, 4, 5)	0.3	0.0773	0.0898	0.0779	0.0791	0.0893	0.0784	0.0791	0.0768
		0.8	0.2581	0.3510	0.2590	0.2618	0.3508	0.2606	0.2618	0.2614
		1.5	0.7472	0.8595	0.7477	0.7509	0.8593	0.7537	0.7509	0.7579
	(1, 4, 7, 11, 15)	0.3	0.0621	0.0666	0.0622	0.0621	0.0664	0.0624	0.0621	0.0614
		0.8	0.1158	0.1557	0.1160	0.1167	0.1552	0.1164	0.1167	0.1159
		1.5	0.3113	0.4370	0.3112	0.3182	0.4366	0.3153	0.3182	0.3193
(50, 50, 50, 50, 50)	(0.1, 0.2, 0.3, 0.4, 0.5)	0.3	0.5882	0.6978	0.5883	0.5996	0.6975	0.5964	0.5996	0.5881
		0.8	0.9955	0.9825	0.9960	1	0.9823	0.9990	1	0.9901
		1.5	1	1	1	1	1	1	1	1
	(0.1, 0.4, 0.7, 1.1, 1.5)	0.3	0.2161	0.3051	0.2163	0.2215	0.3050	0.2178	0.2215	0.2152
		0.8	0.9432	0.9636	0.9433	0.9472	0.9633	0.9440	0.9472	0.9321
		1.5	0.9890	0.9787	0.9892	0.9901	0.9785	0.9872	0.9901	0.9750
	(1, 2, 3, 4, 5)	0.3	0.0912	0.1121	0.0919	0.0929	0.1120	0.0921	0.0929	0.0909
		0.8	0.4361	0.5477	0.4363	0.4465	0.5474	0.4435	0.4465	0.4374
		1.5	0.9502	0.9665	0.9508	0.9574	0.9663	0.9558	0.9574	0.9473
	(1, 4, 7, 11, 15)	0.3	0.0621	0.0701	0.0628	0.0626	0.0700	0.0621	0.0626	0.0610
		0.8	0.1617	0.2248	0.1619	0.1657	0.2245	0.1624	0.1657	0.1602
		1.5	0.5205	0.6456	0.5206	0.5292	0.6455	0.5302	0.5292	0.5197
(4, 6, 10, 14, 16)	(0.1, 0.2, 0.3, 0.4, 0.5)	0.3	0.1680	0.2295	0.1682	0.1448	0.1922	0.1309	0.1448	0.1313
		0.8	0.8345	0.9315	0.8335	0.7925	0.8584	0.7690	0.7925	0.7730
		1.5	0.9833	0.9844	0.9814	0.9795	0.9198	0.9739	0.9795	0.9758
	(0.1, 0.4, 0.7, 1.1, 1.5)	0.3	0.0848	0.1110	0.0847	0.0790	0.0988	0.0745	0.0790	0.0736
		0.8	0.3790	0.5455	0.3790	0.3366	0.5029	0.3283	0.3366	0.3282
		1.5	0.9103	0.9756	0.9072	0.8771	0.9474	0.8920	0.8771	0.8947
	(1, 2, 3, 4, 5)	0.3	0.0633	0.0625	0.0632	0.0605	0.0501	0.0583	0.0605	0.0570
		0.8	0.1252	0.1750	0.1253	0.1103	0.1469	0.1081	0.1103	0.1069
		1.5	0.3671	0.5264	0.3674	0.3126	0.4614	0.3090	0.3126	0.3093
	(1, 4, 7, 11, 15)	0.3	0.0523	0.0537	0.0525	0.0544	0.0471	0.0479	0.0544	0.0474
		0.8	0.0727	0.0902	0.0735	0.0707	0.0791	0.0635	0.0707	0.0630
		1.5	0.1466	0.2189	0.1475	0.1359	0.1930	0.1195	0.1359	0.1197
(12,18, 30, 42, 48)	(0.1, 0.2, 0.3, 0.4, 0.5)	0.3	0.5134	0.6239	0.5129	0.4915	0.6184	0.4904	0.4915	0.4905
		0.8	0.9969	0.9466	0.9959	0.9824	0.9863	0.9910	0.9824	0.9959
		1.5	1	1	1	1	1	1	1	1
	(0.1, 0.4, 0.7, 1.1, 1.5)	0.3	0.1999	0.2816	0.2000	0.1936	0.2697	0.1906	0.1936	0.1904
		0.8	0.9238	0.9291	0.9239	0.9091	0.9556	0.9046	0.9091	0.9183
		1.5	0.9977	0.9543	0.9980	0.9853	0.9882	0.9815	0.9853	0.9980
	(1, 2, 3, 4, 5)	0.3	0.0812	0.1162	0.0814	0.0817	0.1028	0.0804	0.0817	0.0772
		0.8	0.3714	0.4900	0.3713	0.3630	0.4614	0.3567	0.3630	0.3459
		1.5	0.9016	0.9321	0.9017	0.9085	0.9309	0.8963	0.9085	0.8825
	(1, 4, 7, 11, 15)	0.3	0.0611	0.0718	0.0618	0.0615	0.0670	0.0605	0.0615	0.0607
		0.8	0.1502	0.2101	0.1506	0.1445	0.2032	0.1460	0.1445	0.1443
		1.5	0.4561	0.5788	0.4561	0.4425	0.5779	0.4465	0.4425	0.4472
(20, 30, 50, 70, 80)	(0.1, 0.2, 0.3, 0.4, 0.5)	0.3	0.7441	0.8321	0.7448	0.7426	0.8292	0.7403	0.7426	0.7293
		0.8	0.9692	0.9509	0.9695	0.9759	0.9722	0.9731	0.9759	0.9614
		1.5	0.9891	0.9729	0.9895	0.9959	0.9822	0.9921	0.9899	0.9934
	(0.1, 0.4, 0.7, 1.1, 1.5)	0.3	0.7590	0.8306	0.7593	0.7602	0.8469	0.7569	0.7602	0.7499
		0.8	0.9782	0.9444	0.9785	0.9844	0.9874	0.9828	0.9844	0.9747
		1.5	0.9821	0.9449	0.9825	0.9882	0.9882	0.9872	0.9882	0.9787
	(1, 2, 3, 4, 5)	0.3	0.1040	0.1441	0.1047	0.1027	0.1374	0.1018	0.1027	0.1008
		0.8	0.5771	0.6735	0.5773	0.5709	0.7017	0.5684	0.5709	0.5719
		1.5	0.9472	0.9101	0.9475	0.9473	0.9861	0.9499	0.9473	0.9550
	(1, 4, 7, 11, 15)	0.3	0.0651	0.0805	0.0657	0.0654	0.0765	0.0652	0.0654	0.0643
		0.8	0.2322	0.3155	0.2326	0.2275	0.3055	0.2278	0.2275	0.2250
		1.5	0.7033	0.7689	0.7038	0.6962	0.7887	0.6987	0.6962	0.6995

Table 3: Penalized powers for $k = 7$

n_i	σ_i^2	Δ_i	AG	AF	B2	MBF	PB	WA
(10, 10, 10, 10, 10, 10, 10)	(0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7)	0.3	0.0916	0.1176	0.0921	0.1175	0.0954	0.0960
		0.8	0.3745	0.6158	0.3750	0.6154	0.4288	0.4279
		1.5	0.8713	0.9725	0.8715	0.9723	0.9247	0.9068
	(0.1, 0.4, 0.7, 1.1, 1.5, 1.9, 2.3)	0.3	0.0668	0.0655	0.0666	0,0651	0.0660	0.0658
		0.8	0.1501	0.2255	0.1502	0.2253	0.1578	0.1603
		1.5	0.4021	0.6756	0.4023	0.6753	0.4519	0.4519
	(1, 2, 3, 4, 5, 6, 7)	0.3	0.0565	0.0520	0.0568	0.0520	0.0597	0.0588
		0.8	0.0819	0.0944	0.0820	0.0942	0.0831	0.0834
		1.5	0.1612	0.2385	0.1609	0.2381	0.1693	0.1695
	(1, 4, 7, 11, 15, 19, 23)	0.3	0.0560	0.0485	0.0561	0.0483	0.0577	0.0570
		0.8	0.0632	0.0608	0.0635	0.0604	0.0625	0.0633
		1.5	0.0855	0.1044	0.0858	0.1042	0.0860	0.0880
(30, 30, 30, 30, 30, 30, 30)	(0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7)	0.3	0.2204	0.3303	0.2209	0.3301	0.2301	0.2298
		0.8	0.9598	0.9703	0.9596	0.9701	0.9616	0.9640
		1.5	0.9957	0.9778	0.9960	0.9768	0.9911	0.9941
	(0.1, 0.4, 0.7, 1.1, 1.5, 1.9, 2.3)	0.3	0.0978	0.1334	0.0980	0.1324	0.0977	0.0985
		0.8	0.4679	0.6533	0.4680	0.6530	0.4867	0.4904
		1.5	0.9702	0.9822	0.9704	0.9820	0.9630	0.9757
	(1, 2, 3, 4, 5, 6, 7)	0.3	0.0651	0.0712	0.0657	0.0711	0.0670	0.0636
		0.8	0.1652	0.2417	0.1657	0.2414	0.1725	0.1706
		1.5	0.5243	0.6901	0.5247	0.6900	0.5453	0.5461
	(1, 4, 7, 11, 15, 19, 23)	0.3	0.0552	0.0575	0.0555	0.0572	0.0550	0.0537
		0.8	0.0852	0.1060	0.0857	0.1050	0.0853	0.0860
		1.5	0.1810	0.2765	0.1813	0.2755	0.1841	0.1839
(50, 50, 50, 50, 50, 50, 50)	(0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7)	0.3	0.3722	0.5118	0.3725	0.5111	0.3782	0.3787
		0.8	0.9969	0.9853	0.9978	0.9850	0.9988	0.9948
		1.5	1	1	1	1	1	1
	(0.1, 0.4, 0.7, 1.1, 1.5, 1.9, 2.3)	0.3	0.1362	0.1907	0.1365	0.1907	0.1387	0.1380
		0.8	0.7502	0.8843	0.7507	0.8840	0.7686	0.7657
		1.5	0.9943	0.9863	0.9948	0.9852	0.9988	0.9968
	(1, 2, 3, 4, 5, 6, 7)	0.3	0.0762	0.0914	0.0764	0.0910	0.0787	0.0758
		0.8	0.2660	0.3817	0.2661	0.3812	0.2739	0.2697
		1.5	0.8052	0.9063	0.8058	0.9051	0.8202	0.8145
	(1, 4, 7, 11, 15, 19, 23)	0.3	0.0600	0.0629	0.0601	0.0612	0.0605	0.0594
		0.8	0.1093	0.1499	0.1099	0.1479	0.1123	0.1099
		1.5	0.2902	0.4275	0.2906	0.4271	0.2981	0.2971
(4, 6, 8, 10, 12, 14, 16)	(0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7)	0.3	0.1201	0.1540	0.1208	0.1419	0.1025	0.1067
		0.8	0.6065	0.7846	0.6068	0.7525	0.5561	0.5596
		1.5	0.9622	0.9456	0.9617	0.9273	0.9852	0.9544
	(0.1, 0.4, 0.7, 1.1, 1.5, 1.9, 2.3)	0.3	0.0718	0.0814	0.0720	0.0750	0.0629	0.0704
		0.8	0.2169	0.3329	0.2173	0.3132	0.1798	0.1944
		1.5	0.6726	0.8591	0.6734	0.8274	0.6240	0.6532
	(1, 2, 3, 4, 5, 6, 7)	0.3	0.0622	0.0549	0.0621	0.0509	0.0587	0.0609
		0.8	0.1035	0.1204	0.1041	0.1114	0.0856	0.0900
		1.5	0.2357	0.3452	0.2356	0.3250	0.1935	0.2023
	(1, 4, 7, 11, 15, 19, 23)	0.3	0.0561	0.0497	0.0564	0.0481	0.0518	0.0563
		0.8	0.0653	0.0714	0.0654	0.0677	0.0607	0.0641
		1.5	0.1078	0.1385	0.1080	0.1305	0.0939	0.0947
(12, 18, 24, 30, 36, 42, 48)	(0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7)	0.3	0.3332	0.4900	0.3336	0.4627	0.3226	0.3149
		0.8	0.9737	0.9882	0.9740	0.9750	0.9798	0.9740
		1.5	0.9950	1	0.9940	0.9920	0.9906	0.9912
	(0.1, 0.4, 0.7, 1.1, 1.5, 1.9, 2.3)	0.3	0.1277	0.1966	0.1284	0.1814	0.1228	0.1205
		0.8	0.7067	0.8465	0.7070	0.8059	0.6955	0.6979
		1.5	0.9990	0.9970	0.9995	0.9649	0.9950	0.9988
	(1, 2, 3, 4, 5, 6, 7)	0.3	0.0712	0.0870	0.0718	0.0798	0.0675	0.0678
		0.8	0.2458	0.3612	0.2452	0.3438	0.2307	0.2322
		1.5	0.7381	0.8667	0.7380	0.8494	0.7108	0.7255
	(1, 4, 7, 11, 15, 19, 23)	0.3	0.0544	0.0636	0.0541	0.0593	0.0557	0.0530
		0.8	0.0998	0.1537	0.0993	0.1388	0.0956	0.0938
		1.5	0.2679	0.4137	0.2675	0.3839	0.2616	0.2539
(20, 30, 40, 50, 60, 70, 80)	(0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7)	0.3	0.5722	0.7108	0.5720	0.7148	0.5596	0.5577
		0.8	0.9990	0.9614	0.9987	0.9911	0.9960	0.9980
		1.5	1	0.9614	1	1	1	1
	(0.1, 0.4, 0.7, 1.1, 1.5, 1.9, 2.3)	0.3	0.1939	0.2736	0.1935	0.2647	0.1885	0.1875
		0.8	0.9270	0.9103	0.9262	0.9315	0.9254	0.9286
		1.5	0.9921	0.9261	0.9919	0.9509	0.9901	0.9941
	(1, 2, 3, 4, 5, 6, 7)	0.3	0.0835	0.1185	0.0832	0.1079	0.0834	0.0799
		0.8	0.4142	0.5701	0.4140	0.5418	0.4040	0.4014
		1.5	0.9426	0.9747	0.9423	0.9664	0.9433	0.9380
	(1, 4, 7, 11, 15, 19, 23)	0.3	0.0551	0.0709	0.0542	0.0658	0.0563	0.0551
		0.8	0.1388	0.2142	0.1369	0.2012	0.1352	0.1343
		1.5	0.4446	0.6286	0.4442	0.6072	0.4363	0.4414

AQuadtree: an R Package for Quadtree Anonymization of Point Data

by Raymond Lagonigro, Ramon Oller and Joan Carles Martori

Abstract The demand for precise data for analytical purposes grows rapidly among the research community and decision makers as more geographic information is being collected. Laws protecting data privacy are being enforced to prevent data disclosure. Statistical institutes and agencies need methods to preserve confidentiality while maintaining accuracy when disclosing geographic data. In this paper we present the AQuadtree package, a software intended to produce and deal with official spatial data making data privacy and accuracy compatible. The lack of specific methods in *R* to anonymize spatial data motivated the development of this package, providing an automatic aggregation tool to anonymize point data. We propose a methodology based on hierarchical geographic data structures to create a varying size grid adapted to local area population densities. This article gives insights and hints for implementation and usage. We hope this new tool may be helpful for statistical offices and users of official spatial data.

Introduction

Data privacy is a main concern of statistical institutes, national agencies and any other institution responsible for the collection and delivery of statistical information. Current spatial data processing techniques enable linking accurate geographic information to other statistical data (Armstrong and Ruggles, 2005). Thus, the disclosure of geolocated data needs special attention in terms of privacy to avoid re-identification processes. For instance, individual information could be revealed with reverse geocoding techniques when the data delivered includes the location (Curtis et al., 2006; Zimmerman and Pavlik, 2008; Cassa et al., 2008).

Besides, in order to perform quality spatial analysis, there has always been a great demand by researchers and decision makers for accessing accurate geographic data, at fine scales (Duncan and Pearson, 1991; O'Keefe and Rubin, 2015; Chen et al., 2017). There is a strong need to develop techniques and methods to preserve confidentiality when publishing geographic information while, at the same time, providing accurate data. Statistical disclosure control techniques for geographic data must balance both goals, accuracy and privacy.

Several masking methods and frameworks for spatial point data have been proposed and analyzed (Cox, 1996; Armstrong and Ruggles, 2005; Zimmerman et al., 2007). Some geographic masking techniques alter the position of each point event to ensure the actual locations cannot be discovered. On the other hand, geographic aggregation techniques group individual point information (Zimmerman et al., 2007). Political or administrative institutions are often used as units of aggregation, because data is normally collected and summarized for those units. For instance, national agencies usually use counties, neighbourhoods or census tracts to gather and distribute statistical information. Alternatively, information can be collected and summarized on a regular or irregular net of geographically referenced grid cells (Tammilehto-Luode et al., 2000; Tammilehto-Luode, 2011). Grid datasets avoid the dependence on administrative boundaries which may introduce biases on the data.

Some authors have defined different measures of disclosure risks on anonymized datasets to effectively provide guarantees for privacy protection. Sweeney (2002) introduced the concept of *k-anonymity* and proposed generalization and suppression techniques to preserve privacy and truthfulness. An anonymized database accomplishes *k-anonymity* if the set of attributes leading to the identification of an individual cannot be distinguished from at least $k-1$ other individuals. In the context of geographic data privacy, *k-anonymity* requires that for any individual in the dataset, his location is indistinguishable from at least $k-1$ other individuals. Thus, to achieve it, geographic aggregation methods must ensure that all the geographic areas created in the aggregation process include at least k individuals (Vu et al., 2012). While *k-anonymity* refers to the identification of individuals, it cannot completely prevent attribute disclosure. For instance, low diversity or background knowledge are situations which may allow identification of a sensitive attribute for individuals in a geographic area. To avoid revealing confidential or sensitive information, geographic data privacy frameworks must apply additional techniques to complement *k-anonymity* limitations (Machanavajjhala et al., 2007; Li et al., 2007).

There are some packages providing functions for data privacy protection in *R*, although they do not address spatial data. For instance, **anonymizer** (Hendricks, 2015) is a package that provides hash functions to anonymize data containing personal identifiable information. The **SciencePo** package

(Marcelino, 2013), aimed to analyse political behaviour data, provides the anonymize function that replaces individual identifier variables with combinations of random samples. The package **sdcMicro** (Templ et al., 2015) provides a more exhaustive list of functions for the anonymization of micro-data. It implements methods to evaluate and anonymize confidential micro-data sets with micro-aggregation techniques. Several perturbation methodologies are implemented in the package, such as, shuffling, micro-aggregation, post-randomization, local suppression and some others. This package may also be used for measuring information loss and disclosure risk of anonymized data.

The lack of specific methods in *R* to anonymize spatial point data motivated the development of the AQuadtree package (Lagonigro et al., 2020). The package implements functions for the analysis and anonymization of spatial point data sets with aggregation and local suppression techniques and provides an S4 *R* class for the creation, manipulation and export of spatial grids. This article presents the **AQuadtree** package and explains the anonymization methodology used in the package.

Quadtree anonymization

The novel framework proposed here pursues the data accuracy at the smallest possible areas avoiding individual information disclosure. Aggregation and local suppression of point data is performed using a methodology based on hierarchical (quadtree) geographic data structures. The final result is a varying size grid adapted to local area population densities as described in Lagonigro et al. (2017). It follows the guidelines for grid datasets of GEOSTAT project (GEOSTAT, 2014) and uses the grid coding system defined in INSPIRE Data specifications (INSPIRE, 2010).

The European Forum for Geography and Statistics (EFGS), within the GEOSTAT project, promoted a common framework for grid based statistics across all the European countries (GEOSTAT, 2011, 2014). The project aimed to represent several European characteristics in a 1km^2 regular grid dataset and developed the guidelines for datasets and methods to link census-based data with grid datasets. The following recommendations were established regarding grid cell sizes: "*1km^2 appears to be a good compromise between data availability, data confidentiality and suitability for national to European study areas. The project also recommends introducing intermediate grid sizes based on a two-level quadtree (i.e. 250m and 500m as subdivision of the 1km grid)*" (GEOSTAT, 2011, p. 17). The project also proposes the INSPIRE grid coding system (INSPIRE, 2010) to unambiguously identify each cell of the grid dataset. INSPIRE Data specifications consider the European Terrestrial Reference System 89 and the Lambert Azimuthal Equal Area projection (ETRS89-LAEA), although other Coordinate Reference Systems (CRS) and projections are also possible. Each cell of the grid is identified by a code composed by the cell's size and the coordinates of the lower left cell corner in the ETRS89-LAEA system. The cell's size is denoted in meters ("m") for cells' sizes up to 1000 meters, or kilometers ("km") for larger cells' sizes. To reduce the length of the string, values for northing and easting are divided by 10^n (where "n" is the number of zeros in the cell size value measured in meters). For instance, the cell code *"1kmN2599E4695"* identifies the 1km^2 grid cell with coordinates of the lower left corner: *Y=2599000m*, *X=4695000m*.

The aggregation process implemented in the AQuadtree package uses a hierarchical (quadtree) data structure and follows a recursive decomposition of space (Samet, 1984; Behnisch et al., 2013). The methodology recursively splits each squared area into four equal-sized quadrants. The process builds an initial regular grid of a given cell size, where each initial cell is identified following the INSPIRE grid coding system. The initial cells are then recursively subdivided into quadrants. A second identifier containing a sequence of numbers to indicate the position of the cell when successive disaggregation has been performed (see Figure 1) is assigned to each new cell; for instance, the sequence identifier corresponding to the right top cell on the third image in Figure 1 would be *416*, i.e. *fourth cell in the first division, and sixteenth cell in the second division.*

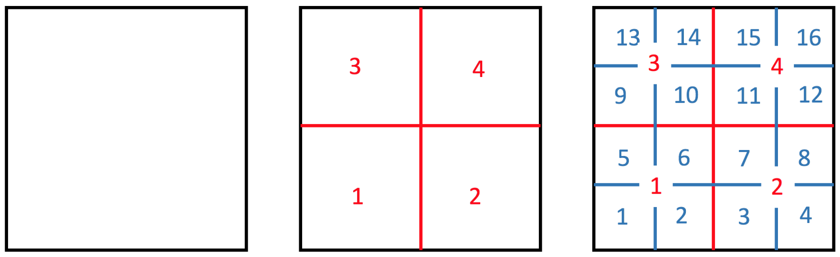
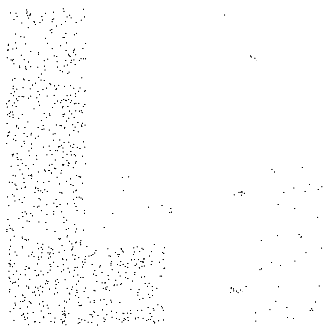


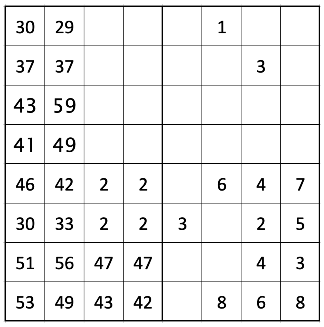
Figure 1: Three level quadtree splitting cell numbering example. Initial cell on the (left); first quadtree subdivision (center); second quadtree subdivision (right).

In the **AQuadtree** package, the spatial points are aggregated considering the *k-anonymity* requirement (Vu et al., 2012). The hierarchical subdivision process applies a minimum threshold on the number of points per cell. Hereafter we will refer to this threshold as the anonymity threshold. A cell is recursively split while the number of points in each sub-cell is greater than this threshold. Additionally, in order to avoid attribute disclosure, it is possible to set thresholds for sensitive attributes. Then, the number of points per cell in groups defined by the attributes should be greater than these supplementary thresholds.

During the splitting process all the resulting cells need to satisfy the threshold restriction, otherwise the division is not performed. In some cases, the threshold restriction may prevent the division of a cell with a very irregular point pattern, which would result in less accuracy on the cell resolution. For instance, Figure 2a, presents a pattern of 932 points unevenly distributed on a 1km cell and Figure 2b shows the corresponding grid of 62.5m cells with no threshold restrictions (the total number of points aggregated in each cell is shown).



(a)

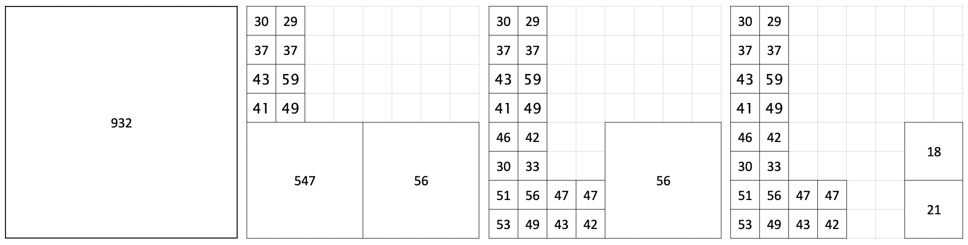


30	29				1		
37	37					3	
43	59						
41	49						
46	42	2	2		6	4	7
30	33	2	2	3		2	5
51	56	47	47			4	3
53	49	43	42		8	6	8

(b)

Figure 2: Set of spatial points (a) and the corresponding 62.5m grid with no threshold restrictions (b) (the numbers indicate the points aggregated in each cell).

Given the cell in Figure 2a and using, for example, a value of 17 as the anonymity threshold, the subdivision process could not proceed because one out of the four resulting quadrants contains only four points. The privacy mechanism would aggregate all the points in the initial cell as presented in Figure 3a, masking an irregular spatial distribution. In order to get more resolution accuracy the `AQuadtree` algorithm considers the suppression of some points before continuing the disaggregation. For instance, the disaggregation shown in Figure 3b is clearly more accurate to the underlying spatial distribution and it would result from suppression of the four points in the top right quadrant of Figure 2b. Moreover, the elimination of more data points would lead to further disaggregation (Figure 3c and Figure 3d).



(a) **(b)** **(c)** **(d)**

Figure 3: Disaggregation examples with an anonymity threshold value of 17 points per cell. a) no disaggregation and no loss; b) disaggregation with suppression of 4 points; c) more disaggregation with suppression of 12 points; c) maximum disaggregation with suppression of 29 points.

Such cases require a strategy, balancing the need of keeping the maximum number of points (minimum information loss) and getting the finest scale cells (maximum resolution accuracy). In order to do this, the method computes the Theil index of inequality (Theil, 1972) for the number of points in the possible quadrants as well as the percentage of suppressed points needed for the division to be carried out. Going back to the example in Figure 2, the initial cell can be subdivided into four quadrants with 547, 56, 325 and 4 points respectively (see Figure 4). The Theil measure four quadrants is given by $\sum\left(x_i \cdot ln\left(x_i/\overline{x}\right)\right)/\sum x_i = 0.514$, indicating high inequality, i.e. if the cell is not subdivided it masks a very uneven distribution of points. Suppressing the four points in the fourth quadrant to split the cell produces a 0.43% loss of points.

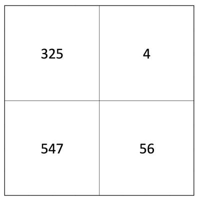


Figure 4: Quadrant subdivision.

Therefore, where the anonymity threshold value prevents disaggregation, high values of inequality measure suggest the need for further subdivision, while high values of the loss rate suggest to stop the subdivision. The algorithm uses default limits for both measures: 0.25 and 0.4 respectively (both values can be defined by the user between 0 and 1). Thus, if there exists any sub-cell with a number of points lower than the anonymity threshold and the Theil entropy is higher than 0.25, then the disaggregation process continues by suppressing these points as long as the loss rate is lower than 0.4. Hence, going back to the example in Figure 2, the default disaggregation produced by the method would be the one shown in Figure 3b.

In order to minimize the effects of information loss, if the number of suppressed points exceeds the anonymity threshold, suppressed points are aggregated into the initial cell. This cell is maintained and marked as a residual cell. Following with the example in Figure 2, the 29 suppressed points in Figure 3d will be marked as a residual cell (see Figure 5).

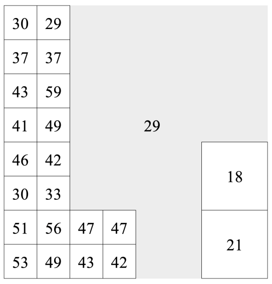


Figure 5: Example of a residual cell.

Next section presents the **AQuadtree** package. For a given set of spatial points, the package main function creates an *R* spatial object of class `AQuadtree` representing an irregular grid with varying cell sizes, and provides a summary of the aggregated data. The function accepts several parameters to adjust the characteristics of the resulting grid. For instance, initial cell size, number of recursive subdivisions to achieve, the threshold on the minimum number of points per cell, the threshold for information loss and inequality or the attributes to be summarized. The package defines methods to manipulate or export `AQuadtree` objects and includes test datasets.

The AQuadtree package

Installation and dependencies

The AQuadtree package can easily be installed from any CRAN repository by executing the R command:

```
R> install.packages("AQuadtree")
```

The **AQuadtree** depends on several *R* packages as shown on the corresponding web page on CRAN[1]. The packages **sp** (Pebesma and Bivand, 2005; Bivand et al., 2013) and **dplyr** (Wickham et al., 2020) will automatically be installed along with **AQuadtree** if the `dependencies` argument is set to `TRUE` on the installation process. The **sp** package provides classes and methods for representation and manipulation of spatial data on which **AQuadtree** relies. The **dplyr** package provides fast and consistent tools for working with data frames which are used in **AQuadtree** to decrease computing times on the automatic aggregation processes. The packages **rgeos** (Bivand and Rundel, 2017), **rgdal** (Bivand et al., 2016) are also useful for some extra functions of the package as, for instance, exporting a `shapefile` for the created grid.

[1]`https://CRAN.R-project.org/package=AQuadtree`

```
R> library(AQuadtree)
Loading required package: sp
Loading required package: dplyr
```

Provided data

In order to probe and test the package functionality, two `SpatialPointsDataFrame` objects are included: `BarcelonaPop` for Barcelona Municipality (Spain) and `CharlestonPop` for Charleston, metropolitan area (USA). Point data was created randomly with the distributions of real data at census scale obtained from different sources.

The package also provides two `SpatialPolygons` objects with the spatial boundaries for each region. `BarcelonaCensusTracts` contains spatial limits of the census tracts for the Barcelona Municipality, and `CharlestoneCensusTracts` provides spatial limits of the census tracts for Charleston, metropolitan area.

`BarcelonaPop` comprises 81,359 sample points in Barcelona, Spain. The original information was obtained from the department of statistics of the Ajuntament de Barcelona and it contains population data (age and sex) at census tract level for 2018 (Ajuntament de Barcelona. Departament d'Estadística, 2018). The points were generated and distributed randomly in space, keeping information at each census tract unchanged. In order to reduce memory allocation, it only contains a 5% sample of points.

```
R> data("BarcelonaPop", package="AQuadtree")
R> summary(BarcelonaPop)
Object of class SpatialPointsDataFrame
Coordinates:
      min     max
x 3655447 3669871
y 2059179 2074546
Is projected: TRUE
proj4string :
[+proj=laea +lat_0=52 +lon_0=10 +x_0=4321000 +y_0=3210000 +ellps=GRS80
+towgs84=0,0,0,0,0,0,0 +units=m +no_defs]
Number of points: 81359
Data attributes:
      age             sex
 Min.   :  0.00   man  :38472
 1st Qu.: 27.00   woman:42887
 Median : 43.00
 Mean   : 43.94
 3rd Qu.: 61.00
 Max.   :100.00
```

In a similar way, the `CharlestonPop` object, containing 54,619 random sample points, was created by using information in Charleston1 dataset from the 2000 Census Tract Data for the Charleston metropolitan area (USA) (Geoda Data and Lab, 2019). For size concerns, it only contains a 10% sample of points.

The AQuadtree class

An `AQuadtree` class object is a spatial dataset representing a varying size grid and is created performing an aggregation of a given set of points observing a minimum threshold for the number of points in each cell. The main function of the package creates the `AQuadtree` object from `SpatialPoints` or `SpatialPointsDataFrame` objects.

```
R> bcn.QT <- AQuadtree(BarcelonaPop)
R> class(bcn.QT)
[1] "AQuadtree"
attr(,"package")
[1] "AQuadtree"
```

As seen in the example, only the spatial object is necessary. Several extra parameters can be used in order to calibrate the resulting object: `dim`, `layers`, `colnames`, `threshold`, `thresholdField`, `funs`, `as`, `ineq.threshold`, `loss.threshold`. All parameters are optional and have default values. For instance, the anonymity threshold is equal to 100 by default. In the next sections all the parameters are detailed.

The plot method for the `AQuadtree` class overrides the generic function for plotting *R* objects. Figure 6 shows the `AQuadtree` created for Barcelona. It comprises areas with very different sizes. Therefore, highly populated zones are disaggregated in small cells to allow dissemination of accurate data whereas, information on low populated zones is aggregated at a higher scale. In some cases, to improve accuracy and create smaller cells, the function removes some points and groups them in cells marked as residual (those printed in red by the plot method).

```
R> plot(bcn.QT)
```

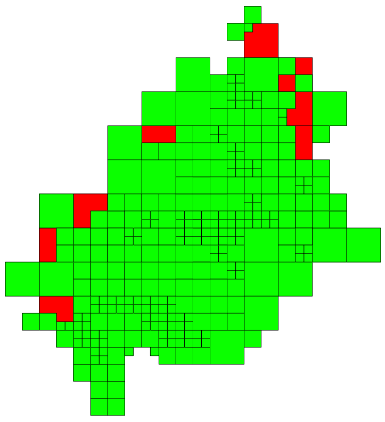


Figure 6: Plot of an AQuadtree for Barcelona.

In the next example the `names` generic function shows the slot names created for each cell in the `AQuadtree` object. Then, in order to show the information from some of the `AQuadtree` cells, we use the overridden `[` (sub-setting) method to extract and print a subset of the cells in the `AQuadtree` (see Section Class methods and slots for details on overridden methods).

```
R> names(bcn.QT)
[1] "cellCode"  "cellNum"  "level"  "residual"  "total"
R> bcn.QT[310:315, ]
An object of class "AQuadtree" with 6 grid cells with sizes between 1km and
125m
         cellCode cellNum level residual total
310 1kmN2072E3665     204     3    FALSE   128
311 1kmN2072E3665     207     3    FALSE   138
312 1kmN2072E3665     208     3    FALSE   166
313 1kmN2073E3666     313     3    FALSE   135
314 1kmN2065E3660             1     TRUE   133
315 1kmN2067E3660             1     TRUE   109
```

The `cellCode` and the `cellNum` properties identify each cell in the grid, following the mechanism of the INSPIRE grid coding system (INSPIRE, 2010). As explained in detail in paragraphs 2 and 3 of section Quadtree anonymization: `cellCode` refers to the initial regular grid (the highest aggregation level) and indicates the cell's size and the coordinates of the lower left cell corner in the false origin of the CRS (Coordinate Reference System); `cellNum` refers to the varying size grid and is the sequence of numbers indicating the position of the cell in every successive subdivision of the initial cell. Three more properties are also added to each cell; the `level` attribute, that indicates the scale of disaggregation of the cell; the `residual` logical value, that states whether or not the cell has been created to store residual points; and finally, the `total` property, indicating the number of points aggregated in the cell.

General usage

The characteristics of the `AQuadtree` object can be adjusted with several parameters. First, the `dim` parameter defines the cell size of the initial grid in meters. The `layers` parameter indicates the number of subdivisions to perform. Thus, specifying the parameters `dim=10000` and `layers=5` would create a grid with cells sizes ranging between `10km` and `625m`. If `dim` and `layers` are not specified, the default values define an initial size of 1000 meters and 5 subdivisions.

```
R> charleston.QT <- AQuadtree(CharlestonPop, dim=10000, layers=5)
```

The `summary` method for `AQuatree` summarizes information of an `AQuatree` object:

```
R> summary(charleston.QT)
Object of class "AQuadtree"
183 grid cells with sizes between 10km and 625m
Coordinates:
      min     max
x 2060000 2160000
y  110000  220000
Is projected: TRUE
proj4string:
 +init=epsg:26978 +proj=lcc +lat_1=38.56666666666667
 +lat_2=37.26666666666667 +lat_0=36.66666666666666
 +lon_0=-98.5 +x_0=400000 +y_0=400000 +datum=NAD83
 +units=m +no_defs +ellps=GRS80 +towgs84=0,0,0
Initial Cell Size: 10km
Number of valid grid Cells: 179
Number of residual grid Cells: 4
Data attributes:
     total
 Min.   : 100.0
 1st Qu.: 156.0
 Median : 217.0
 Mean   : 287.4
 3rd Qu.: 358.0
 Max.   :2281.0
```

If the original dataset includes individual information, the `colnames` parameter can be used to specify the attributes from the dataset to be summarized in the resulting grid. Factor attributes are deployed and a new attribute for each factor label is created. For instance, an attribute `sex` with two labels, `man` and `woman`, would be deployed into the two new attributes: `sex.man`, `sex.woman`. Additionally, the `funs` parameter could be used to specify the functions to compute in the summary for each one of the attributes indicated in the `colnames` parameter. If no functions are specified, the `sum` function is used. Next example creates an `AQuatree` object with the sample points in Barcelona, retaining two attributes, `age` and `sex`, summarized with `mean` and `sum` functions respectively.

```
R> bcn.QT <- AQuadtree(BarcelonaPop,
 + colnames=c('age','sex'),
 + funs=c('mean', 'sum'))
R> summary(bcn.QT)
Object of class "AQuadtree"
321 grid cells with sizes between 1km and 125m
Coordinates:
      min     max
x 3659000 3670000
y 2062500 2074500
Is projected: TRUE
proj4string:
 +proj=laea +lat_0=52 +lon_0=10 +x_0=4321000 +y_0=3210000 +ellps=GRS80
 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs
Initial Cell Size: 1km
Number of valid grid Cells: 313
Number of residual grid Cells: 8
Data attributes:
     total            age            sex.man         sex.woman
 Min.   : 100   Min.   :35.28   Min.   : 40.0   Min.   : 44.0
 1st Qu.: 139   1st Qu.:42.37   1st Qu.: 64.0   1st Qu.: 73.0
 Median : 177   Median :44.42   Median : 83.0   Median : 95.0
 Mean   : 248   Mean   :44.16   Mean   :117.1   Mean   :130.9
 3rd Qu.: 328   3rd Qu.:46.18   3rd Qu.:158.0   3rd Qu.:170.0
 Max.   :1288   Max.   :51.18   Max.   :626.0   Max.   :662.0
```

Anonymity threshold

The package applies a default anonymity threshold value of 100 and it can be adjusted with the `threshold` parameter. By default, the threshold restriction is only applied to the total number of points aggregated in each cell (i.e. the `total` attribute added to the resulting dataset). The threshold restriction can also be applied to any attribute with confidential information. The attribute names that must satisfy the given threshold should be specified in the `thresholdField` parameter.

Next example creates an `AQuadtree` applying a threshold value of `17` on the attributes `sex.man` and `sex.woman`. As shown in the summary, the minimum values for both attributes is `17`; cells with lower values have been grouped to achieve the minimum threshold.

```
R> bcn.QT <- AQuadtree(BarcelonaPop, colnames=c('age','sex'), funs=c('mean', 'sum'),
 + threshold=17, thresholdField=c("sex.man", "sex.woman"))
R> summary(bcn.QT)
Object of class "AQuadtree"
730 grid cells with sizes between 1km and 62.5m
Coordinates:
      min     max
x 3659000 3670000
y 2062000 2075000
Is projected: TRUE
proj4string:
 +proj=laea +lat_0=52 +lon_0=10 +x_0=4321000 +y_0=3210000 +ellps=GRS80
 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs
Initial Cell Size: 1km
Number of valid grid Cells: 713
Number of residual grid Cells: 17
Data attributes:
     total            age           sex.man         sex.woman
 Min.   : 34.0   Min.   :32.63   Min.   : 17.00   Min.   : 17.00
 1st Qu.: 64.0   1st Qu.:41.52   1st Qu.: 30.25   1st Qu.: 33.00
 Median :103.0   Median :43.87   Median : 49.00   Median : 54.00
 Mean   :110.5   Mean   :43.71   Mean   : 52.25   Mean   : 58.21
 3rd Qu.:140.0   3rd Qu.:46.08   3rd Qu.: 65.75   3rd Qu.: 74.00
 Max.   :807.0   Max.   :53.46   Max.   :371.00   Max.   :436.00
```

Balancing information loss and accuracy

Finally, two more parameters can be used to manage the disaggregation process: the inequality threshold (`ineq.threshold`) and the loss rate threshold (`loss.threshold`). As explained in Section Quadtree anonymization, when the anonymity threshold prevents the subdivision of a cell, then the suppression of some points may be considered to allow a further subdivision (see Figures 2 and 3). This decision depends on the inequality between sub-cells (we use the Theil index) and the rate of points that need to be suppressed (we call it loss rate). As matter of fact, the disaggregation process of a cell continues either when the number of points in each sub-cell is greater than the anonymity threshold (threshold), or alternatively when the inequality between sub-cells is greater than the inequality threshold (`ineq.threshold`) and the loss rate is smaller than the loss threshold (`loss.threshold`). The `ineq.threshold` and the `loss.threshold` parameters range between `0` and `1` and the default values are `0.25` and `0.4` respectively. Lower values in the inequality threshold and/or higher values in the loss threshold result in grids with smaller cells.

To illustrate the use of the `ineq.threshold`, in the next example a subset of points corresponding to a cell of 1km, with bottom-left corner `(3660000,2065000)` and top-right corner `(3661000,2066000)` is created:

```
R> point.coord<-as.data.frame(coordinates(BarcelonaPop))
R> points.subset<- BarcelonaPop[point.coord[,1]>=3660000 & point.coord[,1]<=3661000
 +  & point.coord[,2]>=2065000 & point.coord[,2]<=2066000,]
R> plot(points.subset)
R> length(points.subset)
```

The result is the subset of points shown in Figure 7a, which are clustered mainly on the bottom right and bottom left corners of the area. In order to compare the different aggregation results, three `AQuadtree` objects were created using different `ineq.threshold` values. (Figure 7b, 7c, and 7d). For each case the proportion of residual points as a measure of loss, and the coefficient of variation as a measure of dispersion, are computed.

Using a minimum threshold value of 5 points per cell and the default inequality value (0.25), the function creates the grid shown in Figure 7b. It presents a residual cell containing 2.02% of the points, and the number of points per cell has a coefficient of variation of 0.499.

```
R> bcn.QT <- AQuadtree(points.subset, threshold=5)
R> plot(bcn.QT, residual=F, col="grey")
R> bcn.QT[bcn.QT$residual,]$total / length(points.subset)
R> sd(bcn.QT[!bcn.QT$residual,]$total)/mean(bcn.QT[!bcn.QT$residual,]$total)
```

Figure 7c shows a second aggregation example, for the same set of points, with smaller cells, using an ineq.threshold value of 0.01, which results in 8.89% of residual points and a coefficient of variation of 0.365 (i.e. the number points in cells is more similar between them).

```
R> bcn.QT <- AQuadtree(points.subset, threshold=5, ineq.threshold=0.01)
R> plot(bcn.QT, residual=F, col="grey")
R> bcn.QT[bcn.QT$residual,]$total / length(points.subset)
R> sd(bcn.QT[!bcn.QT$residual,]$total)/mean(bcn.QT[!bcn.QT$residual,]$total)
```

Figure 7d on the contrary, uses a higher ineq.threshold value of 0.5, resulting in aggregation on bigger cells, with no loss, but a coefficient of variation of 0.84 (i.e. the number of points in each cell exhibits more inequality).

```
R> bcn.QT <- AQuadtree(points.subset, threshold=5, ineq.threshold=0.5)
R> plot(bcn.QT, residual=F, col="grey")
R> bcn.QT[bcn.QT$residual,]$total / length(points.subset)
R> sd(bcn.QT[!bcn.QT$residual,]$total)/mean(bcn.QT[!bcn.QT$residual,]$total)
```

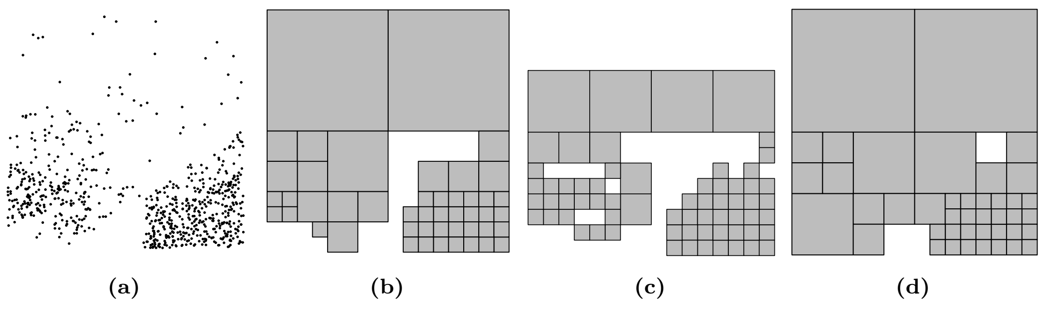


Figure 7: Examples of the effect of the ineq.threshold parameter. a) Subset of points used in the example; b) AQuadtree with default ineq.threshold value; c) AQuadtree with ineq.threshold value 0.01; d) AQuadtree with ineq.threshold value 0.5.

Class methods and slots

The `AQuadtree` class proposes a collection of methods to manage the generated objects and overrides the generic methods `show`, `print`, `summary` and `[` (subsetting) for the `AQuadtree` signature. The `spplot` method overrides the lattice-based plot method for spatial data with attributes from **sp** package (Pebesma and Bivand, 2005), and two extra parameters: `residual`, to indicate if residual cells should be displayed, and `by.density` to indicate if attributes should be divided for the cell areas to make them comparable between zones with different sizes. The `merge` method merges the input data on the `AQuadtree` object with a second data frame using `cellCode` and `cellNum` as the merging attributes. The `writeOGR.QT` method coerces the given `AQuadtree` object to a `SpatialPolygonsDataframe` and uses the `writeOGR` method from **rgdal** package (Bivand et al., 2016) to write out spatial vector data. Finally, the `area.QT` method allows getting the areas of the grid cells in square meters. An `AQuadtree` object can also be coerced to a `SpatialPolygonsDataFrame` or a `SpatialPolygons` object using the generic method `as` from methods package:

```
R> bcn.QT <- AQuadtree(BarcelonaPop)
R> class(bcn.QT)
[1] "AQuadtree"
attr(,"package")
[1] "AQuadtree"
R> bcn.SP <- as(bcn.QT, "SpatialPolygonsDataFrame")
R> class(bcn.SP)
[1] "SpatialPolygonsDataFrame"
attr(,"package")
[1] "sp"
```

The basic structure of the `AQuadtree` class is based on a `SpatialPolygonsDataFrame` with some extra information on the object created.

```
R> slotNames(example.QT)
[1] "dim" "layers" "colnames" "threshold" "thresholdField" "loss" "data"
[8] "polygons""plotOrder" "bbox" "proj4string"
```

The properties `dim`, `layers`, `colnames`, `threshold` and `thresholdField` specify the parameters used to create the grid and denote the grid characteristics: `dim` indicates the scale in meters of the highest level cells; `layers` shows the number of subdivision levels; `colnames` enumerates the attribute names summarized in the resulting grid; `threshold` is the value used for anonymization, indicating the minimum number of points or a minimum number an attribute that a cell may contain; and `thresholdField` specifies the attributes to which the threshold restriction has been applied.

The aggregation process aims to keep all the original points in the summarized grid, but in some cases, when the aggregation at the highest level (the biggest cells) does not satisfy the threshold restriction those points must be suppressed. The `loss` property indicates the number of points suppressed on the subdivision process because their aggregation, at any level, does not accomplish the threshold restriction. The rest of the `slots`, i.e. `data`, `polygons`, `plotOrder`, `bbox` and `proj4string`, define the grid structure as a `SpatialPolygonsDataFrame`.

Package functions

The **AQuadtree** package also implements several functions that help working with the proposed grid system. In this section we focus on the capabilities for merging two `AQuadtree` objects, for aggregating spatial points to an `AQuadtree` object and, finally, for creating a fixed size grid.

Join AQuadtrees

As pointed by Martin (2002), administrative units may change over time, but grid cells are time independent. Therefore, grid datasets from different periods can be compared. In the case of aquadtrees, this is true only when the disaggregation process gets the same resolutions in all areas for the periods being compared; i.e. information in two datasets is only comparable if there are exactly the same cells in both datasets. A problem arises when some cells have different subdivision levels. The function `joinAQuadtrees` is intended to deal with this problem. The function `joinAQuadtrees` merges two `AQuadtree` objects and makes their information comparable. The function creates a new `AQuadtree` object combining the information of the two input objects at the lowest common resolution. The initial size of the highest scale cells should be the same in both input objects.

The resulting `AQuadtree` object maintains the attributes of the two input objects. The attributes are automatically renamed, adding the suffix `".1"` or `".2"` to indicate whether the attribute comes from the first or the second object. When the grids have different subdivision schemes, the algorithm works in a quadtree manner, it merges the cells to bigger ones and aggregates information. The optional parameters `mean.1` and `mean.2` can be used to indicate a list of attributes that should be aggregated using a weighted mean instead of the `sum` function applied by default. The function can be used to merge `AQuadtrees` with the same attributes, but also with different information.

The following example combines two `AQuadtrees`, with different information. The first `AQuadtree` contains the population of Barcelona maintaining only the `age` attribute with a threshold value `25` on the number of points per grid cell.

```
R> Barcelona.AQT_1<-AQuadtree(BarcelonaPop,colnames="age",threshold=25, fun="mean")
```

A second `AQuadtree` for the same population preserving the `sex` attribute with a threshold value `17` is also created.

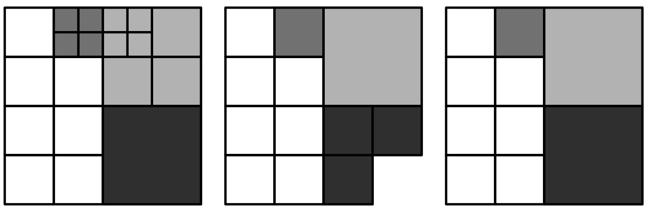
```
R> Barcelona.AQT_2 <- AQuadtree(BarcelonaPop, colnames="sex", threshold=17,
 + thresholdField=c("sex.man", "sex.woman"))
```

Now both objects are combined to create a new `AQuadtree` aggregating the `age` attribute using the weighted mean.

```
R> Barcelona.AQT_1_2 <- joinAQuadtrees(Barcelona.AQT_1, Barcelona.AQT_2,
 + mean.1="age", withResiduals=TRUE)
```

To illustrate how the merging process works we focus on a particular cell.

```
R> plot(Barcelona.AQT_1[Barcelona.AQT_1$cellCode=="1kmN2065E3665",])
R> plot(Barcelona.AQT_2[Barcelona.AQT_2$cellCode=="1kmN2065E3665",])
R> plot(Barcelona.AQT_1_2[Barcelona.AQT_1_2$cellCode=="1kmN2065E3665",])
```



(a) (b) (c)

Figure 8: Cell join example. AQuadtree objects in a) and b) are joined in c).

Figure 8 presents the results of the previous example, the input grids are shown in Figure 8a and 8b and the output grid is shown in Figure 8c. Some cells have been highlighted in different grey tones to clarify the merging process. For instance, looking at the bottom right cells in Figure 8a and 8b (see the darkest gray-scale level), the three cells in Figure 8b, must be merged to make them comparable to the one in Figure 8a. The resulting grid in Figure 8c preserves the cell in Figure 8a and aggregates the information of the cells in Figure 8b. Table 1 presents the attribute information in each cell of the input and in the output grid to clarify the merging performed.

Barcelona.AQT_1				Barcelona.AQT_2					Barcelona.AQT_1_2						
cell Num	level	total	age	cell Num	level	total	sex. man	sex. woman	cell Num	level	total .1	age .1	total .2	sex. man .2	sex. woman .2
101	3	79	43.6	101	3	79	37	42	101	3	79	43.6	79	37	42
102	3	133	40.9	102	3	133	75	58	102	3	133	40.9	133	75	58
105	3	63	45.0	105	3	63	37	2	105	3	63	45.0	63	37	26
106	3	93	40.0	106	3	93	54	39	106	3	93	40.0	93	54	39
2	2	65	38.1	203	3	118	81	37	2	2	265	38.1	253	146	107
				207	3	92	47	45							
				208	3	43	18	25							
309	3	79	41.5	309	3	79	38	41	309	3	79	41.5	79	38	41
310	3	121	38.8	310	3	121	54	67	310	3	121	38.8	121	54	67
313	3	89	35.4	313	3	89	50	39	313	3	89	35.4	89	50	39
31451	4	41	44.5	314	3	158	68	90	314	3	158	39.6	158	68	90
31452	4	44	39.4												
31459	4	32	34.1												
31460	4	41	39.2												
411	3	128	39.8	4	2	379	192	187	4	2	379	41.0	379	192	187
412	3	62	40.1												
41553	4	53	46.2												
41554	4	32	35.6												
41561	4	42	41.9												
41562	4	31	41.3												
416	3	31	43.0												

Table 1: Cell join example.

The `joinAQuadtrees` function can also be used to incorporate a set of points to an existing `AQuadtree` object building a grid with threshold of 1 point per cell. For instance, let us consider an `AQuadtree` aggregating the population for the Charleston SCMA using a threshold value 17 individuals per cell and a `SpatialPointsDataFrame` object from the same population selecting white people aged 65 or more:

```
R> Charleston.AQT<-AQuadtree(CharlestonPop, threshold=17)
R> head(Charleston.AQT)
An object of class "AQuadtree" with 6 grid cells with sizes between 1km and
125m
       cellCode cellNum level residual total
1 1kmN0126E2135            1    FALSE    22
2 1kmN0127E2135            1    FALSE    20
3 1kmN0127E2136            1    FALSE    24
4 1kmN0131E2140            1    FALSE    29
5 1kmN0133E2134            1    FALSE    45
6 1kmN0133E2135            1    FALSE    20

R> CharlestonWomen65 <- CharlestonPop[CharlestonPop$origin=='white'
 + & CharlestonPop$age=='over65', 'sex']
```

Both objects can be aggregated by creating an auxiliary `AQuadtree` with threshold value 1 (i.e. with maximum level of disaggregation) for the set of points

```
R> CharlestonWomen65.AQT<-AQuadtree(CharlestonWomen65, threshold=1, colnames='sex')
```

and then joining both `AQuadtree` objects:

```
R> Charleston.AQT.ext<-joinAQuadtrees(Charleston.AQT, CharlestonWomen65.AQT)
R> head(Charleston.AQT.ext)
An object of class "AQuadtree" with 6 grid cells with sizes between 1km and
125m
     cellCode cellNum level residual total.1 total.2 sex.male.2 sex.female.2
1 1kmN0126E2135           1    FALSE      22       4          2            2
2 1kmN0127E2135           1    FALSE      20       2          1            1
3 1kmN0127E2136           1    FALSE      24       3          1            2
4 1kmN0131E2140           1    FALSE      29       2          1            1
5 1kmN0133E2134           1    FALSE      45       5          4            1
6 1kmN0133E2135           1    FALSE      20       2          1            1
```

The resulting `AQuadtree` object may contain less cells than the original, because joining `AQuadtrees` only merges common geometries.

```
R> length(Charleston.AQT)
[1] 997
R> length(Charleston.AQT.ext)
[1] 856
```

An optimized function to aggregate points to an existing `AQuadtree` object maintaining all the original cells is explained in the next subsection.

Aggregate points to an AQuadtree

As seen in the previous subsection, a set of points may be converted into an `AQuadtree` object and then it may be joined to another `AQuadtree`. The package includes the `pointsToAQuadtree` function to optimize this process. The function takes two elements, an `AQuadtree` object and a `SpatialPoints` or a `SpatialPointsDataFrame` object, and aggregates the set of points to the `AQuadtree`. The function aggregates numeric attributes of the input set of points using the `mean` function. It also deploys factor attributes creating new attributes for each label of the factor and computing the number of occurrences. The point's attributes added to the resulting `AQuadtree` object are prefixed with `"p."`. The function also creates a `"p.total"` attribute to compute the total number of points aggregated to each cell of the input `AQuadtree`.

Following the example given in the previous section (see at the end of previous section), the aggregation of the `CharlestonWomen65` and `Charleston.AQT` objects is given by

```
R> Charleston.AQT.ext<- pointsToAQuadtree(Charleston.AQT, CharlestonWomen65)
R> tail(Charleston.AQT.ext)
An object of class "AQuadtree" with 6 grid cells with sizes between 1km and
125m
```

```
       cellCode cellNum level residual total p.total p.sex.male p.sex.female
185 1kmN0142E2133   313     3    FALSE    23      NA         NA           NA
186 1kmN0142E2133   314     3    FALSE    19       1          0            1
188 1kmN0142E2134   101     3    FALSE    27       1          1            0
189 1kmN0142E2134   102     3    FALSE    27       1          0            1
190 1kmN0142E2134   105     3    FALSE    35       4          2            2
191 1kmN0142E2134   106     3    FALSE    32       2          0            2
```

The resulting AQuadtree object has the same cells as the original, and the cells not covering any points contain NA values for the new attributes.

```
R> length(Charleston.AQT)
[1] 997
R> length(Charleston.AQT.ext)
[1] 997
```

Create a fixed size grid

The createGrid function creates a grid with fixed cell size (by default 1km), based on the INSPIRE Specification on Geographical Grid Systems (INSPIRE, 2010). The grid covers entirely a given area which can be a single polygon, a set of polygons or a set of points. The resulting grid will be a SpatialPolygons object where the ID of each polygon is the cellCode as explained in Section Quadtree anonymization. The size of the grid cells can be specified with the dim argument in meters; by default cells will be 1000m. The intersect argument specifies whether the resulting grid should be intersected with the given zone; otherwise the function builds a rectangular grid. The outline argument, indicates whether the resulting grid should be clipped with the outline borders of the given zone (only applicable with SpatialPolygons or SpatialPolygonsDataFrame classes).

Figure 9 presents an example of a SpatialPolygons object representing the census tracts for Barcelona, and three possible grids with different values for the intersect and outline arguments. Figure 9a provides the census tract spatial limits for Barcelona,

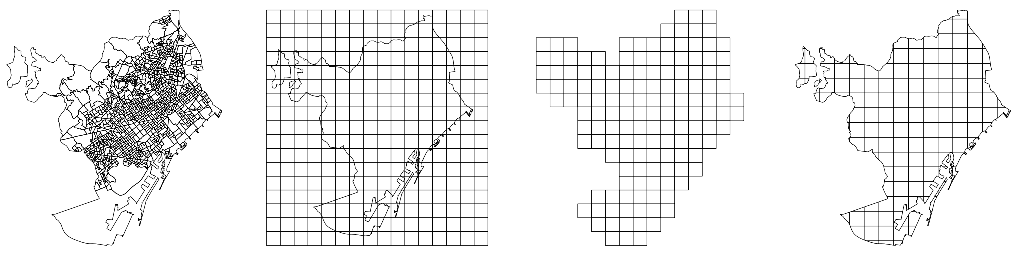
```
R> plot(BarcelonaCensusTracts)
```

Figure 9b provides a rectangular grid covering the current spatial region without intersection. To illustrate the region covered, the boundaries for the Barcelona Municipality have also been plotted,

```
R> plot(createGrid(BarcelonaCensusTracts, intersect=FALSE))
R> plot(rgeos::gUnaryUnion(BarcelonaCensusTracts), add=TRUE)
```

and finally, Figure 9c and Figure 9d provide intersected grids with and without outlined borders,

```
R> plot(createGrid(BarcelonaCensusTracts, intersect=TRUE, outline=FALSE))
R> plot(createGrid(BarcelonaCensusTracts, intersect=TRUE, outline=TRUE))
```



(a) (b) (c) (d)

Figure 9: Barcelona census tracts (a); grid for Barcelona city without intersection (b); grid intersected (c); grid intersected and outlined (d).

The polygons in the resulting grids are identified by their corresponding INSPIRE cell code (INSPIRE, 2010),

```
R> Bcn.Grid<-createGrid(BarcelonaCensusTracts, intersect=TRUE,
 + outline=FALSE)
```

```
R> row.names(Bcn.Grid)
  [1] "1kmN2074E3665" "1kmN2074E3666" "1kmN2074E3667" "1kmN2073E3664"
  [5] "1kmN2073E3665" "1kmN2073E3666" "1kmN2073E3667" "1kmN2072E3655"
  [9] "1kmN2072E3656" "1kmN2072E3657" "1kmN2072E3661" "1kmN2072E3662"
 [13] "1kmN2072E3663" "1kmN2072E3664" "1kmN2072E3665" "1kmN2072E3666"
 [17] "1kmN2072E3667" ...
```

Once a fixed size grid is created, functionalities of spatial packages like **sp** can be used. For instance, a set of points can be aggregated to the created grid:

```
R> Bcn.Grid.ext<-aggregate(BarcelonaPop[,'age'], by=Bcn.Grid, FUN="mean")
R> head(as.data.frame(Bcn.Grid.ext))
                   age
1kmN2074E3665 42.14286
1kmN2074E3666 39.38129
1kmN2074E3667 39.00000
1kmN2073E3664 66.00000
1kmN2073E3665 39.51493
1kmN2073E3666 40.06069
```

The same result can be achieved using the `spatialPointsCellCodes` function in the **AQuadtree** package. This function provides, for each point in a `spatialPointsDataFrame`, the corresponding cell codes (`cellCode`) of a rectangular grid covering the full set of points.

```
R> Bcn.points<-spatialPointsCellCodes(BarcelonaPop)
R> Bcn.points.agr<-aggregate(age~cellCode, Bcn.points, "mean")
R> Bcn.Grid$cellCode<-row.names(Bcn.Grid)
R> Bcn.Grid.ext<-merge(Bcn.Grid, Bcn.points.agr)
R> head(as.data.frame(Bcn.Grid.ext))
         cellCode      age
143 1kmN2074E3665 42.14286
144 1kmN2074E3666 39.38129
145 1kmN2074E3667 39.00000
139 1kmN2073E3664 66.00000
140 1kmN2073E3665 39.51493
141 1kmN2073E3666 40.06069
```

The advantage of this second method is that it gives more control over the aggregation process, for example, the summarizing functions provided in the **dplyr** package may be used.

```
R> Bcn.points.agr<-summarise(group_by(as.data.frame(Bcn.points),cellCode),
 +  total.points=n(), mean.age=mean(age), sd.age=sd(age))
R> Bcn.Grid.ext<-merge(Bcn.Grid, Bcn.points.agr)
R> head(as.data.frame(Bcn.Grid.ext))
         cellCode total.points mean.age   sd.age
143 1kmN2074E3665           35 42.14286 27.22579
144 1kmN2074E3666          139 39.38129 22.79283
145 1kmN2074E3667            1 39.00000       NA
139 1kmN2073E3664            1 66.00000       NA
140 1kmN2073E3665          268 39.51493 21.95441
141 1kmN2073E3666          346 40.06069 23.05115
```

Final remarks

In this paper we have presented the R package AQuadtree which provides an automatic aggregation tool to anonymize point data using a methodology based on hierarchical (quadtree) geographic data structures. The main goal of the proposed system is respecting privacy of geographic data, while, at the same time, offering best accuracy to perform truthful spatial analysis. The implemented methodology is helpful for the production of official spatial data and for researchers to deal with this type of data. Further research should evaluate the quality and usefulness of ad-hoc datasets created using the package.

Data privacy concerns are built upon the *k-anonymity* concept. The methodology to build the datasets performs aggregation and suppression of spatial data, yet controlled with thresholds defined by the user and covering as many attributes as needed. Thereby, the package can create grids where the set of attributes leading to the identification of an individual cannot be distinguished from at least $k-1$ other individuals in the same grid cell. Datasets produced ensure individual privacy as long as the anonymity thresholds used are high enough.

The process has been designed to be time efficient so it can be used with large spatial datasets. It has been tested on a 2,5 GHz Intel 4 Core i7 computer with the Catalonian 2014 population register dataset containing information on 7,566,464 individuals. For instance, a grid with 85,408 cells with sizes between 1km and 31.25m, with a threshold value of 17 individuals per cell was created in 46 seconds.

Acknowledgements

The authors most gratefully thank the Institut d'Estadística de Catalunya for providing us with the database to perform tests with real data. We thank the Data Analysis and Modeling research group (2017SGR71) for their comments on previous versions of the paper. We sincerely thank Rafa Madariaga for his diligent proofreading of the manuscript.

Bibliography

Ajuntament de Barcelona. Departament d'Estadística. Població segons Padró d'Habitants, 2018. URL https://www.bcn.cat/estadistica/catala/dades/tpob/pad/padro/a2018/. [p213]

M. P. Armstrong and A. J. Ruggles. Geographic Information Technologies and Personal Privacy. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 40(4): 63–73, 2005. doi: https://doi.org/10.3138/RU65-81R3-0W75-8V21. [p209]

M. Behnisch, G. Meinel, S. Tramsen, and M. Diesselmann. Using Quadtree Representations in Building Stock Visualization and Analysis. *Erdkunde*, 67(2):151–166, 2013. ISSN 00140015. doi: https://doi.org/10.3112/erdkunde.2013.02.04. [p210]

R. S. Bivand and C. Rundel. *rgeos: Interface to Geometry Engine - Open Source (GEOS)*, 2017. URL https://cran.r-project.org/package=rgeos. R package version 0.3-22. [p212]

R. S. Bivand, E. Pebesma, and V. Gomez-Rubio. *Applied spatial data analysis with R, Second edition*. Springer, NY, 2013. URL http://www.asdar-book.org/. [p212]

R. S. Bivand, T. Keitt, B. Rowlingson, and E. Pebesma. *rgdal: Bindings for the Geospatial Data Abstraction Library*, 2016. URL https://CRAN.R-project.org/package=rgdal. R package version 1.2-5. [p212, 217]

C. A. Cassa, S. C. Wieland, and K. D. Mandl. Re-identification of Home Addresses from Spatial Locations Anonymized by Gaussian Skew. *International journal of health geographics*, 7:45, 2008. ISSN 1476-072X. doi: https://doi.org/10.1186/1476-072X-7-45. [p209]

C.-C. Chen, J.-H. Chuang, D.-W. Wang, C.-M. Wang, B.-C. Lin, and T.-C. Chan. Balancing Geo-privacy and Spatial Patterns in Epidemiological Studies. *Geospatial health*, 2017. doi: https://doi.org/10.4081/gh.2017.573. [p209]

L. H. Cox. Protecting Confidentiality in Small Population Health and Environmental Statistics. *Statistics in medicine*, 15(17):1895–1905, 1996. doi: https://doi.org/10.1002/(SICI)1097-0258(19960915)15:17<1895::AID-SIM401>3.0.CO;2-W. [p209]

A. J. Curtis, J. W. Mills, and M. Leitner. Spatial Confidentiality and GIS: Re-engineering mortality locations from published maps about Hurricane Katrina. *International journal of health geographics*, 5:44, 2006. ISSN 1476-072X. doi: https://doi.org/10.1186/1476-072X-5-44. [p209]

G. T. Duncan and R. W. Pearson. Enhancing Access to Microdata while Protecting Confidentiality: Prospects for the Future. *Statistical Science*, 6(3):219–232, 1991. doi: https://doi.org/10.1214/ss/1177011681. [p209]

Geoda Data and Lab. Sample Data Referenced in the Tutorials for GeoDa, GeoDaSpace, and CAST. Technical report, Center for Spatial Data Science. University of Chicago, Illinois, 2019. URL https://geodacenter.github.io/data-and-lab/. [p213]

GEOSTAT. ESSnet Project GEOSTAT 1A – Development of an European Population Grid Dataset. Technical report, The European Forum for GeoStatistics, 2011. URL `https://www.efgs.info/geostat/1a/`. [p210]

GEOSTAT. ESSnet Project GEOSTAT 1B – Representing 2011 Census Data on Grid. Technical report, The European Forum for GeoStatistics, 2014. URL `https://www.efgs.info/geostat/1b/`. [p210]

P. Hendricks. *anonymizer: Anonymize Data Containing Personally Identifiable Information*, 2015. URL `https://github.com/paulhendricks/anonymizer`. R package version 0.2.0. [p209]

INSPIRE. INSPIRE Specification on Geographical Grid Systems – Guidelines (D2.8.I.2). Technical Report March, INSPIRE Infrastructure for Spatial Information in Europe: European Commission, 2010. URL `http://inspire.ec.europa.eu/documents/Data{_}Specifications/INSPIRE{_}Specification{_}GGS{_}v3.0.1.pdf`. [p210, 214, 221]

R. Lagonigro, R. Oller, and J. C. Martori. A Quadtree Approach Based on European Geographic Grids: Reconciling Data Privacy and Accuracy. *SORT*, 41(1), 2017. ISSN 20138830. doi: https://doi.org/10.2436/20.8080.02.55. [p210]

R. Lagonigro, R. Oller, and J. C. Martori. *AQuadtree: Confidentiality of Spatial Point Data*, 2020. URL `https://cran.r-project.org/package=AQuadtree`. R package version 1.0.1. [p210]

N. Li, T. Li, and S. Venkatasubramanian. t-closeness: Privacy beyond k-anonymity and l-diversity. In *2007 IEEE 23rd International Conference on Data Engineering*, pages 106–115. IEEE, 2007. doi: https://doi.org/10.1109/ICDE.2007.367856. [p209]

A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam. l-diversity: Privacy beyond k-anonymity. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1):3–es, 2007. doi: https://doi.org/10.1145/1217299.1217302. [p209]

D. Marcelino. *SciencesPo: A Tool Set for Analyzing Political Behaviour Data*, 2013. R package version 1.4.1. [p210]

D. Martin. Geography for the 2001 Census in England and Wales. *Population Trends*, Summer(108): 7–15, 2002. [p218]

C. M. O'Keefe and D. B. Rubin. Individual Privacy versus Public Good: Protecting Confidentiality in Health Research. *Statistics in medicine*, 34(23):3081–3103, 2015. doi: https://doi.org/10.1002/sim.6543. [p209]

E. Pebesma and R. S. Bivand. Classes and Methods for Spatial Data in R. *R News*, 5(2):9–13, nov 2005. URL `https://cran.r-project.org/doc/Rnews/`. [p212, 217]

H. Samet. The Quadtree and Related Hierarchical Data Structures. *ACM Computing Surveys*, 16 (2):187–260, 1984. ISSN 03600300. doi: https://doi.org/10.1145/356924.356930. [p210]

L. Sweeney. k-anonymity: A Model for Protecting Privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(05):557–570, 2002. doi: https://doi.org/10.1142/S0218488502001648. [p209]

M. Tammilehto-Luode. Opportunities and challenges of grid-based statistics. In *World Statistics Congress of the International Statistical Institute*, Dublin, Ireland, 2011. [p209]

M. Tammilehto-Luode, L. Backer, and L. Rogstad. Grid Data and Area Delimitation by Definition Towards a better European Territorial Statistical System. *Statistical Journal of the United Nations Economic Commission for Europe*, 17(2):109–117, 2000. doi: https://doi.org/10.3233/SJU-2000-17202. [p209]

M. Templ, A. Kovarik, and B. Meindl. Statistical Disclosure Control for Microdata Using the R Package sdcMicro. *Journal of Statistical Software*, 67(4), 2015. doi: https://doi.org/10.18637/jss.v067.i04. [p210]

H. Theil. *Statistical decomposition analysis.* Amsterdam: North Holland, 1972. [p211]

K. Vu, R. Zheng, and J. Gao. Efficient Algorithms for k-anonymous Location Privacy in Participatory Sensing. In *2012 Proceedings IEEE INFOCOM*, pages 2399–2407. IEEE, 2012. doi: https://doi.org/10.1109/INFCOM.2012.6195629. [p209, 211]

H. Wickham, R. Francois, L. Henry, and K. Müller. *dplyr: A Grammar of Data Manipulation*, 2020. URL https://CRAN.R-project.org/package=dplyr. R package version 0.8.4. [p212]

D. L. Zimmerman and C. Pavlik. Quantifying the Effects of Mask Metadata Disclosure and Multiple Releases on the Confidentiality of Geographically Masked Health Data. *Geographical Analysis*, 40 (1):52–76, 2008. ISSN 00167363. doi: https://doi.org/10.1111/j.0016-7363.2007.00713.x. [p209]

D. L. Zimmerman, M. P. Armstrong, and G. Rushton. Alternative Techniques for Masking Geographic Detail to Protect Privacy. In *Geocoding Health Data: The Use of Geographic Codes in Cancer Prevention and Control, Research and Practice*, chapter Chapter 7, pages 127–138. CRC Press, Boca Raton, Fla., 2007. ISBN 978-0-8493-8419-6. [p209]

Raymond Lagonigro
Department of Economics and Business
Data Analysis and Modeling Research Group
University of Vic - UCC
Spain
(ORCiD: 0000-0002-8091-4296)
raymond.lagonigro@uvic.cat

Ramon Oller
Department of Economics and Business
Data Analysis and Modeling Research Group
University of Vic - UCC
Spain
(ORCiD: 0000-0002-4333-0021)
ramon.oller@uvic.cat

Joan Carles Martori
Department of Economics and Business
Data Analysis and Modeling Research Group
University of Vic - UCC
Spain
(ORCiD: 0000-0002-8400-5487)
martori@uvic.cat

miWQS: Multiple Imputation Using Weighted Quantile Sum Regression

by Paul M. Hargarten and David C. Wheeler

Abstract The **miWQS** package in the Comprehensive R Archive Network (CRAN) utilizes weighted quantile sum regression (WQS) in the multiple imputation (MI) framework. The data analyzed is a set/mixture of continuous and correlated components/chemicals that are reasonable to combine in an index and share a common outcome. These components are also interval-censored between zero and upper thresholds, or detection limits, which may differ among the components. This type of data is found in areas such as chemical epidemiological studies, sociology, and genomics. The **miWQS** package can be run using complete or incomplete data, which may be placed in the first quantile, or imputed using bootstrap or Bayesian approach. This article provides a stepwise and hands-on approach to handle uncertainty due to values below the detection limit in correlated component mixture problems.

Introduction

When studying public health, researchers want to determine if a set/mixture of continuous and correlated components/chemicals is associated with an outcome and if so, which components are important in that mixture (Braun et al., 2016). These components share a common univariate outcome but are interval-censored between zero and low thresholds, or detection limits, that may be different across the components.

We have created the **miWQS** package to analyze epidemiological studies with chemical exposures, but researchers may also apply the package to public health, genomics, or other areas in public health and medicine. Epidemiologists examine chemical mixtures because human exposure to a large number of chemicals may increase the risk of disease (Braun et al., 2016). Researchers may also create a socioeconomic status (SES) index that is generally composed of continuous correlated variables in the following domains: educational achievement, race, income, housing, and employment (Wheeler et al., 2017, 2019a). For example, race may be represented by percent of the population that is white. There are several examples of this in the literature (Wheeler et al., 2019b, 2020). Although these variables may have missing values throughout the distribution, researchers may use the **miWQS** package to create SES index even in the presence of missing data. Alternatively, genome-wide association studies (GWAS's) analyze DNA sequence variation using single nucleotide polymorphisms (SNPs) (Bush and Moore, 2012). As SNPs constitute high-frequency changes of a single base in the DNA sequence throughout the genome, SNPs serve as markers of a genomic region (Bush and Moore, 2012). Thus, SNPs are highly correlated (Bush and Moore, 2012; Ferber and Archer, 2015). The research aim of a GWAS is to find associations between genes and common and complex diseases like schizophrenia and to identify specific associated genes. The **miWQS** package can answer this research aim while simultaneously accounting for the correlation between SNPs.

In the data, an approach to account for the correlation among completely observed components is the weighted quantile sum (WQS) regression (Carrico et al., 2014; Czarnota et al., 2015b; Gennings et al., 2013). The application of WQS regression to censored data has been limited statistically and computationally on CRAN (the Comprehensive R Archive Network) (Czarnota et al., 2015a; Horton et al., 2015; Czarnota and Wheeler, 2015; Renzetti et al., 2020). In order to fully account for the uncertainty due to censoring, the **miWQS** package utilizes WQS regression in the multiple imputation (MI) framework (Hargarten and Wheeler, 2020, 2021).

As compared to other WQS packages in R, the **miWQS** package is specifically designed to use highly correlated data that include interval-censoring. The **wqs** (Czarnota and Wheeler, 2015) package performs WQS regression only on complete mixtures that share a continuous or binary outcome. The `wqs.est()` function in the **wqs** package can be used for continuous outcomes and displays an error if fed incomplete information. The `gwqs()` function in the **gWQS** package runs WQS regression when the outcome is continuous, binary, binomial, multinomial, or a count. If incomplete components are inputted into `gwqs()`, the function uses non-missing data without warning (Renzetti et al., 2020). By contrast, the **miWQS** functions are constructed to handle both complete and incomplete mixture data that share a continuous, binary, or count outcome by using MI.

The MI approach provides valid statistical inference in estimating regression parameters when data are missing (Dong and Peng, 2013; Rubin, 1987; White et al., 2011). Specifically, MI consists of three stages: (1) imputation, (2) analysis, and (3) pooling (Figure 1). First, we create several imputed datasets by replacing the below the detection limit (BDL) values by plausible data values. The complete

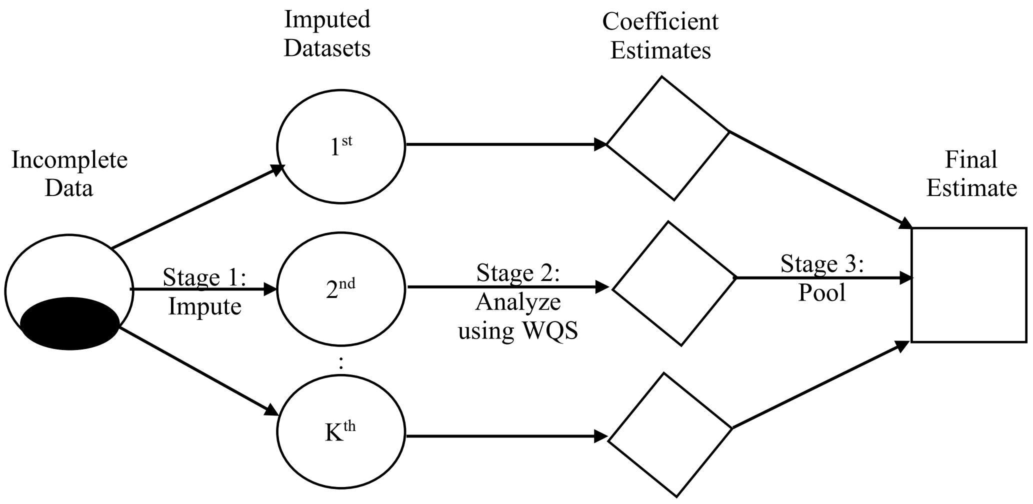


Figure 1: Multiple Imputation in connection with the Weighted Quantile Sum regression (MI-WQS). Given partially observed correlated chemical exposures that share a common outcome and covariates, (stage 1) researchers impute the below detection limit values (dark circles) K times to form complete datasets. In stage 2, each imputed dataset is analyzed using WQS regression. In stage 3, the coefficient estimates from the K WQS regressions (diamonds) are combined into a final estimate (square).

datasets are identical for the observed data but are different in the imputed values. Second, we analyze each complete dataset using WQS regression to obtain estimates (Carrico et al., 2014; Czarnota et al., 2015b; Gennings et al., 2013; Hargarten and Wheeler, 2020). Lastly, we combine each WQS estimate from different analyses to form one final estimate, to find its variance, and to perform statistical tests in order to determine the significance of the exposure effects.

Other MI packages in R have functions that combine estimates, but these are different than the `pool.mi()` function used in the **miWQS** package. The **mice** (multiple imputation by chained equations) package implements a strategy to impute multivariate missing data using fully conditional densities (van Buuren and Groothuis-Oudshoorn, 2011). Its pool function combines one estimate at a time, while `pool.mi()` combines all estimates simultaneously. The **norm** package allows users to impute values with an assumed multivariate normal distribution (Novo and Schafer, 2013). Its pool function, `mi.inference()`, does not allow the user to adjust the degree of freedom due to small sample sizes in contrast to `pool.mi()`. The **mi** package performs multiple imputation with missing values and saves the results as a `mi-class` object (Su et al., 2011). As a `mi-class` object is used to pool estimates inside the **mi** package, we cannot use it to pool estimates obtained in other packages.

Contrasting with the other packages on CRAN, the purpose of the **miWQS** package is to find an association of interval-censored mixture data with an outcome. The **miWQS** package can be run using complete or incomplete data. Incomplete data may be placed in the first quantile of the index or imputed using bootstrap or Bayesian approach. In this vignette, we will discuss how the data are formatted and then answer the research objectives using the **miWQS** package in four different ways: (1) with complete data, (2) with incomplete data placed in the first quantile, (3) with incomplete data imputed by bootstrapping, and (4) with incomplete data by using a Bayesian approach.

Data structure

This section describes what the data should look like in order to use the **miWQS** package. We wish to assess the association of the mixture of components, $\boldsymbol{X}$, and a univariate outcome, y, while accounting for other covariates, $\boldsymbol{Z}$. However, the continuous non-detects in the mixture ($\boldsymbol{X}$) are interval-censored between zero and different detection limits DL. Any missing values in the covariates or outcome are ignored and removed before imputation and analysis. Although $\boldsymbol{X}$ may refer to a variable with no obvious DL, we consider chemical concentrations $\boldsymbol{X}$ with each being partially observed in this vignette.

Our example demonstrating the use of the **miWQS** package is the provided dataset, `simdata87`. It is a list that consists of: 14 non-missing chemical concentrations, 14 chemical concentrations with each having 10% missing, 14 detection limits, a binary outcome representing cancer diagnosis, and three covariates. The dataset was generated as part of a simulation study with 1,000 subjects (Hargarten and Wheeler, 2020).

After installing the R package **miWQS** from CRAN, load the package and the dataset as follows.

```
> library("miWQS")

Loading required package: parallel

> data("simdata87")
```

The numeric components of interest to combine into an index $\boldsymbol{X}$ are stored in a matrix or a data frame. Any missing values in $\boldsymbol{X}$ are denoted by NA's and are assumed to be censored between zero and an upper threshold, *DL*. The *DL* is a numeric vector, where each element represents the detection limit (DL) for each chemical. In order to use the imputation techniques in **miWQS**, each chemical must have a known DL, or an upper bound. Otherwise, chemical values are placed in the first quantile (BDLQ1) of the weighted index (see Example 2). For instance, 14 non-missing chemical concentrations are saved as columns in a matrix `simdata87$X.true`. The matrix `simdata87$X.bdl` contains these 14 chemical concentrations, but 100 values are subbed as missing for each chemical between zero and different detection limits. These detection limits are saved in element `DL` of `simdata87` and are printed below along with their chemical names.

```
> simdata87$DL

  alpha-chlordane          dieldrin   gamma-chlordane           lindane
        0.9244609         4.4464426        29.1202898         8.2705681
     methoxychlor               dde               ddt pentachlorophenol
       41.3440690         2.3958978         4.5525251         5.1020673
          pcb_105           pcb_118           pcb_138           pcb_153
        1.6490457         1.9822575         1.2512259         0.7401736
          pcb_170           pcb_180
        3.3034084         1.0357342
```

A heat map of the observed logarithmic chemical concentrations (`simdata87$X.bdl`) shows the correlations among the components in our dataset (Figure 2). The **miWQS** package handles such correlated component data to examine whether the mixture is associated with the outcome.

```
>
> GGally::ggcorr(
+   log(simdata87$X.bdl),
+   method = c("pairwise", "spearman"),
+   geom = "tile",
+   layout.exp = 2,
+   hjust = 0.75,
+   size = 3,
+   legend.position = "bottom"
+ )
```

Chemical exposure patterns often differ between individuals due to demographics and other confounders. The additional covariates $\boldsymbol{Z}$ can be represented as a vector, data frame, or matrix. For example, the element `Z.sim` in the list `simdata87` is a matrix that contains an individual's age, sex (Female/Male), ethnicity (Hispanic/Non-Hispanic), and race (White/non-White). Some statistics of the covariates are shown below.

```
> summary(simdata87$Z.sim[, "Age"])

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.0224  2.4909  3.7443  3.7176  4.8805  7.9771

> apply(simdata87$Z.sim[, -1], 2, table)

  Female Hispanic Non-Hispanic_Others
0    611      670                 766
1    389      330                 234
```

The univariate outcome shared among the components, y, may be continuous, count-based, or binary; it is represented as a numeric vector or a factor in R. The mean of the outcome, ξ, relates the

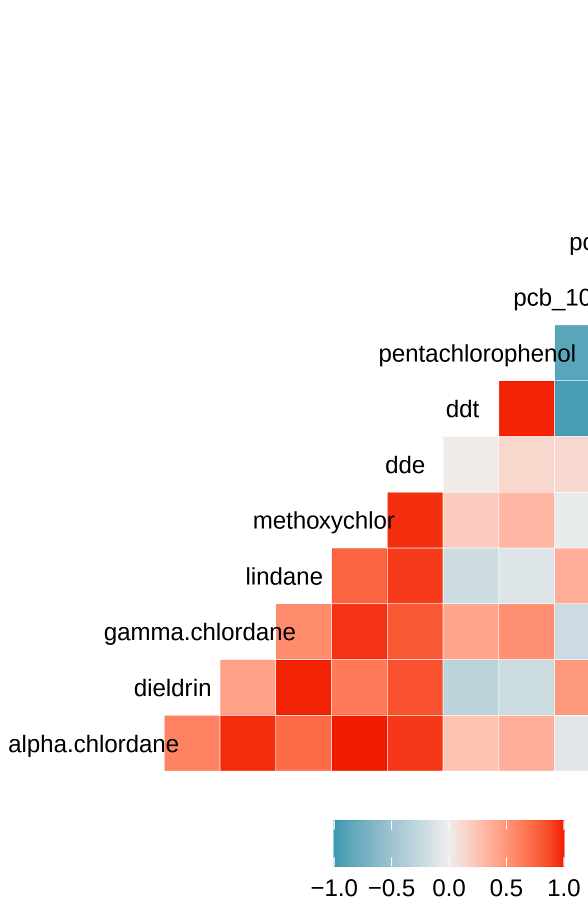


Figure 2: Heat map of the correlations using the fourteen observed chemical logarithmic concentrations in dataset simdata87 can be analyzed with the package miWQS. The heat map was generated using the GGally package.

covariates and chemicals by a link function *g()* as in generalized linear models. Continuous, count-based, and binary outcomes all commonly arise in public health and medicine. First, exposure to a mixture of chemicals may be associated with continuous health outcomes, such as body mass index (BMI), systolic blood pressure, or cholesterol. When y is continuous, we assume a Gaussian distribution using an identity link. Next, count health outcomes may arise in evaluations of socioeconomic data or environmental exposures in census regions. When y is a count, we assume a Poisson distribution with a log link and use an offset if a rate is modeled. Finally, binary health outcomes are common in environmental exposure data and in case-control studies. When y is binary, we assume a Bernoulli distribution using a logistic link. In our dataset, the `y.scenario` element of `simdata87` is binary. Suppose that `y.scenario` consists of cancer cases (represented by 1) and controls (represented by 0). The table below shows that 457 individuals (45.7%) are diagnosed with cancer.

```
> cat("Counts")
> table(simdata87$y.scenario)

Counts
  0   1
543 457
```

In our dataset–`simdata87`–we will like to answer the following research questions: (1) Is the mixture of correlated chemicals associated with cancer; (2) if so, what are the important chemicals? In the examples that follow, we will use both non-missing and missing chemical concentrations that are handled in four different ways.

Example 1: WQS regression using complete data

WQS regression allows us to estimate the effect of a chemical mixture on the disease while parsimoniously selecting important components (Carrico et al., 2014; Czarnota et al., 2015b; Gennings et al., 2013; Hargarten and Wheeler, 2020). Briefly, WQS regression was designed to select components in environmental exposure analysis. The correlated components are scored into quantiles. Let q_{ij} represent the values of the jth chemical exposed in the ith subject. Ideally, the data should be randomly split into a training dataset and validation dataset. While the training set is used to create the WQS index, the validation dataset is used to assess the association of the weighted index with the outcome. Yet, small datasets should not be split as splitting them may result in inadequate power to detect a signal.

In the training dataset, the weights are estimated from B bootstrapped samples of size n_T to form the weighted index. Each bootstrap sample is used to estimate the unknown weights w_j that maximize the likelihood in the following nonlinear model:

$$g(\xi_i) = \beta_{0b}^{(T)} + \beta_{1b}^{(T)} \cdot \left(\sum_{j=1}^{c} w_{jb} \cdot q_{ij} \right) + z'_{ib} \cdot \boldsymbol{\theta}^{(T)},$$

subject to

$$\beta_{1b}^{(T)} > 0,\ 0 \le w_{jb} \le 1, \text{ and } \sum_{j=1}^{c} w_{jb} = 1$$

for the b^{th} bootstrap sample. The parameters are as follows: $\beta_{0b}^{(T)}$ is the intercept, $\beta_{1b}^{(T)}$ is the overall mixture effect, and $\boldsymbol{\theta}$ are the covariate parameters. The term $\left(\sum_{j=1}^{c} w_{jb} \cdot q_{ij}\right)$ represents the weighted index of the c chemicals of interest. The parameters in the training dataset are represented with superscript T. The final weight estimate $\bar{w}_j$ is calculated as an average of the bootstrap estimates $\hat{w}_{jb}$ for the jth chemical:

$$\bar{w}_j = \frac{1}{B} \sum_{b=1}^{B} \hat{w}_{jb}.$$

A constraint is placed on $\beta_{1b}^{(T)}$ to allow for interpretation of the index (Carrico et al., 2014). Often, exploratory single-chemical analyses, shown in Appendix 1, show that some components in the mixture have a negative association with the outcome, while others have a positive association. In environmental risk analysis, researchers are often interested in a positive association between the mixture of components and an adverse health outcome. However, if a researcher hypothesizes that the overall mixture is protective of the outcome, the constraint $\beta_{1b}^{(T)} > 0$ should be switched to $\beta_{1b}^{(T)} < 0$.

Then, the weighted quantile index score of the i^{th} individual is specified as: $WQS_i = \sum_{j=1}^{c} \bar{w}_j \cdot q_{ij}$, which uses the quantiles in the validation dataset. In the validation dataset, the significance of the WQS parameter ($\beta_1^{(V)}$) can be determined from:

$$g(\xi_i) = \beta_0^{(V)} + \beta_1^{(V)} WQS_i + z'_i \cdot \boldsymbol{\theta}^{(V)},$$

where superscript V represents the regression coefficients in the validation dataset. While $\beta_1^{(V)}$ describes the effect of the chemical mixture on the health outcome, the mean weight $\bar{w}_j$ identifies the relative importance that chemical j imposes on the outcome (Carrico et al., 2014; Czarnota et al., 2015b; Gennings et al., 2013; Hargarten and Wheeler, 2020).

The `estimate.wqs()` function performs WQS regression in the **miWQS** package. The data as specified in Data structure section are placed in the first three arguments. The `y` argument takes the outcome, like `simdata87$y.scenario`. As `y.scenario` is binary, the binomial distribution is specified by setting the `family` argument to `"binomial"`. The `X` argument takes the chemicals of interest, like `simdata87$X.true`. If `X` contains `NA`'s (that represents missing values), the BDL values are placed in the first quantile by default (see Example 2). Any additional demographic covariates, like `simdata87$Z.sim`, are placed into the `Z` argument. If no covariates are present, set `Z` to `NULL`. The `b1.pos` argument controls whether the overall mixture effect, $\beta_1^{(T)}$, is positively related to the outcome. A way to decide the direction is to use the `analyze.individually()` function, which is described in more detail in Appendix 1. In our dataset, we assume a positive relationship between the mixture and an outcome; we consequently set `b1.pos` to `TRUE`. The `proportion.train` argument specifies the proportion of data given to the training dataset. As the sample size of our example dataset is large (n = 1000), we will use 50% of the data to train. The `B` argument is the number of bootstraps used to estimate the weights w_j's.

We set a seed to ensure reproducibility as we bootstrapped the data. The execution of the `estimate.wqs()` function creates an object of class `wqs`, and printing it answers the main research questions.

```
> set.seed(50679)
> wqs.eg1 <- estimate.wqs(
+   y = simdata87$y.scenario, X = simdata87$X.true, Z = simdata87$Z.sim,
+   proportion.train = 0.5,
+   n.quantiles = 4,
+   place.bdls.in.Q1 = FALSE,
+   B = 100,
+   b1.pos = TRUE,
+   signal.fn = "signal.converge.only",
+   family = "binomial",
+   verbose = FALSE
+ )

#> No missing values in matrix detected. Regular quantiles computed.

> wqs.eg1


Odd Ratios & 95% CI (N.valid = 500)
                     Odds Ratio  SE.OR                95% CI  P-value
(Intercept)               0.142   1.51  0.142 (0.063, 0.320)   <0.001
Age                       0.950   1.06  0.950 (0.854, 1.056)    0.339
Female                    0.947   1.22  0.947 (0.646, 1.388)    0.780
Hispanic                  1.580   1.23  1.578 (1.059, 2.352)    0.025
Non.Hispanic_Others       1.030   1.25  1.034 (0.671, 1.593)    0.880
WQS                       3.660   1.25  3.663 (2.372, 5.659)   <0.001
AIC:  660.7468

All (100) bootstraps have converged.

 Weights Adjusted by signal.converge.only using N.train = 500 observations:
             ddt           pcb_105           pcb_170           pcb_138
          0.3905            0.2413            0.1105            0.1014
         pcb_153               dde           pcb_118  pentachlorophenol
          0.0344            0.0339            0.0217            0.0216
         lindane   gamma.chlordane       methoxychlor    alpha.chlordane
          0.0200            0.0138            0.0043            0.0028
         pcb_180          dieldrin
          0.0024            0.0014
Important chemicals defined as mean weights > 1/14~0.071.
```

An increase in the chemical mixture is associated with an increase in the odds of being diagnosed with cancer by 3.66. The `coef(wqs.eg1)` gives us estimates on the logit scale of coefficients in the validation model. We identify chemicals in the mixture as important if their weight estimates are greater than the reciprocal of the number of chemicals. Alpha-chlordane, PCB 153, PCB 105, and p,p-DDE approximately constitute 88% of the effect in the index. Thus, these three chemicals are associated with increased cancer risk. The weight estimates are directly extracted with `wqs.eg1$processed.weights`. The Akaike information criterion (AIC) is used as the goodness-of-fit measure of the WQS model and is directly computed using `AIC(wqs.eg1$fit)`.

Plotting a WQS object gives a list of histograms: the distributions of the weight estimates, the overall effect of the mixture, and the WQS index score (Wickham, 2016).

```
> eg1.plots <- plot(wqs.eg1)
> names(eg1.plots)

[1] "hist.weights" "hist.beta1"   "hist.WQS"
```

Commonly, researchers look at distributions of the weight estimates to determine which chemicals are important in the mixture (Figure 3). Looking at the histograms for complete WQS data, most chemicals have no effect among all bootstraps. However, this panel of histograms indicates that alpha-chlordane, p,p-DDE, PCB 153, and PCB 105 are important, which agrees with our above statistical analysis.

The second histogram provides us insight into the distribution of the overall effect of the mixture on the outcome, $\beta_1^{(T)}$, across the bootstraps (Figure 4). Most bootstraps indicate that the chemical mixture is not associated with the outcome (median around 1).

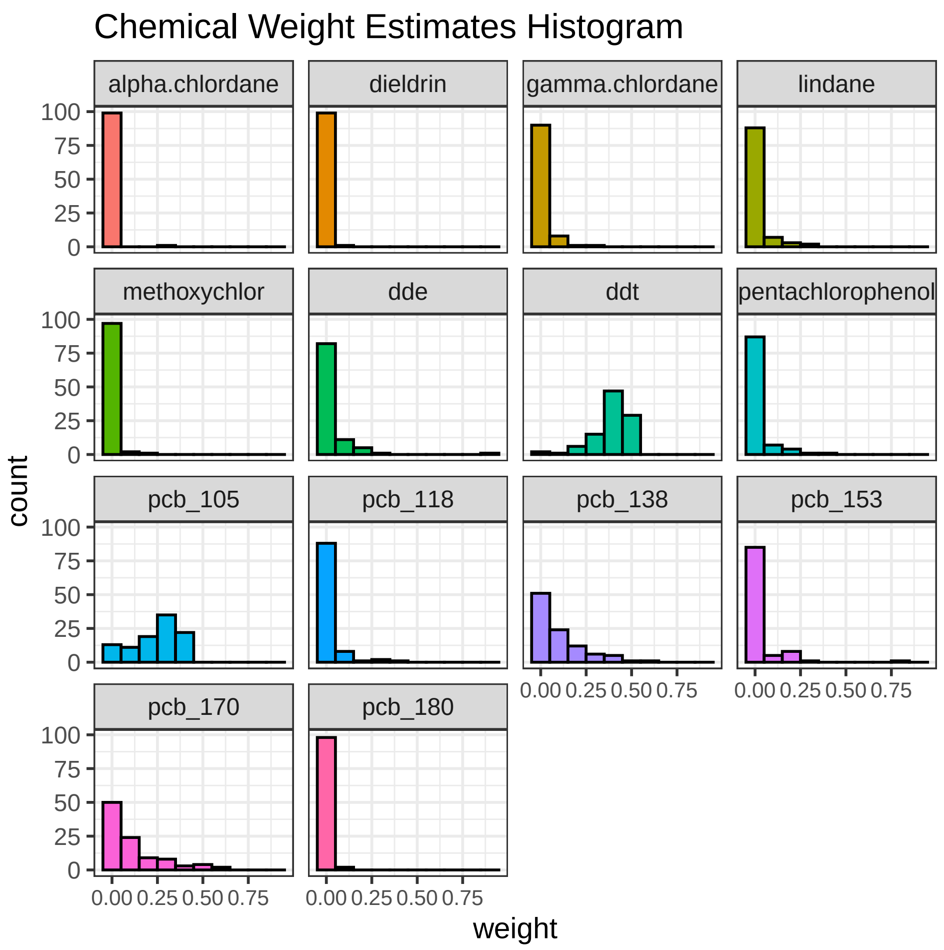


Figure 3: Histograms of chemical weight estimates across 100 bootstraps for Example 1 to select important chemicals. Weight estimates are constrained to be between zero and one.

The third histogram shows us the distribution of the weighted quantile sum. Given constraints placed on the weights, WQS is a continuous index between zero and the number of quantiles minus 1 (given by the `n.quantiles` argument in `estimate.wqs()`) (Figure 5). In our example, the number of quantiles is four. Across the bootstrap samples, most values of the chemical mixture are between one and two.

Example 2: BDLQ1 approach on interval-censored data

BDLQ1 approach

Unlike Example 1, many studies contain partially observed chemical concentrations that are measured to different detection limits. One approach to use WQS with missing data is to place the BDL values into the first quantile (BDLQ1), and to score the observed component values in the remaining quantiles. The `make.quantile.matrix()` function demonstrates this approach by creating `n.quantiles` quantiles from a matrix argument `X`. If `X` is completely observed, regular quantiles are made; however, if the first values in `X` are missing, they are placed in the first quantile. For example, suppose we are interested in making four quantiles of 14 chemicals using 1,000 subjects in our dataset. If we use the completely observed concentrations found in `X.true` element of `simdata87`, regular quantiles for all 14 chemicals are made with the following number of individuals per quantile.

```
> q <- make.quantile.matrix(
+   X = simdata87$X.true,
+   n.quantiles = 4
+   )
```

```
#> No missing values in matrix detected. Regular quantiles computed.
```

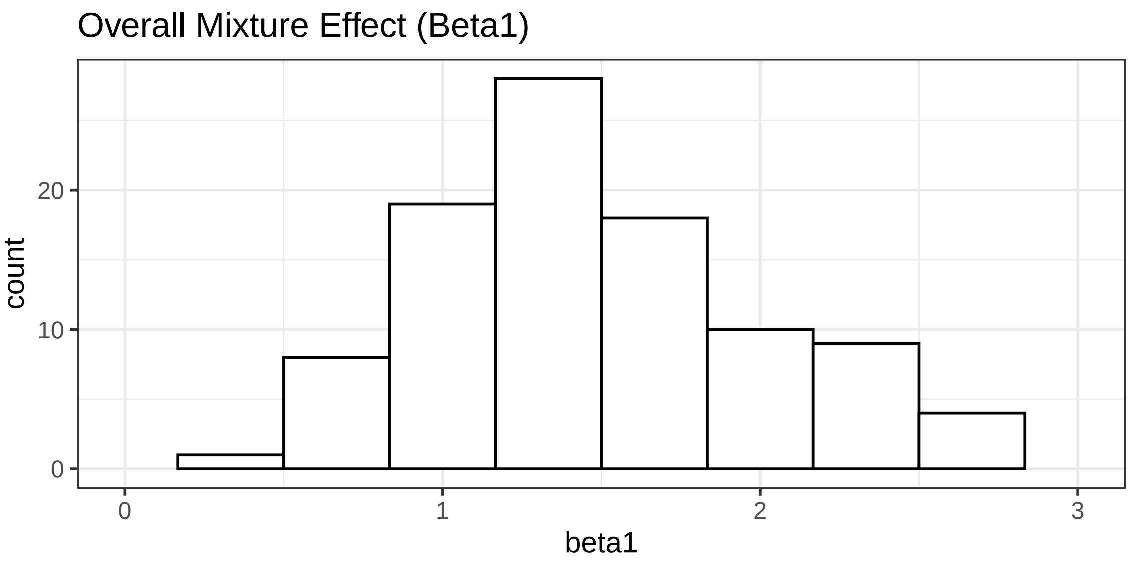


Figure 4: Histogram of overall chemical effect in the training dataset across 100 bootstraps for Example 1. Its constraint is governed by the b1.pos argument in the estimate.wqs() function. In the simdata87 dataset, the overall mixture is constrained to have a positive association with cancer. Most bootstraps indicate that the chemical mixture is not associated with the outcome.

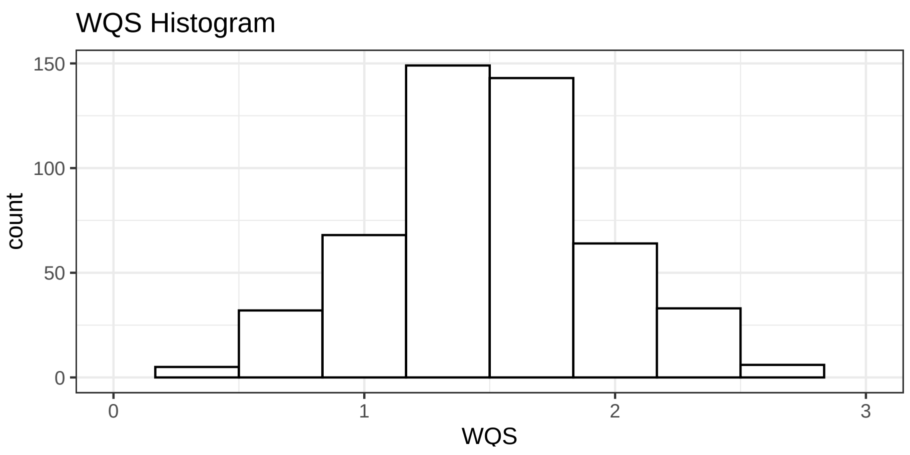


Figure 5: Histogram of the weighted quantile sum (WQS) using validation quantiles for Example 1 to show where most values of the chemical mixture lie.

```
> apply(q, 2, table)

  [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14]
0  250  250  250  250  250  250  250  250  250   250   250   250   250   250
1  250  250  250  250  250  250  250  250  250   250   250   250   250   250
2  250  250  250  250  250  250  250  250  250   250   250   250   250   250
3  250  250  250  250  250  250  250  250  250   250   250   250   250   250
```

However, if the chemical concentrations are incomplete (with the missing values indicated as `NA`'s), the BDLQ1 approach works as follows. Suppose we wish to make quartiles of the `X.bdl` matrix in our dataset, where each chemical has 100 BDL concentrations. Using BDLQ1, the 100 observations are placed into the first quartile, and the remaining quartiles are evenly split in which each contains 900/3 = 300 observations. The number of individuals in each quartile of each chemical, and the total number of missing values in each chemical are shown below. Note that the first row of the matrix matches the total number of missing values (100).

```
> q <- make.quantile.matrix(
+   simdata87$X.bdl,
+   n.quantiles = 4,
+   verbose = TRUE
+   )
```

```
#> All BDLs are placed in the first quantile

##> Summary of Quantiles
  [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14]
0  100  100  100  100  100  100  100  100  100   100   100   100   100   100
1  300  300  300  300  300  300  300  300  300   300   300   300   300   300
2  300  300  300  300  300  300  300  300  300   300   300   300   300   300
3  300  300  300  300  300  300  300  300  300   300   300   300   300   300
##> Total Number of NAs--Q1 (The first row) should match.
100 100 100 100 100 100 100 100 100 100 100 100 100 100
```

The number of individuals in the first quantile in BDLQ1 increases if more BDL values exist. For instance, `X.80` substitutes 800 values for each chemical from `simdata87$X.true` to be missing BDL. Applying the BDLQ1 approach to `X.80`, all 800 values are placed into the first quartile, while roughly $200/3 \approx 66$ values are placed in remaining quartiles.

```
> q <- make.quantile.matrix(X.80, n.quantiles = 4, verbose = TRUE)

#> All BDLs are placed in the first quantile

##> Summary of Quantiles
  [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14]
0  800  800  800  800  800  800  800  800  800   800   800   800   800   800
1   67   67   67   67   67   67   67   67   67    67    67    67    67    67
2   66   66   66   66   66   66   66   66   66    66    66    66    66    66
3   67   67   67   67   67   67   67   67   67    67    67    67    67    67
##> Total Number of NAs--Q1 (The first row) should match.
800 800 800 800 800 800 800 800 800 800 800 800 800 800
```

Instead of quantiles, we could also categorize the chemicals into deciles by changing the `n.quantiles` argument to ten. Suppose now that we wish to form deciles in `simdata87$X.bdl`. The first 100 BDL values are placed in the first decile, while the remaining 900 are evenly spread out in the remaining nine deciles (900/9 = 100).

```
> q <- make.quantile.matrix(simdata87$X.bdl, n.quantiles = 10, verbose = TRUE)

#> All BDLs are placed in the first quantile

##> Summary of Quantiles
  [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14]
0  100  100  100  100  100  100  100  100  100   100   100   100   100   100
1  100  100  100  100  100  100  100  100  100   100   100   100   100   100
2  100  100  100  100  100  100  100  100  100   100   100   100   100   100
3  100  100  100  100  100  100  100  100  100   100   100   100   100   100
4  100  100  100  100  100  100  100  100  100   100   100   100   100   100
5  100  100  100  100  100  100  100  100  100   100   100   100   100   100
6  100  100  100  100  100  100  100  100  100   100   100   100   100   100
7  100  100  100  100  100  100  100  100  100   100   100   100   100   100
8  100  100  100  100  100  100  100  100  100   100   100   100   100   100
9  100  100  100  100  100  100  100  100  100   100   100   100   100   100
##> Total Number of NAs--Q1 (The first row) should match.
100 100 100 100 100 100 100 100 100 100 100 100 100 100
```

The BDLQ1 method has been used in single-chemical analyses (Metayer et al., 2013; Ward et al., 2014, 2009) and WQS (Hargarten and Wheeler, 2020). However, it has not been coded in other WQS packages to the best of our knowledge.

WQS analysis

The BDLQ1 method works because WQS regression uses quantile scores from each chemical in the mixture. At this step, the `estimate.wqs()` function calls the `make.quantile.matrix()` function. Setting the argument `place.bdls.in.Q1` to `TRUE` allows us to use the WQS regression in conjunction with the BDLQ1 method. Yet, if the `X` argument contains any missing values, the BDLQ1 approach is automatically used. The incomplete data `X.bdl` is now assigned to the chemical mixture `X` argument. The remaining arguments in `estimate.wqs()` are the same as in Example 1. Printing the resulting object answers the research questions of interest. The research aims are to determine the association of the mixture with cancer and to find the important chemicals (if the association exists).

```
> set.seed(50679)
> wqs.BDL <- estimate.wqs(
+   y = simdata87$y.scenario, X = simdata87$X.bdl, Z = simdata87$Z.sim,
+   proportion.train = 0.5,
+   n.quantiles = 4,
+   place.bdls.in.Q1 = TRUE,
+   B = 100,
+   b1.pos = TRUE,
+   signal.fn = "signal.converge.only",
+   family = "binomial",
+   verbose = FALSE
+ )

#> All BDLs are placed in the first quantile

> wqs.BDL


Odd Ratios & 95% CI (N.valid = 500)
                     Odds Ratio  SE.OR                95% CI  P-value
(Intercept)               0.214   1.52  0.214 (0.094, 0.483)   <0.001
Age                       0.952   1.05  0.952 (0.858, 1.057)    0.356
Female                    0.926   1.21  0.926 (0.636, 1.349)    0.688
Hispanic                  1.560   1.22  1.558 (1.052, 2.306)    0.027
Non.Hispanic_Others       1.050   1.24  1.052 (0.687, 1.612)    0.816
WQS                       2.360   1.21  2.358 (1.626, 3.420)   <0.001
AIC:  677.0172

1 bootstrap(s) have failed to converged. Those are:
[1] 60

 Weights Adjusted by signal.converge.only using N.train = 500 observations:
         pcb_105  pentachlorophenol    gamma.chlordane    alpha.chlordane
          0.2575             0.2548             0.1622             0.0699
         pcb_153            pcb_138                ddt            lindane
          0.0658             0.0606             0.0378             0.0349
    methoxychlor            pcb_118                dde            pcb_170
          0.0180             0.0128             0.0075             0.0066
        dieldrin            pcb_180
          0.0065             0.0051
Important chemicals defined as mean weights > 1/14~0.071.
```

An increase in one-quartile of the chemical mixture is associated with an increase in the odds of obtaining cancer by 2.36. Compared to the complete case analysis, PCB 105 and alpha-chlordane are still important, but DDT, PCB 170, and methoxychlor are also important in the BDLQ1 analysis. As we forced some complete concentrations `simdata87$X.true` to be BDL values in creating `simdata87$X.bdl`, we used AIC to compare fit between the two WQS models in Examples 1 and 2. Intuitively, a WQS model using the BDLQ1 approach (AIC: 677) fits the data worse than a WQS model using complete data (AIC: 661).

Example 3: Bootstrapping interval-censored data

An alternative to the BDLQ1 approach is to perform multiple imputation of the missing chemical values by bootstrapping (Lubin et al., 2004). Given completely observed covariates $z_{i1}, \ldots z_{\text{ik}}$ in $i = 1, \ldots n$ subjects exposed to $j = 1, \ldots c$ chemicals, an independent log-normal distribution for each chemical j with mean μ_j and variance σ_j^2 is assumed:

$$\log(x_{ij}) | z_1 \cdots z_p \sim^{indep} N\left(\mu_j = z_i' \cdot \gamma_j, \sigma_j^2\right).$$

Let $f(.)$ denote the normal probability density function and $F(.)$ denote its cumulative distribution function. For each chemical j, the dataset is bootstrapped K times to form K complete datasets. As each bootstrap b is sampled with replacement from the original data, the number of times the i^{th} subject is selected for j^{th} chemical is represented by w_{ij}. The log likelihood function for the bootstrap data in j^{th} chemical is given by:

$$l(\gamma_j,\sigma_j^2) = \sum_{i=1}^{n_{0j}} w_{ij} * \log\left[P\left(0 < X_{ij} < DL_j; z_i' \cdot \gamma_j, \sigma_j^2\right)\right] + \sum_{i=n_{0j}+1}^{n} \log\left[f(x_{ij}; z_i' \cdot \gamma_j, \sigma_j^2)\right],$$

where n_{0j} represents the number of BDL values for chemical j. The estimates that maximize the log likelihood are $\left(\tilde{\gamma}_j, \tilde{\sigma}_j^2\right)$. Then, the BDL values are imputed by the following method. We generate an independent and identically distributed uniform sample between zero and $F\left(\log(DL_j); z_i' \cdot \tilde{\gamma}_j, \tilde{\sigma}_j^2\right)$. Then, we assign value $F^{-1}\left(u_{\text{ij}}\right)$ for each missing value x_{ij} below the detection limit of the j^{th} chemical DL_j (Lubin et al., 2004). These imputed values are joined with the observed ones to form one complete set of exposures for the j^{th} chemical. The `impute.Lubin()` function performs multiple imputation by bootstrapping for one chemical. For instance, suppose we wish to impute the dieldrin concentrations BDL twice ($K = 2$) in `simdata87` by bootstrapping using the following covariates: childhood age, sex, and child race/ethnicity. The dieldrin concentrations are found in the first column of `X.bdl` in `simdata87` dataset (e.g. `simdata87$X.bdl[ ,1]`), and the detection limit of dieldrin is in the first entry in `DL` element (e.g. `simdata87$DL[1]`). The `chemcol` argument is a numeric vector of chemical concentrations that we wish to impute (e.g. `simdata87$X.bdl[ ,1]`). The `dlcol` argument is the detection limit of the chemical (e.g. `simdata87$DL[1]`). The `Z` argument contains any covariates used in the imputation (e.g. `simdata87$Z.sim` and `simdata87$y.scenario`). We included the outcome in the imputation of BDL values because its omission assumes that it is not associated with the BDL values and thereby bias the subsequent WQS coefficients towards zero (Forer, 2014; Barnard et al., 2015). The `K` argument is the number of imputed datasets (e.g. 2).

```
> set.seed(472195)
> answer <- impute.Lubin(
+   chemcol = simdata87$X.bdl[, 1],
+   dlcol = simdata87$DL[1],
+   Z = cbind(simdata87$y.scenario, simdata87$Z.sim),
+   K = 2
+ )
> summary(answer$imputed_values)
```

```
     Imp.1              Imp.2
 Min.   :       0   Min.   :       0
 1st Qu.:      11   1st Qu.:      11
 Median :     125   Median :     125
 Mean   :   44099   Mean   :   44099
 3rd Qu.:    1682   3rd Qu.:    1682
 Max.   :17354723   Max.   :17354723
```

The `answer$imputed_values` is a matrix with rows of 1000 subjects and two columns consisting of the imputed dieldrin concentrations. Since most concentrations are observed, the summaries of the two datasets should look the same. However, if we look at BDL values, the two imputed datasets are different, and both are under the detection limit (0.924).

```
> cat("Summary of BDL Values \n")
> imp <- answer$imputed_values[, 1] < simdata87$DL[1]
> summary(answer$imputed_values[imp, ])

Summary of BDL Values
     Imp.1             Imp.2
 Min.   :0.00124   Min.   :0.001417
 1st Qu.:0.04579   1st Qu.:0.057819
 Median :0.22420   Median :0.201560
 Mean   :0.32314   Mean   :0.272618
 3rd Qu.:0.59102   3rd Qu.:0.444859
 Max.   :0.91690   Max.   :0.854974
```

More than one chemical often needs to be imputed in many studies. To implement the bootstrap approach, we use the `impute.boot()` function, which repeatedly executes the `impute.Lubin()` function. In `simdata87`, now suppose that we wish to impute the `X.bdl` matrix twice by bootstrapping using the covariates (`Z`) of age, sex, and race/ethnicity. The `X` argument takes a matrix with incomplete data, like `simdata87$X.bdl`. The next argument, `DL`, takes the detection limits of `X` as a numeric vector, like `simdata87$DL`. The `K` and `Z` arguments are exactly the same as in `impute.Lubin()`. A seed is set before the function to ensure that the same bootstrap samples are selected for each chemical. The function returns a list `l.boot`.

```
> set.seed(472195)
> l.boot <- impute.boot(
+   X = simdata87$X.bdl,
+   DL = simdata87$DL,
+   Z = cbind(simdata87$y.scenario, simdata87$Z.sim),
+   K = 2
+   )

#> Check: The total number of imputed values that are above the detection limit is 0.

> results.Lubin <- l.boot$X.imputed
```

The `X.imputed` element of `l.boot` saves the imputed chemical values as an array, where the first dimension is the number of subjects (n), the second is the number of chemicals (c), and the third is the number of imputed datasets (K). The sample minima, fifth percentile (`P.5`), means, and maxima of the chemicals are calculated in each imputed dataset (by the function `f()`). As the two imputed datasets are different, the application of MI should yield different parameter estimates.

```
> apply(results.Lubin, 2:3, f)

, , Imp.1

     alpha-chlordane     dieldrin gamma-chlordane   lindane methoxychlor
min     1.239744e-03 5.214163e-02        14.85745  3.122209     16.13658
P.5     2.257984e-01 2.016689e+00        25.79766  7.147067     35.19478
mean    4.409885e+04 4.331448e+02        49.38257 17.214769     83.45674
max     1.735472e+07 4.867330e+04       139.33689 75.360498    316.07141
              dde        ddt pentachlorophenol     pcb_105     pcb_118
min  4.500939e-02   1.532625          1.604183   0.1372503   0.4547588
P.5  8.736265e-01   3.544432          3.923535   1.1031301   1.4373202
mean 2.546607e+03  16.122825         20.262419  16.4815577   9.7025958
max  3.835674e+05 178.853960        285.229348 400.3601114 142.2619560
          pcb_138      pcb_153     pcb_170     pcb_180
min    0.08042499   0.03363274   0.7155792   0.1741525
P.5    0.80147195   0.47230663   2.5599230   0.7335900
mean  12.63094227  13.82089093  11.6125246   8.4248157
max  269.09903138 383.79850341 115.2060922 203.8593938

, , Imp.2

     alpha-chlordane     dieldrin gamma-chlordane   lindane methoxychlor
min     1.417111e-03     0.117659        14.57128  3.690147     15.35223
P.5     2.024978e-01     2.276511        25.50909  6.751412     36.11395
mean    4.409884e+04   433.153356        49.36521 17.201076     83.39251
max     1.735472e+07 48673.296171       139.33689 75.360498    316.07141
              dde         ddt pentachlorophenol     pcb_105     pcb_118
min  3.502864e-02   0.9292443          1.299465   0.1512212   0.4144885
P.5  9.079915e-01   3.3656892          3.959147   1.0578307   1.3154884
mean 2.546614e+03  16.1134725         20.250618  16.4788855   9.6934267
max  3.835674e+05 178.8539599        285.229348 400.3601114 142.2619560
         pcb_138      pcb_153     pcb_170      pcb_180
min    0.2291489   0.08258941   0.4825019   0.05475401
P.5    0.8403295   0.52882557   2.5113298   0.69996133
mean  12.6359530  13.82390718  11.6064361   8.42308671
max  269.0990314 383.79850341 115.2060922 203.85939377
```

Next, we implement WQS regression on the two complete datasets, which are saved in the `results.Lubin` object. Instead of performing WQS on one dataset as in Examples 1 and 2, the `do.many.wqs()` function repeatedly executes WQS regression on each dataset. The arguments for the `do.many.wqs()` function are the same as the `estimate.wqs()` function, with one exception. The `X.imputed` argument now is an array of the imputed chemical values, which has three dimensions: n subjects, c chemicals, and K imputed datasets. This array is the output from the `impute.boot()` function: `results.Lubin`.

```
> set.seed(50679)
> boot.wqs <- do.many.wqs(
+   y = simdata87$y.scenario, X.imputed = results.Lubin, Z = simdata87$Z.sim,
+   proportion.train = 0.5,
+   n.quantiles = 4,
+   B = 100,
+   b1.pos = TRUE,
+   signal.fn = "signal.converge.only",
+   family = "binomial"
+ )

#> Sample size: 1000; Number of chemicals: 14;
Number of completed datasets: 2; Number of covariates modeled:  4
```

The do.many.wqs() function returns list and matrix versions of the output generated from the estimate.wqs() function. The wqs.imputed.estimates element of the boot.wqs list is a three-dimensional array that gives the WQS estimates for each imputed dataset. The first dimension consists of the total number parameters in the WQS model. The second dimension consists of two columns: the mean and standard deviation of estimates. The third dimension is the *K* imputation draws.

```
> formatC(boot.wqs$wqs.imputed.estimates, format = "fg", flag = "#", digits = 3)

, , Imputed.1

                    Estimate  Std.Error
alpha.chlordane     "0.00285" "0.0285"
dieldrin            "0.00139" "0.0101"
gamma.chlordane     "0.0138"  "0.0442"
lindane             "0.0200"  "0.0525"
methoxychlor        "0.00429" "0.0244"
dde                 "0.0339"  "0.104"
ddt                 "0.391"   "0.0936"
pentachlorophenol   "0.0216"  "0.0628"
pcb_105             "0.241"   "0.129"
pcb_118             "0.0217"  "0.0697"
pcb_138             "0.101"   "0.131"
pcb_153             "0.0344"  "0.0986"
pcb_170             "0.110"   "0.149"
pcb_180             "0.00236" "0.0114"
(Intercept)         "-1.95"   "0.415"
Age                 "-0.0516" "0.0540"
Female              "-0.0546" "0.195"
Hispanic            "0.456"   "0.204"
Non.Hispanic_Others "0.0333"  "0.221"
WQS                 "1.30"    "0.222"

, , Imputed.2

                    Estimate  Std.Error
alpha.chlordane     "0.00424" "0.0185"
dieldrin            "0.0392"  "0.0801"
gamma.chlordane     "0.00843" "0.0257"
lindane             "0.00191" "0.0119"
methoxychlor        "0.0103"  "0.0518"
dde                 "0.0767"  "0.114"
ddt                 "0.148"   "0.159"
pentachlorophenol   "0.272"   "0.163"
pcb_105             "0.156"   "0.114"
pcb_118             "0.0102"  "0.0302"
pcb_138             "0.213"   "0.193"
pcb_153             "0.0136"  "0.0513"
pcb_170             "0.0180"  "0.0526"
pcb_180             "0.0291"  "0.0625"
(Intercept)         "-1.70"   "0.363"
Age                 "0.0367"  "0.0539"
```

```
Female              "-0.199"  "0.192"
Hispanic            "0.577"   "0.200"
Non.Hispanic_Others "0.235"   "0.223"
WQS                 "0.762"   "0.163"
```

As expected, the weights and WQS parameter estimates are different across the two imputed datasets. Finally, the `pool.mi()` function implements the pooling rules discussed in Rubin 1987 (Rubin, 1987) in order to form one estimate (Dong and Peng, 2013; Rubin, 1987; White et al., 2011). The `to.pool` argument takes an array with rows referring to the number of parameters, columns referring to the mean and standard error, and the third dimension referring to the number of imputed datasets. This describes the WQS output, `boot.wqs$wqs.imputed.estimates`, from the `do.many.wqs()` function. The second argument of `pool.mi()`, `n`, is the sample size, which is the number of rows in original data (i.e. `nrow(simdata87$X.bdl)`). The additional Boolean argument `prt` allows the user to print out selective parts of the `pool.mi` object, if desired.

```
> boot.est <- pool.mi(
+   to.pool = boot.wqs$wqs.imputed.estimates,
+   n = nrow(simdata87$X.bdl),
+   prt = FALSE
+ )

#> Pooling estimates from 2 imputed analyses for 20 parameters.

                    pooled.mean pooled.total.se se.within se.between
alpha.chlordane           0.004           0.024     0.024      0.001
dieldrin                  0.020           0.066     0.057      0.027
gamma.chlordane           0.011           0.036     0.036      0.004
lindane                   0.011           0.041     0.038      0.013
methoxychlor              0.007           0.041     0.041      0.004
dde                       0.055           0.115     0.109      0.030
ddt                       0.269           0.247     0.130      0.172
pentachlorophenol         0.147           0.250     0.123      0.177
pcb_105                   0.198           0.143     0.122      0.061
pcb_118                   0.016           0.055     0.054      0.008
pcb_138                   0.157           0.191     0.165      0.079
pcb_153                   0.024           0.081     0.079      0.015
pcb_170                   0.064           0.137     0.112      0.065
pcb_180                   0.016           0.051     0.045      0.019
(Intercept)              -1.828           0.447     0.390      0.178
Age                      -0.007           0.094     0.054      0.062
Female                   -0.127           0.230     0.193      0.102
Hispanic                  0.517           0.228     0.202      0.086
Non.Hispanic_Others       0.134           0.282     0.222      0.143
WQS                       1.030           0.504     0.195      0.379
                    frac.miss.info   CI.1   CI.2 p.value
alpha.chlordane              0.005 -0.044  0.051   0.883
dieldrin                     0.327 -0.119  0.160   0.762
gamma.chlordane              0.019 -0.060  0.083   0.761
lindane                      0.181 -0.072  0.094   0.791
methoxychlor                 0.019 -0.073  0.087   0.859
dde                          0.125 -0.174  0.284   0.632
ddt                          0.836 -0.850  1.388   0.395
pentachlorophenol            0.859 -1.099  1.393   0.624
pcb_105                      0.359 -0.109  0.506   0.187
pcb_118                      0.037 -0.091  0.123   0.770
pcb_138                      0.337 -0.250  0.564   0.424
pcb_153                      0.057 -0.135  0.183   0.766
pcb_170                      0.454 -0.249  0.378   0.652
pcb_180                      0.273 -0.089  0.120   0.759
(Intercept)                  0.314 -2.770 -0.885   0.001
Age                          0.795 -0.373  0.358   0.943
Female                       0.392 -0.631  0.378   0.593
Hispanic                     0.276  0.044  0.989   0.034
Non.Hispanic_Others          0.509 -0.538  0.807   0.650
WQS                          0.919 -2.441  4.501   0.233
```

The `pool.mi()` function returns the statistics of the combined estimates for each WQS parameter. While the standard error between the imputed sets, `se.between`, measures the uncertainty due to the BDL values, the standard error within the imputed sets, `se.within`, measures the uncertainty in the WQS regression. Using the pooled mean and standard error, 95% t~confidence intervals are constructed in columns `CI.1` and `CI.2`. The p~values from the t~test whether the regression coefficient is zero are contained in the `p.value` column. The `frac.miss.info` column gives the fraction of missing information, which estimates the proportion of variability due to the BDL values for each WQS parameter. A larger fraction of missing information of any WQS parameter implies that we may need to increase the number of imputations (K). Yet, finding the optimal number of imputations remains an open area of research (Pan and Wei, 2016; Savalei and Rhemtulla, 2012). For instance, some covariates have high fractions of missing information, such as 0.73 or 0.85, which suggests that more than two imputations are needed.

The WQS `pooled.mean` estimate answers the question of whether a chemical mixture is associated with cancer. To find the odds ratio, we can exponentiate the estimate and its 95% confidence interval (CI) like: `exp(boot.est["WQS",c(1,7:8)])`. A one-quartile increase in the chemical mixture is (95% CI:) times as likely to obtain cancer. The first 14 rows of `boot.est` give us summary statistics about the weight estimates. Using the criterion that the pooled mean of the weight estimate greater than 1/14 is important, the following chemicals have the largest contributions to the overall mixture.

```
> chemicals <- boot.est[1:14, ]
> row.names(chemicals)[chemicals$pooled.mean >= 1 / 14]
```

```
[1] "ddt"              "pentachlorophenol" "pcb_105"
[4] "pcb_138"
```

We can also obtain an overall sense of how WQS model fits the data from bootstrapping imputation. In a similar spirit in combining the WQS parameter estimates, we combine the AIC from the two models. The `combine.AIC()` function takes the average and standard deviation of the individual AIC estimates from the separate WQS models. The only argument, `AIC`, takes a numeric vector of AIC's, which is saved in a `do.many.wqs()` object (eg. `boot.wqs$AIC`).

```
> boot.wqs$AIC
```

```
[1] 660.7468 665.1193
```

```
> boot.AIC <- combine.AIC(boot.wqs$AIC)
```

Compared to Examples 1 and 2, the bootstrapped MI-WQS model (AIC: 662.9 +- 3.1) fits the data similar to a WQS model using the BDLQ1 approach (AIC: 677.0) and worse than a WQS model using complete data (AIC: 660.7).

Example 4: Univariate Bayesian multiple imputation of BDL values

Instead of using bootstrapping imputation, the `impute.univariate.bayesian.mi()` imputes the BDL values using a Bayesian paradigm. The logs of the observed chemicals x_{ij} are assumed to independently follow normal distributions with mean μ_j and standard error σ_j. We place a Jeffrey's prior of the univariate normal on the parameters. In order to sample from the posterior predictive density of missing values ($X_{j,\text{miss}}$) given the observed values ($X_{j,\text{obs}}$), we run a Gibbs sampler of length T for each chemical. In step t of the sampler:

(Step 0): Given complete data $X = (X_{\text{miss}}^{(t-1)}, X_{\text{obs}})$, calculate the mean ϖ and variance S as:

$$\varpi = \frac{1}{n} \cdot \sum_{i=1}^{n} \log(x_i) \text{ and } S = \frac{1}{n-1} \cdot \sum_{i=1}^{n} (\log(x_i) - \varpi)^2 .$$

(Step 1): Simulate the posterior variance $\sigma^{2^{(t)}}$ given the mean and complete data from the inverse gamma distribution:

$$\sigma^2 | \mu^{(t-1)}, \log(X_{obs}), \log\left(X_{miss}^{(t-1)}\right) \sim IG\left(\frac{n-1}{2}, \frac{n-1}{2} * S\right).$$

(Step 2): Simulate the posterior mean $\mu^{(t)}$ given the variance and complete data from the normal distribution:

$$\mu | \sigma^{2^{(t)}}, \log(X_{obs}), \log\left(X_{miss}^{(t-1)}\right) \sim N\left(\varpi, sd = \frac{\sigma^{(t)}}{\sqrt{n}}\right).$$

(Step 3): Using current parameter estimates, impute $\log(X^{(t)}_{miss,i})$ from the normal distribution truncated between 0 and DL_j, or:

$$\log\left(X_{miss,i}\right)|\mu^{(t)},\sigma^{2^{(t)}} \sim TruncNorm\left(\mu^{(t)},\sigma^{2^{(t)}},a=0,b=DL_j\right).$$

for $i = 1\cdots n_{0j}$, where n_{0j} is the total number of BDL values for the *jth* chemical. We assessed convergence using Gelman-Rubin's R statistics (Gelman and Rubin, 1992). To construct approximately independent sets of complete concentrations, we join the observed values with the imputed values taken every tenth state from the end of the missing value chain. This Gibbs Sampler is repeated for all chemicals.

The impute.univariate.bayesian.mi() function applies this Bayesian algorithm to our dataset. The X argument takes a matrix with incomplete data, like simdata87$X.bdl. The DL argument takes the detection limits of X, which must be a numeric vector, like simdata87$DL. Bayesian imputation currently does not use covariate information. The T argument specifies the length of the Gibbs sampler (like 6000), and the n.burn argument specifies the burn-in (like 400). The K argument gives the number of imputed datasets generated (like 2). The impute.univariate.bayesian.mi() function returns a list consisting of three categories: a series of checks, the imputed array, and the MCMC (Markov chain Monte Carlo) chains.

```
> set.seed(472195)
> result.imputed <- impute.univariate.bayesian.mi(
+   X = simdata87$X.bdl,
+   DL = simdata87$DL,
+   T = 6000,
+   n.burn = 400,
+   K = 2
+ )

#> Start MCMC Data Augmentation Algorithm...

#> Checking for convergence with 2nd chain ...

  gelman.stat     is.converge
 Min.   :0.9998   Mode:logical
 1st Qu.:1.0001   TRUE:1428
 Median :1.0004
 Mean   :1.0010
 3rd Qu.:1.0013
 Max.   :1.0182
#> Evidence suggests that all 1428 parameters have converged.
#> Draw 2 Multiple Imputed Set(s) from states
[1] 6000 5990
#> Check: Indicator of # of missing values above detection limit
[1] 0
```

The impute.univariate.bayesian.mi() function returns a check of convergence in convg.table and a check of correct imputation in indicator.miss. To check for convergence, a summary of a data frame convg.table is shown above. The first column consists of the Gelman-Rubin statistics of the MCMC variables. (In the dataset simdata87, there are $(100+2)*14 = 1428$ MCMC variables, as each chemical has 102 MCMC variables: 100 missing values, mean, and variance.) The is.converge column of convg.table is a logical vector that specifies whether each MCMC variable has converged. This occurs if its Gelman-Rubin statistic is less than 1.1. In our example, the chains give evidence of convergence. The "Indicator of # missing values above the detection limit" shown above, represented with indicator.miss, is included to check if the imputation scheme occurred correctly. It should be zero, which it is shown above. The indicator.miss sums a logical vector of length c, in which an entry is TRUE if the imputed values are above the detection limit.

The element X.imputed of result.imputed list saves the imputed chemical values as an array, where the first dimension is the number of subjects (n), the second is the number of chemicals (c), and the third is the number of imputed datasets generated (K). Sample minima, means, and maxima (calculated by function f()) between two imputed datasets indicate that datasets are different; so when MI is applied, the parameter estimates should be different. Note that low values from Bayesian imputation differ from low bootstrap values as in Example 3.

```
> apply(result.imputed$X.imputed, 2:3, f)
```

```
, , Imputed.1

     alpha-chlordane     dieldrin gamma-chlordane   lindane methoxychlor
min     2.359430e-03 3.086685e-01        1.691056  1.215282     1.540713
P.5     5.242831e-01 3.183691e+00        3.610778  2.600923     4.182447
mean    4.409886e+04 4.332269e+02       47.283406 16.800662    80.420552
max     1.735472e+07 4.867330e+04      139.336894 75.360498   316.071414
              dde        ddt pentachlorophenol      pcb_105      pcb_118
min  2.768024e-02   0.482848         0.7046569   0.09017045   0.09142874
P.5  1.543214e+00   2.237090         2.7680096   1.16068653   1.30752584
mean 2.546653e+03  16.013644        20.1570964  16.48534459   9.68569878
max  3.835674e+05 178.853960       285.2293482 400.36011141 142.26195601
          pcb_138      pcb_153     pcb_170      pcb_180
min    0.01173117   0.02468827   0.7360772 2.164636e-03
P.5    0.68685631   0.38681360   2.0959549 5.996663e-01
mean  12.61938196  13.81475010  11.5772453 8.412071e+00
max  269.09903138 383.79850341 115.2060922 2.038594e+02

, , Imputed.2

     alpha-chlordane     dieldrin gamma-chlordane   lindane methoxychlor
min     8.514882e-03 4.098326e-01        1.462564  1.089740     1.425771
P.5     4.664125e-01 3.332290e+00        3.418596  2.638882     4.321919
mean    4.409886e+04 4.332296e+02       47.258091 16.797956    80.424921
max     1.735472e+07 4.867330e+04      139.336894 75.360498   316.071414
              dde         ddt pentachlorophenol      pcb_105      pcb_118
min  5.568180e-02   0.9352681          0.297144 3.085055e-03 5.716621e-03
P.5  1.580193e+00   2.5371056          2.670481 1.110512e+00 1.215440e+00
mean 2.546663e+03  16.0346907         20.148387 1.648278e+01 9.684614e+00
max  3.835674e+05 178.8539599        285.229348 4.003601e+02 1.422620e+02
          pcb_138      pcb_153     pcb_170      pcb_180
min  9.471776e-03 3.198244e-03   0.5399104   0.00652091
P.5  7.249506e-01 4.114339e-01   2.1037311   0.58501721
mean 1.262209e+01 1.381682e+01  11.5786930   8.41077395
max  2.690990e+02 3.837985e+02 115.2060922 203.85939377
```

The `impute.univariate.bayesian.mi()` function also returns the three entire MCMC chains: the means of components, the standard errors, and the imputed missing values. The **coda** package, which "provides functions for summarizing and plotting the output from ... MCMC simulations", saved these MCMC chains as `MCMC` objects (Plummer et al., 2006).

Using the imputed datasets saved in array `result.imputed$X.imputed`, the `do.many.wqs()` function implements WQS regression on both datasets with a binary outcome, as in Example 3. The setup is the same as before, but we are using Bayesian imputed datasets, as in `result.imputed$X.imputed`. Similar to Example 3, the element, `wqs.imputed.estimates`, in the resulting `bayes.wqs` list contains the WQS parameter estimates for each imputed dataset.

```
> set.seed(50679)
> bayes.wqs <- do.many.wqs(
+   y = simdata87$y.scenario, X.imputed = result.imputed$X.imputed,
+   Z = simdata87$Z.sim,
+   proportion.train = 0.5,
+   n.quantiles = 4,
+   B = 100,
+   b1.pos = TRUE,
+   signal.fn = "signal.converge.only",
+   family = "binomial"
+ )
> wqs.imputed.estimates <- bayes.wqs$wqs.imputed.estimates

#> Sample size: 1000; Number of chemicals: 14;
Number of completed datasets: 2; Number of covariates modeled:  4
```

Lastly, we can combine the multiple WQS estimates using the `pool.mi()` function, exactly as in Example 3. The output, given in `bayesian.est`, returns the statistics of the combined estimates for each WQS parameter and answers the research questions of interest (Table 2).

```
> bayesian.est <- pool.mi(
+   to.pool = bayes.wqs$wqs.imputed.estimates,
+   n = nrow(simdata87$X.bdl),
+   prt = TRUE
+ )

#> Pooling estimates from 2 imputed analyses for 20 parameters.
                    pooled.mean pooled.total.se frac.miss.info   CI.1   CI.2
alpha.chlordane           0.004           0.024          0.005 -0.044  0.051
dieldrin                  0.020           0.066          0.327 -0.119  0.160
gamma.chlordane           0.011           0.036          0.019 -0.060  0.083
lindane                   0.011           0.041          0.181 -0.072  0.094
methoxychlor              0.007           0.041          0.019 -0.073  0.087
dde                       0.055           0.115          0.125 -0.174  0.284
ddt                       0.269           0.247          0.836 -0.850  1.388
pentachlorophenol         0.147           0.250          0.859 -1.099  1.393
pcb_105                   0.198           0.143          0.359 -0.109  0.506
pcb_118                   0.016           0.055          0.037 -0.091  0.123
pcb_138                   0.157           0.191          0.337 -0.250  0.564
pcb_153                   0.024           0.081          0.057 -0.135  0.183
pcb_170                   0.064           0.137          0.454 -0.249  0.378
pcb_180                   0.016           0.051          0.273 -0.089  0.120
(Intercept)              -1.828           0.447          0.314 -2.770 -0.885
Age                      -0.007           0.094          0.795 -0.373  0.358
Female                   -0.127           0.230          0.392 -0.631  0.378
Hispanic                  0.517           0.228          0.276  0.044  0.989
Non.Hispanic_Others       0.134           0.282          0.509 -0.538  0.807
WQS                       1.030           0.504          0.919 -2.441  4.501
                    P.value
alpha.chlordane       0.883
dieldrin              0.762
gamma.chlordane       0.761
lindane               0.791
methoxychlor          0.859
dde                   0.632
ddt                   0.395
pentachlorophenol     0.624
pcb_105               0.187
pcb_118               0.770
pcb_138               0.424
pcb_153               0.766
pcb_170               0.652
pcb_180               0.759
(Intercept)          <0.001
Age                   0.943
Female                0.593
Hispanic              0.034
Non.Hispanic_Others   0.650
WQS                   0.233
```

Looking at the WQS estimate in `bayesian.est`, the odds ratio of the overall chemical mixture on cancer is 2.8 with a 95% confidence interval between 0.09 and 90.15. The following chemicals, in which their weight estimates are greater than 1/14, are considered an important and may be associated with increased cancer risk.

```
> chemicals <- bayesian.est[1:14, ]
> row.names(chemicals)[chemicals$pooled.mean >= 1 / 14]

[1] "ddt"               "pentachlorophenol" "pcb_105"
[4] "pcb_138"
```

To get an overall sense of how the Bayesian-imputed WQS models fit the data, the `combine.AIC()` function combines the AIC calculated from Bayesian MI-WQS models (`bayes.wqs$AIC`).

```
> bayes.wqs$AIC
```

```
[1] 660.7468 665.1193

> miWQS::combine.AIC(bayes.wqs$AIC)

[1] "662.9 +- 3.1"
```

The Bayesian MI-WQS model (AIC: 662.9 +- 3.1) has the same fit as the bootstrapped MI-WQS (Example 3, AIC: 662.9 +- 3.1).

Recommendations in using miWQS package

We have integrated WQS regression into the MI framework in a flexible R package called **miWQS** to meet a wide variety of needs (Figure 6). The data used in this package consist of a mixture of correlated components that share a common outcome while adjusting for other covariates. The correlated components in the set, $\boldsymbol{X}$, may be complete or interval-censored between zero and low thresholds, or detection limits, that may be different across the components. The common outcome, y, may be modeled as binary, continuous, count-based, or rate-based and can be adjusted by the `family` and `offset` arguments of `estimate.wqs()`.

Additional covariates, $\boldsymbol{Z}$, may be used in the bootstrap imputation and WQS models. However, the univariate Bayesian model does not include covariate information in imputing the BDL values. This makes any covariate confounders uncorrelated with the imputed concentrations BDL. Thereby, the WQS regression coefficients, such as the weights and overall mixture effect, may be biased towards zero (Forer, 2014; Little, 1992).

Another limitation of the univariate Bayesian and bootstrap imputation models is that the X's are imputed independently while the actual X's are correlated. This makes the correlations among the imputed BDL values of different components biased towards zero. One concern is that the mixture with independently imputed BDL values may introduce some bias in the health effect estimate if a large amount of BDL values is present. As an alternative, an imputation model could take advantage of the correlations to impute a potentially more precise estimate (Dong and Peng, 2013; Little, 1992). One such approach is the multivariate Bayesian regression imputation model, which we are evaluating in ongoing work (Hargarten and Wheeler, 2020).

If $\boldsymbol{X}$ is interval-censored, the choice of the imputation technique depends on the majority vote of BDL values among the components (Hargarten and Wheeler, 2020) (Figure 6). Previous literature suggests ignoring any chemicals that have greater than 80% of its values BDL (Helsel, 2012, pg. 93) (Bolks et al., 2014, pg. 14). When most chemicals have 80% of its values BDL, we suggest using the BDLQ1 approach (Hargarten and Wheeler, 2020). When most chemicals have less than 80% of its values BDL, the user should perform Bayesian or bootstrapping multiple imputation (Hargarten and Wheeler, 2020). The **miWQS** package, though, still allows the user to perform single imputation. Regardless of the technique used, researchers may use the **miWQS** package in order to detect an association between the mixture and the outcome and to identify the important components in that mixture.

Conclusion

Although environmental exposures data motivated us to develop the **miWQS** package, the package may be applied to other areas in public health and medicine. Wheeler et al. (Wheeler et al., 2019a) recently used WQS regression to estimate the effect of a SES index on childhood blood lead risk and to find which socioeconomic variables are important. The correlated SES variables considered were of these types: educational achievement, race, income, health, housing, and employment. The five most important variables found were: percent of homes built before 1940, percent of not using Social-Security income, percent of renter-occupied housing, percent unemployed, and percent of the African American population (Wheeler et al., 2019a, pg.974). Other similar studies may be analyzed using the **miWQS** package. To our knowledge, WQS has not yet been applied in analyzing a high-throughput gene expression dataset. For instance, a GWAS is conducted to find genetic risks for complex disease and to identify specific genes. Given that SNPs are correlated with each other (Ferber and Archer, 2015) and a binary or continuous health outcome, the **miWQS** package may be used to conduct a WQS regression to address these research aims. In the years to come, researchers may add other imputation models to our established computational structure in order to find components that impact human health.

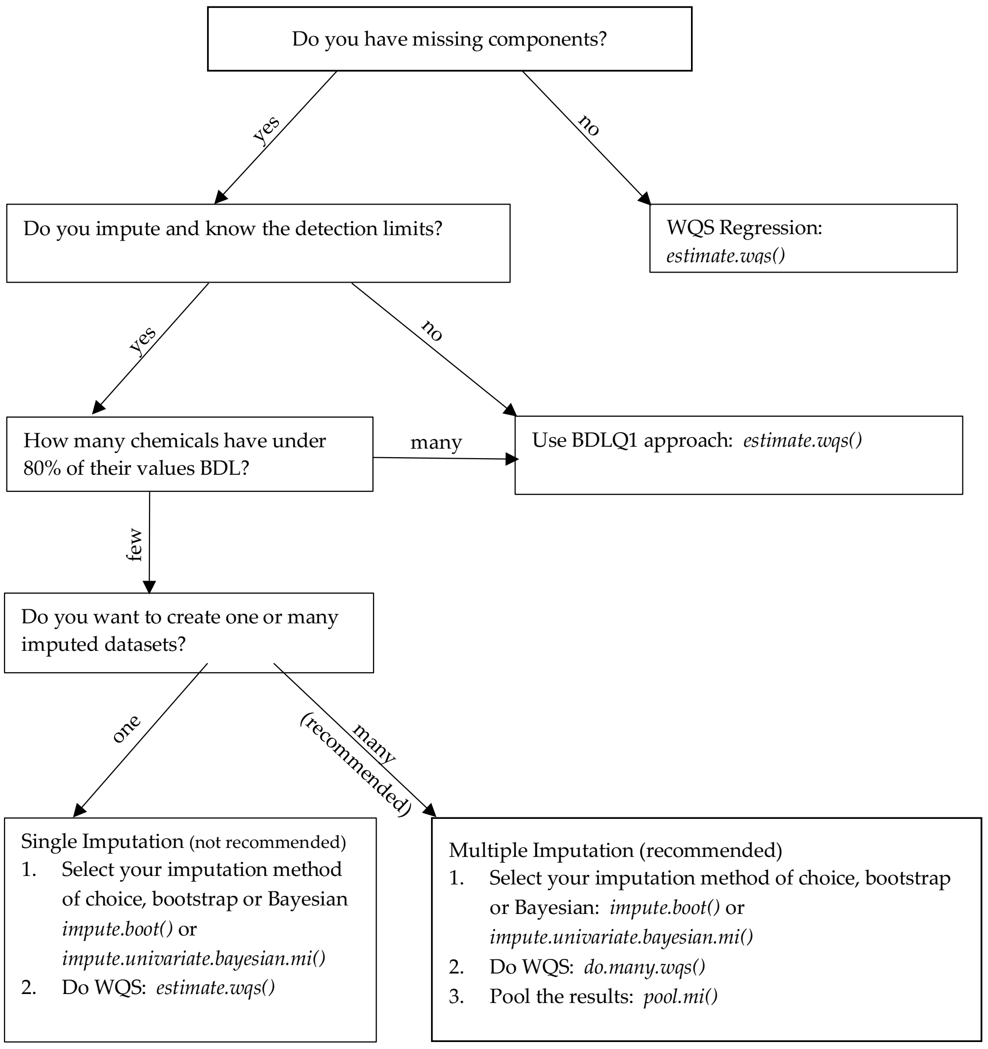

Do you have missing components?
yes
no
Do you impute and know the detection limits?
WQS Regression: estimate.wqs()
yes
no
How many chemicals have under 80% of their values BDL?
many
Use BDLQ1 approach: estimate.wqs()
few
Do you want to create one or many imputed datasets?
one
many (recommended)
Single Imputation (not recommended)
1. Select your imputation method of choice, bootstrap or Bayesian impute.boot() or impute.univariate.bayesian.mi()
2. Do WQS: estimate.wqs()
Multiple Imputation (recommended)
1. Select your imputation method of choice, bootstrap or Bayesian: impute.boot() or impute.univariate.bayesian.mi()
2. Do WQS: do.many.wqs()
3. Pool the results: pool.mi()

Figure 6: A decision tree to help researchers in using the miWQS package. The package is flexible and can meet a wide range of needs.

Computational details

The functions in **miWQS** package relied upon code developed in other packages on CRAN. The steps in the `estimate.wqs()` function also relied upon other packages: the `solnp()` function in **Rsolnp** package (Ghalanos and Theussl, 2015), the `glm2()` function in **glm2** package (Marschner and Donoghoe, 2011, pg. 2), the `list.merge()` function in **rlist** package (Ren, 2016), the `format.pval()` in **Hmisc** package (Harrell, 2020), the `gather()` function from **tidyr** package (Wickham and Henry, 2020), and the **ggplot2** package (Wickham, 2016). The `impute.Lubin()` function used the **survival** package (Therneau and Lumley, 2015). The `impute.univariate.bayesian.mi()` function used: the `rinvgamma()` function in the **invgamma** package (Kahle and Stamey, 2017), the `rtruncnorm()` function in **truncnorm** package (Mersmann et al., 2020), the `possibly()` function in the **purrr** package (Henry and Wickham, 2020) and the **coda** package (Plummer et al., 2006). Additionally, the `ggcorr()` function in the **GGally** produced the heat map in Figure 2 (Schloerke et al., 2020).

This vignette is successfully processed using the following.

```
-- Session info ---------------------------------------------------

setting  value
version  R version 4.0.2 (2020-06-22)
os       macOS  10.16
system   x86_64, darwin17.0
ui       X11
```

```
 language (EN)
 collate  en_US.UTF-8
 ctype    en_US.UTF-8
 tz       America/New_York
 date     2021-01-20

--  Packages -------------------------------------------------------

 package   * version date       lib source
 coda        0.19-4  2020-09-30 [1] CRAN (R 4.0.2)
 GGally    * 2.0.0   2020-06-06 [1] CRAN (R 4.0.2)
 ggplot2   * 3.3.3   2020-12-30 [1] CRAN (R 4.0.2)
 glm2        1.2.1   2018-08-11 [1] CRAN (R 4.0.2)
 gWQS        3.0.0   2020-06-23 [1] CRAN (R 4.0.2)
 Hmisc       4.4-2   2020-11-29 [1] CRAN (R 4.0.2)
 invgamma    1.1     2017-05-07 [1] CRAN (R 4.0.2)
 knitr     * 1.30    2020-09-22 [1] CRAN (R 4.0.2)
 mi          1.0     2015-04-16 [1] CRAN (R 4.0.2)
 mice        3.10.0  2020-07-13 [1] CRAN (R 4.0.2)
 miWQS     * 0.4.0   2020-07-27 [1] local
 norm        1.0-9.5 2013-02-28 [1] CRAN (R 4.0.2)
 purrr       0.3.4   2020-04-17 [1] CRAN (R 4.0.2)
 rlist       0.4.6.1 2016-04-04 [1] CRAN (R 4.0.2)
 rmarkdown   2.3     2020-06-18 [1] CRAN (R 4.0.2)
 Rsolnp      1.16    2015-12-28 [1] CRAN (R 4.0.2)
 rticles     0.16.1  2020-09-22 [1] Github (rstudio/rticles@b0bbbc0)
 survival    3.1-12  2020-04-10 [1] CRAN (
 tidyr       1.1.2   2020-08-27 [1] CRAN (R 4.0.2)
 tinytex     0.26    2020-09-22 [1] CRAN (R 4.0.2)
 truncnorm   1.0-8   2020-07-27 [1] Github (olafmersmann/truncnorm@eea186e)
 wqs         0.0.1   2015-10-05 [1] CRAN (R 4.0.2)
 yaml        2.2.1   2020-02-01 [1] CRAN (R 4.0.2)

[1] /Library/Frameworks/R.framework/Versions/4.0/Resources/library
```

Acknowledgments

We like to thank Keith W. Zirkle and Anny-Claude Joseph for their editorial comments on this vignette. Additionally, we thank the anonymous reviewers of *The R Journal* who have improved this vignette. Lastly, we appreciate Yihui Xie's work in creating the **rticles** package that enabled us to write this vignette from the Rmarkdown environment (Xie et al., 2020).

Abbreviations

- AIC: Akaike information criterion
- BDL: below the detection limit
- BDLQ1: placing the BDL values into the first quantile
- BMI: body mass index
- CRAN: the comprehensive R archive network
- DL: detection limit
- GWAS: genomic wide association study
- MCMC: Markov chain Monte Carlo
- MI: multiple imputation
- MI-WQS: multiple Imputation in connection with the weighted quantile sum regression
- SES: socioeconomic status
- SNPs: single nucleotide polymorphisms
- WQS: weighted quantile sum

Notation: + n sample size + c number of chemicals + K number of imputations

Appendix

Deciding whether the overall mixture effect is positively or negatively related to the outcome in WQS regression

A researcher must decide whether the overall mixture effect, β_1, is positively or negatively related to the outcome in WQS regression. One way is to perform a series of individual chemical regressions and look at the sign of the regression coefficients. This is performed via the `analyze.individually()` function. In each regression, the outcome y is regressed on the log of each chemical $\boldsymbol{X}$ and any covariates $\boldsymbol{Z}$ using the **glm2** package (Marschner and Donoghoe, 2011). Any missing values are ignored. The arguments in `analyze.individually()` are the same as the arguments specified in `estimate.wqs()`. In `simdata87`, our outcome is element `y.scenario`, the chemical mixture is `X.true`, the covariates are contained in `Z.sim`. As the outcome in `simdata87` is binary, we assign `"binomial"` to the `family` argument. The `analyze.individually()` function returns a data frame that consists of: the name of the chemical, the individual chemical effect estimate and its standard error, and an assessment of the WQS model fit using the Akaike Information Criterion (AIC).

```
> analyze.individually(
+   y = simdata87$y.scenario, X = simdata87$X.true,
+   Z = simdata87$Z.sim, family = "binomial"
+ )

       Chemical.Name Estimate Std.Error      AIC
1    alpha-chlordane    0.128     0.018 1315.527
2           dieldrin    0.176     0.033 1339.606
3    gamma-chlordane    1.310     0.192 1319.118
4            lindane    0.817     0.139 1332.276
5       methoxychlor    1.056     0.150 1315.461
6                dde    0.169     0.025 1319.293
7                ddt    0.176     0.086 1365.064
8  pentachlorophenol    0.245     0.081 1360.018
9            pcb_105   -0.026     0.051 1368.984
10           pcb_118    0.332     0.072 1347.162
11           pcb_138    0.356     0.056 1325.112
12           pcb_153    0.308     0.046 1321.903
13           pcb_170    0.404     0.087 1346.696
14           pcb_180    0.311     0.059 1339.602
```

The sign of the estimates indicates whether the overall mixture effect should be positive or negative. As most of the estimates are positive here, we will assume that the overall mixture is positively related to the outcome. Then, we can set the `b1.pos` argument in `estimate.wqs()` to be `TRUE`. In terms of model fit, the complete-data mixture WQS model in Example 1 with an AIC of 660 fits the data better than any individual chemical model (see the AIC's above).

Bibliography

J. Barnard, N. Schenker, and D. Rubin. Multiple Imputation. *International Encyclopedia of the Social & Behavioral Sciences*, pages 88–93, Mar. 2015. URL `https://doi.org/10.1016/B978-0-08-097086-8.42083-0`. [p236]

A. Bolks, A. DeWire, and J. Harcum. Baseline Assessment of Left-Censored Environmental Data Using R. Technical Report 1, Tetra Tech, Inc., Fairfax, VA, June 2014. URL `https://www.epa.gov/nps/nonpoint-source-monitoring-technotes`. [p244]

J. M. Braun, C. Gennings, R. Hauser, and T. F. Webster. What Can Epidemiological Studies Tell Us About the Impact of Chemical Mixtures on Human Health? *Environmental Health Perspectives*, 124 (1):A6–A9, Jan. 2016. ISSN 0091-6765. URL `https://doi.org/10.1289/ehp.1510569`. [p226]

W. S. Bush and J. H. Moore. Chapter 11: Genome-Wide Association Studies. *PLOS Computational Biology*, 8(12):e1002822, Dec. 2012. ISSN 1553-7358. doi: 10.1371/journal.pcbi.1002822. [p226]

C. Carrico, C. Gennings, D. C. Wheeler, and P. Factor-Litvak. Characterization of Weighted Quantile Sum Regression for Highly Correlated Data in a Risk Analysis Setting. *Journal of Agricultural, Biological, and Environmental Statistics*, 20(1):100–120, Dec. 2014. ISSN 1085-7117, 1537-2693. doi: 10.1007/s13253-014-0180-3. [p226, 227, 229, 230]

J. Czarnota and D. Wheeler. {wqs}: Weighted Quantile Sum Regression, Oct. 2015. URL https://cran.r-project.org/web/packages/wqs/index.html. [p226]

J. Czarnota, C. Gennings, J. S. Colt, A. J. De Roos, J. R. Cerhan, R. K. Severson, P. Hartge, M. H. Ward, and D. C. Wheeler. Analysis of Environmental Chemical Mixtures and Non-Hodgkin Lymphoma Risk in the NCI-SEER NHL Study. *Environmental Health Perspectives*, 123(10):965–970, Oct. 2015a. ISSN 1552-9924. URL https://doi.org/10.1289/ehp.1408630. [p226]

J. Czarnota, C. Gennings, and D. C. Wheeler. Assessment of Weighted Quantile Sum Regression for Modeling Chemical Mixtures and Cancer Risk. *Cancer Informatics*, 14:159–171, May 2015b. ISSN 1176-9351. URL https://doi.org/10.4137/CIN.S17295. [p226, 227, 229, 230]

Y. Dong and C.-Y. J. Peng. Principled Missing Data Methods for Researchers. *SpringerPlus*, 2(1), 2013. ISSN 2193-1801. doi: 10.1186/2193-1801-2-222. [p226, 239, 244]

K. Ferber and K. J. Archer. Modeling Discrete Survival Time Using Genomic Feature Data. *Cancer Informatics*, 14s2:37–43, Mar. 2015. ISSN 1176-9351. URL https://doi.org/10.4137/CIN.S17275. [p226, 244]

B. Forer. Missing Data. In A. C. Michalos, editor, *Encyclopedia of Quality of Life and Well-Being Research*, pages 4078–4082. Springer Netherlands, Dordrecht, 2014. ISBN 978-94-007-0752-8. doi: 10.1007/978-94-007-0753-5_1821. [p236, 244]

A. Gelman and D. B. Rubin. Inference from Iterative Simulation Using Multiple Sequences. *Statistical Science*, 7(4):457–472, Nov. 1992. ISSN 0883-4237. URL https://doi.org/10.1214/ss/1177011136. [p241]

C. Gennings, C. Carrico, P. Factor-Litvak, N. Krigbaum, P. M. Cirillo, and B. A. Cohn. A Cohort Study Evaluation of Maternal PCB Exposure Related to Time to Pregnancy in Daughters. *Environmental Health*, 12(1):66, Aug. 2013. ISSN 1476-069X. URL https://doi.org/10.1186/1476-069X-12-66. [p226, 227, 229, 230]

A. Ghalanos and S. Theussl. Rsolnp: General Non-Linear Optimization Using Augmented Lagrange Multiplier Method, July 2015. URL https://cran.r-project.org/web/packages/Rsolnp/index.html. [p245]

P. M. Hargarten and D. C. Wheeler. Accounting for the Uncertainty Due to Chemicals Below the Detection Limit in Mixture Analysis. *Environmental Research*, 186:109466, July 2020. ISSN 00139351. URL https://doi.org/10.1016/j.envres.2020.109466. [p226, 227, 228, 230, 234, 244]

P. M. Hargarten and D. C. Wheeler. miWQS: Multiple Imputation Using Weighted Quantile Sum Regression, 2021. URL https://CRAN.R-project.org/package=miWQS. [p226]

F. E. Harrell, Jr. Hmisc: Harrell Miscellaneous, 2020. URL https://cran.r-project.org/web/packages/Hmisc/index.html. [p245]

D. R. Helsel. *Statistics for Censored Environmental Data Using Minitab and R*. John Wiley & Sons, Hoboken, NJ, USA, second edition, 2012. ISBN 978-0-470-47988-9. [p244]

L. Henry and H. Wickham. Purrr: Functional programming tools, 2020. URL https://CRAN.R-project.org/package=purrr. [p245]

M. K. Horton, B. C. Blount, L. Valentin-Blasini, R. Wapner, R. Whyatt, C. Gennings, and P. Factor-Litvak. CO-Occurring Exposure to Perchlorate, Nitrate and Thiocyanate Alters Thyroid Function in Healthy Pregnant Women. *Environmental Research*, 143(Pt A):1–9, Nov. 2015. ISSN 1096-0953. URL https://doi.org/10.1016/j.envres.2015.09.013. [p226]

D. Kahle and J. Stamey. Invgamma: The Inverse Gamma Distribution, May 2017. URL https://CRAN.R-project.org/package=invgamma. [p245]

R. J. A. Little. Regression With Missing X's: A Review. *Journal of the American Statistical Association*, 87 (420):1227–1237, 1992. ISSN 0162-1459. doi: 10.2307/2290664. [p244]

J. H. Lubin, J. S. Colt, D. Camann, S. Davis, J. R. Cerhan, R. K. Severson, L. Bernstein, and P. Hartge. Epidemiologic Evaluation of Measurement Data in the Presence of Detection Limits. *Environmental Health Perspectives*, 112(17):1691–1696, Dec. 2004. ISSN 0091-6765. URL https://doi.org/10.1289/ehp.7199. [p235, 236]

I. C. Marschner and M. W. Donoghoe. Glm2: Fitting Generalized Linear Models with Convergence Problems. *The R Journal*, 3(2):12–15, Dec. 2011. ISSN 2073-4859. [p245, 247]

O. Mersmann, H. Trautmann, D. Steuer, and B. Bornkamp. Truncnorm: Truncated Normal Distribution, 2020. URL https://github.com/olafmersmann/truncnorm. [p245]

C. Metayer, J. S. Colt, P. A. Buffler, H. D. Reed, S. Selvin, V. Crouse, and M. H. Ward. Exposure to Herbicides in House Dust and Risk of Childhood Acute Lymphoblastic Leukemia. *Journal of Exposure Science & Environmental Epidemiology*, 23(4):363–370, July 2013. ISSN 1559-0631, 1559-064X. URL https://doi.org/10.1038/jes.2012.115. [p234]

A. A. Novo and J. L. Schafer. Norm: Analysis of Multivariate Normal Datasets with Missing Values, 2013. URL https://CRAN.R-project.org/package=norm. [p227]

Q. Pan and R. Wei. Fraction of Missing Information at Different Missing Data Fractions in the 2012 NAMCS Physician Workflow Mail Survey. *Applied Mathematics*, 07(10):1057–1067, 2016. ISSN 2152-7385, 2152-7393. URL https://doi.org/10.4236/am.2016.710093. [p240]

M. Plummer, N. Best, K. Cowles, and K. Vines. Coda: Convergence Diagnosis and Output Analysis for MCMC. *R News*, 6(1):7–11, Mar. 2006. [p242, 245]

K. Ren. Rlist: A Toolbox for Non-Tabular Data Manipulation, 2016. URL https://CRAN.R-project.org/package=rlist. [p245]

S. Renzetti, P. Curtin, A. C. Just, G. Bello, and C. Gennings. gWQS: Generalized Weighted Quantile Sum Regression, 2020. URL https://CRAN.R-project.org/package=gWQS. [p226]

D. B. Rubin. *Multiple Imputation for Nonresponse in Surveys*. John Wiley & Sons, New York, NY, USA, 1987. ISBN 978-0-471-08705-2. [p226, 239]

V. Savalei and M. Rhemtulla. On Obtaining Estimates of the Fraction of Missing Information from Full Information Maximum Likelihood. *Structural Equation Modeling: A Multidisciplinary Journal*, 19(3): 477–494, July 2012. ISSN 1070-5511, 1532-8007. doi: 10.1080/10705511.2012.687669. [p240]

B. Schloerke, J. Crowley, D. Cook, F. Briatte, M. Marbach, E. Thoen, A. Elberg, and J. Larmarange. GGally: Extension to 'ggplot2', 2020. URL https://CRAN.R-project.org/package=GGally. [p245]

Y.-S. Su, A. Gelman, J. Hill, and M. Yajima. Multiple Imputation with Diagnostics (mi) in R: Opening Windows into the Black Box. *Journal of Statistical Software*, 45(2):1–31, Dec. 2011. ISSN 1548-7660. doi: 10.18637/jss.v045.i02. [p227]

T. M. Therneau and T. Lumley. A Package for Survival Analysis in S, 2015. URL https://CRAN.R-project.org/package=survival. [p245]

S. van Buuren and K. Groothuis-Oudshoorn. Mice: Multivariate Imputation by Chained Equations in R. *Journal of Statistical Software*, 45(3):1–67, 2011. ISSN 1548-7660. URL https://doi.org/10.18637/jss.v045.i03. [p227]

M. H. Ward, J. S. Colt, C. Metayer, R. B. Gunier, J. Lubin, V. Crouse, M. G. Nishioka, P. Reynolds, and P. A. Buffler. Residential Exposure to Polychlorinated Biphenyls and Organochlorine Pesticides and Risk of Childhood Leukemia. *Environmental Health Perspectives*, 117(6):1007–1013, June 2009. ISSN 0091-6765. URL https://doi.org/10.1289/ehp.0900583. [p234]

M. H. Ward, J. S. Colt, N. C. Deziel, T. P. Whitehead, P. Reynolds, R. B. Gunier, M. Nishioka, G. V. Dahl, S. M. Rappaport, P. A. Buffler, and C. Metayer. Residential Levels of Polybrominated Diphenyl Ethers and Risk of Childhood Acute Lymphoblastic Leukemia in California. *Environmental Health Perspectives*, 122(10):1110–1116, June 2014. ISSN 0091-6765. URL https://doi.org/10.1289/ehp.1307602. [p234]

D. C. Wheeler, J. Czarnota, and R. M. Jones. Estimating an Area-Level Socioeconomic Status Index and Its Association with Colonoscopy Screening Adherence. *PLOS One*, 12(6):e0179272, 2017. ISSN 1932-6203. doi: 10.1371/journal.pone.0179272. [p226]

D. C. Wheeler, R. M. Jones, M. Schootman, and E. J. Nelson. Explaining Variation in Elevated Blood Lead Levels Among Children in Minnesota Using Neighborhood Socioeconomic Variables. *Science of the Total Environment*, 650:970–977, Feb. 2019a. ISSN 00489697. doi: 10.1016/j.scitotenv.2018.09.088. [p226, 244]

D. C. Wheeler, S. Raman, R. M. Jones, M. Schootman, and E. J. Nelson. Bayesian Deprivation Index Models for Explaining Variation in Elevated Blood Lead Levels among Children in Maryland. *Spatial and Spatio-temporal Epidemiology*, 30:100286, Aug. 2019b. ISSN 18775845. doi: 10.1016/j.sste.2019.100286. [p226]

D. C. Wheeler, E. K. Do, R. B. Hayes, K. Fugate-Laus, Westley L. Fallavollita, C. Hughes, and Bernard F. Fuemmeler. Neighborhood Disadvantage and Tobacco Retail Outlet and Vape Shop Outlet Rates. *International Journal of Environmental Research and Public Health*, 17(8):2864, Apr. 2020. ISSN 1660-4601. URL https://doi.org/10.3390/ijerph17082864. [p226]

I. R. White, P. Royston, and A. M. Wood. Multiple Imputation Using Chained Equations: Issues and Guidance for Practice. *Statistics in Medicine*, 30(4):377–399, Feb. 2011. ISSN 02776715. doi: 10.1002/sim.4067. [p226, 239]

H. Wickham. *Ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag, New York, NY, USA, second edition, 2016. ISBN 978-3-319-24277-4. URL http://ggplot2.org. [p231, 245]

H. Wickham and L. Henry. Tidyr: Easily Tidy Data with 'spread()' and 'gather()' Functions, 2020. URL https://CRAN.R-project.org/package=tidyr. [p245]

Y. Xie, J. Allaire, R Foundation, H. Wickham, Journal of Statistical Software, R. Vaidyanathan, Association for Computing Machinery, C. Boettiger, Elsevier, K. Broman, K. Mueller, B. Quast, R. Pruim, B. Marwick, C. Wickham, O. Keyes, M. Yu, D. Emaasit, T. Onkelinx, A. Gasparini, M.-A. Desautels, D. Leutnant, MDPI, Taylor and Francis, O. Öğreden, D. Hance, D. Nüst, P. Uvesten, E. Campitelli, J. Muschelli, A. Hayes, Z. N. Kamvar, N. Ross, R. Cannoodt, D. Luguern, D. M. Kaplan, S. Kreutzer, S. Wang, J. Hesselberth, and C. Dervieux. Rticles: Article formats for r markdown, 2020. URL https://github.com/rstudio/rticles. [p246]

Paul M. Hargarten
Virginia Commonwealth University
One Capitol Square
830 East Main Street Seventh Floor
Richmond, Virginia 23219

hargartenp@vcu.edu

David C. Wheeler
Virginia Commonwealth University
One Capitol Square
830 East Main Street Seventh Floor
Richmond, Virginia 23219

david.wheeler@vcuhealth.org

User-Specified General-to-Specific and Indicator Saturation Methods

by Genaro Sucarrat

Abstract General-to-Specific (GETS) modelling provides a comprehensive, systematic and cumulative approach to modelling that is ideally suited for conditional forecasting and counterfactual analysis, whereas Indicator Saturation (ISAT) is a powerful and flexible approach to the detection and estimation of structural breaks (e.g. changes in parameters), and to the detection of outliers. To these ends, multi-path backwards elimination, single and multiple hypothesis tests on the coefficients, diagnostics tests and goodness-of-fit measures are combined to produce a parsimonious final model. In many situations a specific model or estimator is needed, a specific set of diagnostics tests may be required, or a specific fit criterion is preferred. In these situations, if the combination of estimator/model, diagnostics tests and fit criterion is not offered in a pre-programmed way by publicly available software, then the implementation of user-specified GETS and ISAT methods puts a large programming-burden on the user. Generic functions and procedures that facilitate the implementation of user-specified GETS and ISAT methods for specific problems can therefore be of great benefit. The R package **gets** is the first software – both inside and outside the R universe – to provide a complete set of facilities for user-specified GETS and ISAT methods: User-specified model/estimator, user-specified diagnostics and user-specified goodness-of-fit criteria. The aim of this article is to illustrate how user-specified GETS and ISAT methods can be implemented with the R package **gets**.

Introduction

General-to-Specific (GETS) modelling provides a comprehensive, systematic and cumulative approach to modelling that is ideally suited for scenario analysis, e.g. conditional forecasting and counterfactual analysis. To this end, well-known ingredients (tests of coefficients, multi-path backwards elimination, diagnostics tests and fit criteria) are combined to produce a parsimonious final model that passes the chosen diagnostics. GETS modelling originated at the London School of Economics (LSE) during the 1960s, and gained widespread acceptance and usage in economics during the 1980s and 1990s. The two-volume article collection by Campos et al. (2005) provides a comprehensive historical overview of key-developments in GETS modelling. Software-wise, a milestone was reached in 1999, when the data-mining experiment of Lovell (1983) was re-visited by Hoover and Perez (1999). They showed that automated GETS modelling could improve substantially upon the then prevalent modelling approaches. The study spurred numerous new studies and developments, including Indicator Saturation (ISAT) methods, see Hendry et al. (2008) and Castle et al. (2015). ISAT methods provide a powerful and flexible approach to the detection and estimation of structural breaks (e.g. changes in parameters), and to the detection of outliers.

On CRAN, there are two packages that provide GETS methods. The second, named **gets**, is simply the successor of the first, which is named **AutoSEARCH**.[1] Since October 2014 the development of **AutoSEARCH** is frozen, and all development efforts have been directed towards **gets** together with Dr. Felix Pretis and Dr. James Reade.[2] An introduction to the **gets** package is provided by Pretis et al. (2018). However, it does does not cover the user-specification capabilities of the package, some of which were not available at the time.

At the time of writing (September 2020), the publicly available softwares that provide GETS and ISAT methods are contained in Table 1. Although they offer GETS and ISAT methods for some of the most popular models in applications, in many situations a specific model or estimator will be needed, a specific set of diagnostics tests may be required, or a specific fit criterion is preferred. In these situations, if the combination of estimator/model, diagnostics tests and fit criterion is not offered in a pre-programmed way by the publicly available softwares, then the implementation of user-specified GETS and ISAT methods puts a large programming-burden on the user. Generic functions and procedures that facilitate the implementation of user-specified GETS and ISAT methods for specific problems can therefore be of great benefit. The R package **gets**, since version 0.20 (September 2019), is the first software – both inside and outside the R universe – to provide a complete set of facilities for user-specified GETS and ISAT methods: User-specified model/estimator, user-specified diagnostics

[1]Both packages were created by me. Originally, I simply wanted to rename the first to the name of the second. This, however, is inconvenient in practice I was told, so I was instead asked by CRAN to publish a "new" package with the new name.

[2]Recently, Jonas Kurle and Moritz Schwarz have also made contributions. See the Gitub page for the current development version of the package: `https://github.com/gsucarrat/gets/`.

	HP1999 (MATLAB)	Autometrics (OxMetrics)	Grocer (Scilab)	genspec (STATA)	EViews*	**gets** (R)
More than 10 paths		Yes	Yes	Yes	Yes	Yes
GETS of linear regression	Yes	Yes	Yes	Yes	Yes	Yes
GETS of variance models						Yes
GETS of logit/count models		Yes				
GETS of probit models		Yes	Yes			
GETS of panel models			Yes	Yes		
GETS of MIDAS models					Yes	
ISAT of linear regression		Yes	Yes		Yes	Yes
User-specified GETS			Yes			Yes
User-specified ISAT						Yes
User-specified diagnostics			Yes			Yes
User-specified goodness-of-fit						Yes
Menu-based GUI		Yes			Yes	
Free and open source	Yes**		Yes	Yes**		Yes

Table 1: A comparison of publicly available GETS and ISAT softwares with emphasis on user-specification capabilities. HP1999, the MATLAB code of Hoover and Perez (1999). Autometrics, OxMetrics version 15, see Doornik and Hendry (2018). Grocer, version 1.8, see Dubois and Michaux (2019). genspec, version 1.2.2, see Clarke (2014). EViews, version 12, see IHS Markit (2020). **gets**, version 0.25, see Sucarrat et al. (2020), and Pretis et al. (2018).
*To be included in version 12 (November 2020).
**The modules in themselves are free and open source, but they run in non-free and closed source software environments (MATLAB and STATA, respectively).

and user-specified goodness-of-fit criteria. The aim of this article is to illustrate how user-specified GETS and ISAT methods can be implemented.

The rest of this article contains four sections. In the next section the model selection properties of GETS and ISAT methods are summarised. This is followed by a section that outlines the general principles of how user-specified estimation, user-specified diagnostics and user-specified goodness-of-fit measures are implemented. Next, a section with four illustrations follows. The final section contains a summary.

Model selection properties of GETS and ISAT methods

It is useful to denote a generic model for observation t as

$$m\left(y_t, \boldsymbol{x}_t, \boldsymbol{\beta}\right), \qquad t = 1, 2, \ldots, n, \tag{1}$$

where y_t is the dependent variable, $\boldsymbol{x}_t = (x_{1t}, x_{2t}, \ldots)'$ is a vector of covariates, $\boldsymbol{\beta} = (\beta_1, \beta_2, \ldots)'$ is a vector of parameters to be estimated and n is the sample size. Two examples are the linear regression model and the logit-model:

$$y_t = \beta_1 x_{1t} + \cdots + \beta_k x_{kt} + \epsilon_t, \tag{2}$$

$$Pr\left(y_t = 1 | \boldsymbol{x}_t\right) = \frac{1}{1 + \exp\left(-h_t\right)} \qquad \text{with} \qquad h_t = \beta_1 x_{1t} + \cdots + \beta_k x_{kt}. \tag{3}$$

Note that, in a generic model $m(y_t, \boldsymbol{x}_t, \boldsymbol{\beta})$, the dimension $\boldsymbol{\beta}$ is usually – but not necessarily – equal to the dimension of $\boldsymbol{x}_t$. Here, for notational convenience, they will both have dimension k unless otherwise stated.

In (2)–(3), a variable $x_{jt} \in \boldsymbol{x}_t$ is said to be relevant if $\beta_j \neq 0$ and irrelevant if $\beta_j = 0$. Let $k_{\text{rel}} \geq 0$ and $k_{\text{irr}} \geq 0$ denote the number of relevant and irrelevant variables, respectively, such that $k_{\text{rel}} + k_{\text{irr}} = k$. GETS modelling aims at finding a specification that contains as many relevant variables as possible, and a proportion of irrelevant variables that on average equals the significance level α chosen by the investigator. Put differently, if $\widehat{k}_{\text{rel}}$ and $\widehat{k}_{\text{irr}}$ are the retained number of relevant and irrelevant variables in an empirical application, respectively, then GETS modelling aims at satisfying

$$\mathbb{E}\left(\widehat{k}_{\text{rel}}/k_{\text{rel}}\right) \to 1 \quad \text{and} \quad \mathbb{E}\left(\widehat{k}_{\text{irr}}/k_{\text{irr}}\right) \to \alpha \quad \text{as} \quad n \to \infty, \tag{4}$$

when $k_{\text{rel}}, k_{\text{irr}} > 0$. If either $k_{\text{rel}} = 0$ or $k_{\text{irr}} = 0$, then the targets are modified in natural ways: If $k_{\text{rel}} = 0$, then the first target is $\mathbb{E}\left(\widehat{k}_{\text{rel}}\right) = 0$, and if $k_{\text{irr}} = 0$, then the second target is $\mathbb{E}\left(\widehat{k}_{\text{irr}}\right) = 0$. Sometimes, the irrelevance proportion $\widehat{k}_{\text{irr}}/k_{\text{irr}}$ is also referred to as *gauge*, whereas the relevance proportion $\widehat{k}_{\text{irr}}/k_{\text{irr}}$ is also referred to as *potency*.

In targeting a relevance proportion equal to 1 and an irrelevance proportion equal to α, GETS modelling combines well-known ingredients: Multi-path backwards elimination, tests on the β_j's (both single and multiple hypothesis tests), diagnostics tests and fit-measures (e.g. information criteria). Let $V\left(\widehat{\boldsymbol{\beta}}\right)$ denote the estimated coefficient-covariance. GETS modelling in the package **gets** can be described as proceeding in three steps:[3]

1. Formulate a General Unrestricted Model (GUM), i.e. a starting model, that passes a set of chosen diagnostic tests. A regressor x_j in the GUM is non-significant if the p-value of a two-sided t-test is lower than the chosen significance level α, and each non-significant regressor constitutes the starting point of a backwards elimination path. The test-statistics of the t-tests are computed as $\widehat{\beta}_j/se\left(\widehat{\beta}_j\right)$, where $se\left(\widehat{\beta}_j\right)$ is the square root of the jth. element of the diagonal of $V\left(\widehat{\boldsymbol{\beta}}\right)$.
2. Undertake backwards elimination along multiple paths by removing, one-by-one, non-significant regressors as determined by the chosen significance level α. Each removal is checked for validity against the chosen set of diagnostic tests, and for parsimonious encompassing (i.e. a multiple hypothesis test) against the GUM. These multiple hypothesis tests on subsets of $\boldsymbol{\beta}$ are implemented as Wald-tests.
3. Multi-path backwards elimination can result in multiple terminal models. The last step of GETS modelling consists of selecting, among the terminal models, the specification with the best fit according to a fit-criterion, e.g. the Schwarz (1978) information criterion.

In ISAT methods, the vector $\boldsymbol{x}_t$ contains at least $n-1$ indicators in addition to other covariates that are considered. Accordingly, standard estimation methods are infeasible, since the number of variables in $\boldsymbol{x}_t$ is usually larger than the number of observations n. The solution to this problem provided by ISAT methods is to first organise $\boldsymbol{x}_t$ into B blocks: $\boldsymbol{x}_t^{(1)}, \ldots, \boldsymbol{x}_t^{(B)}$. These blocks need not be mutually exclusive, so a variable or subset of variables can appear in more than one block. Next, GETS modelling is applied to each block, which leads to B final models. Note that, in the `isat` function, the default is that no diagnostic tests are undertaken. Finally, a new round of GETS modelling is undertaken with the union of the retained variables from the B blocks as covariates in a new starting model (i.e. a new GUM). The model selection properties targeted by ISAT methods are the same as those of GETS methods. Note, however, that since the starting model (the GUM) contains at least $n-1$ regressors, a tiny significance level – e.g. $\alpha = 0.001$ or smaller – is usually recommended in ISAT methods.

User-specification: General principles

In the current version of the package **gets**, version 0.25, the functions that admit user-specified estimation are `arx`, `getsm`, `getsFun` and `isat`.[4] The user-specification principles are the same in all four. However, if the result (i.e. a list) returned from the user-specified estimator does not have the same structure as that returned from the default estimator `ols` (part of the **gets** package), then `arx`, `getsm` and `isat` may not always work as expected. This is particularly the case with respect to their extraction functions (e.g. `print`, `coef`, `residuals` and `predict`). User-specified diagnostics and goodness-of-fit functions are optional. By default, `getsFun` and `isat` do not perform any diagnostics tests, whereas the default in `arx` and `getsm` is to test the standardised residuals for autocorrelation and Autoregressive Heteroscedasticity (ARCH). This is implemented via the `diagnostics` function (part of the **gets** package). Also by default, all four functions use the Schwarz (1978) information criterion as goodness-of-fit measure, which favours parsimony, via the `infocrit` function (part of the **gets** package).

[3]The way GETS modelling is implemented across softwares varies. For example, in Autometrics and Grocer the diagnostics are not checked at each deletion.

[4]In the future, the plan is to also enable user-specified GETS modelling with the function `getsv`, which implements GETS modelling of the log-variance.

The getsFun function

The recommended, most flexible and computationally most efficient approach to user-specified GETS modelling is via the `getsFun` function. Currently, it accepts up to twenty-five arguments. For the details of all these arguments, the reader is referred to the discussion of the `getsm` function (Section 5) in Pretis et al. (2018), and the help pages of `getsFun` (type `?getsFun`). For the purpose of user-specified estimation, user-specified diagnostics and user-specified goodness-of-fit measures, the most important arguments are:

```
getsFun(y, x,
  user.estimator = list(name = "ols"),
  user.diagnostics = NULL,
  gof.function = list(name = "infocrit", method = "sc"),
  gof.method = c("min", "max"),
  ...)
```

The `y` is the left-hand side variable (the regressand), `x` is the regressor or design matrix, `user.estimator` controls which estimator or model to use and further arguments – if any – to be passed on to the estimator, `user.diagnostics` controls the user-specified diagnostics if any, and `gof.function` and `gof.method` control the goodness-of-fit measure used. Note that `y` and `x` should satisfy '`is.vector(y) == TRUE`' and '`is.matrix(x) == TRUE`', respectively, and enter in "clean" ways: If either `y` or `x` are objects of class, say, `"ts"` or `"zoo"`, then `getsFun` may not behave as expected. By default, the estimator `ols` is used with its default arguments, which implements OLS estimation via the `qr` function. The value `NULL` on `user.diagnostics` means no diagnostics checks are undertaken by default. The following code illustrates `getsFun` in linear regression (the default), and reproduces the information printed while searching:

```
n <- 40 #number of observations
k <- 20 #number of Xs

set.seed(123) #for reproducibility
y <- rnorm(n) #generate Y
x <- matrix(rnorm(n*k), n, k) #create matrix of Xs

#do gets with default estimator (ols), store output in 'result':
result <- getsFun(y, x)

#the information printed while searching:
18 path(s) to search
Searching: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
```

The object named `result` is a list, and the code `summary(results)` returns a summary of its contents. The most important entries are:

- `paths`: A list of vectors containing the searched paths. Each vector (i.e. path) indicates the sequence of deletion of the regressors. In the example above the first path is

  ```
  $paths[[1]]
  [1]  1 15  6  7  3 14 11 16  4  2  8 12  5  9 20 19 13
  ```

  That is, regressor no. 1 was the first to be deleted, regressor no. 15 was the second, regressor no. 6 was the third, and so on. If the regressors in `x` were named, then a name-representation of the first deletion path is obtained with `colnames(x)[ paths[[1]] ]`.

- `terminals`: A list of vectors with the distinct terminal models of the specification search. In the example above it is equal to

  ```
  $terminals
  $terminals[[1]]
  [1] 10 17 18

  $terminals[[2]]
  [1] 10 18
  ```

  That is, two terminal models. The first contains regressors 10, 17 and 18, whereas the second contains regressors 10 and 18.

- `terminals.results`: A data frame with the goodness-of-fit information of the terminal models. In the above example the entry is equal to:

```
$terminals.results
        info(sc)      logl  n k
spec 1: 2.514707 -44.76081 40 3
spec 2: 2.529923 -46.90958 40 2
```

`spec 1` is short for specification 1, i.e. terminal model 1, and `spec 2` is short for specification 2, i.e. terminal model 2. `info(sc)` indicates that the Schwarz (1978) criterion (the default) is used as goodness-of-fit measure, whereas `n` and `k` denote the number of observations and parameters, respectively.

- `best.terminal`: An integer that indicates which terminal model is the best according to the goodness-of-fit criterion used. In the example above the value is 1.
- `specific.spec`: A vector of integers that indicates which regressors that are contained in the best terminal model. In the above example it is

```
$specific.spec
[1] 10 17 18
```

That is, the best terminal model contains regressors no. 10, 17 and 18.

User-specified estimation

User-specified estimation is carried out via the `user.estimator` argument. By default, the argument is `NULL`, so no diagnostic tests are undertaken. To carry out diagnostics, the argument must be specified as a list containing at least one entry – a character – named `name` with the name of the estimator to be invoked.[5] Optionally, the list can also contain an item named `envir`, a character, which indicates the environment in which the user-specified estimator resides. Additional entries in the list, if any, are passed on to the estimator as arguments.

The user-specified estimator must also satisfy the following:

1. It should be of the form `myEstimator(y,x,...)`, where `y` is a vector and `x` is a matrix. In other words, while the name of the function is arbitrary, the first argument should be the regressand and the second the matrix of covariates.
2. The user-defined estimator should return a `list` with a minimum of six items:
   - `n` (the number of observations)
   - `k` (the number of coefficients)
   - `df` (degrees of freedom, used in the *t*-tests)
   - `coefficients` (a vector with the coefficient estimates)
   - `vcov` (the coefficient covariance matrix)
   - `logl` (a goodness-of-fit value, e.g. the log-likelihood)

   The items need not appear in this order. However, the naming should be exactly as indicated. If also the diagnostics and/or the goodness-of-fit criterion is user-specified, then additional objects may be required, see the subsections below on user-specified diagnostics and goodness-of-fit criteria. Note also that, if the goodness-of-fit criterion is user-specified, then `logl` can in certain situations be replaced by another item (which needs not be named `logl`).
3. The user-defined estimator must be able to handle `NULL` regressor-matrices, i.e. situations where either `NCOL(x)` is 0 or `is.null(x)` is `TRUE`. This is needed in situations where a terminal model is empty (i.e. no regressors are retained).

To illustrate how the requirements above can be met in practice, suppose – as an example – that we would like to use the function `lm` for estimation rather than `ols`. The first step is then to make a function that calls `lm` while satisfying requirements 1 to 3:

```
lmFun <- function(y, x, ...){

  ##create list:
  result <- list()

  ##n, k and df:
  result$n <- length(y)
```

[5]To carry out the same diagnostics as the default of the `getsm` function, the argument can be set to `list(name = "diagnostics", pval = c(0.025, 0.025))`.

```
    if( is.null(x) || NCOL(x) == 0 ){
      result$k <- 0
    }else{
      result$k <- NCOL(x)
    }
    result$df <- result$n - result$k

    ##call lm if k > 0:
    if( result$k > 0){
      tmp <- lm(y ~ x - 1)
      result$coefficients <- coef(tmp)
      result$vcov <- vcov(tmp)
      result$logl <- as.numeric(logLik(tmp))
    }else{
      result$coefficients <- NULL
      result$vcov <- NULL
      result$logl <- sum(dnorm(y, sd = sqrt(var(y)), log = TRUE))
    }

    ##return result:
    return(result)

  }
```

The code

```
getsFun(y, x, user.estimator = list(name = "lmFun"))
```

undertakes the same specification search as earlier, but uses `lmFun` rather than `ols`.

User-specified diagnostics

User-specified diagnostics is carried out via the `user.diagnostics` argument. The argument must be a list containing at least two entries: A character named `name` containing the name of the diagnostics function to be called, and an entry named `pval` that contains a vector with values between 0 and 1, i.e. the chosen significance level(s) for the diagnostics test(s).[6] If only a single test is undertaken by the diagnostics function, then `pval` should be of length one. If two tests are undertaken, then `pval` should be of length two. And so on. An example of the argument when only a single test is undertaken is:

```
user.diagnostics = list(name = "myDiagnostics", pval = 0.05))
```

That is, the name of the function is `myDiagnostics`, and the chosen significance level for the single test that is carried out is 5%. Optionally, just as when the estimator is user-specified, the list can contain an item named `envir`, a character, which indicates the environment in which the user-specified diagnostics function resides. Additional items in the list, if any, are passed on to the user-specified function as arguments.

The user-specified diagnostics function must satisfy the following:

1. It should be of the form `myDiagnostics(result,...)`, where `result` is the list returned from the estimator in question, e.g. that of the user-specified estimator (recall requirement 2 in the previous section above).
2. It should return an $m \times 3$ matrix that contains the p-value(s) of the test(s) in the third column, where $m \geq 1$ is the number of tests carried out. So if only a single test is carried out, then $m = 1$ and the p-value should be contained in the third column. An example could look like:

   ```
               statistic df   pval
   normality          NA NA 0.0734
   ```

   Note that the row-names and column-names in the example are not required. However, they do indicate what kind of information you may wish to put there for reporting purposes, e.g. by using the function `diagnostics` (also part of the **gets** package).

[6]A word of caution is required. Let $R^{(i)}$ denote the event of rejecting the null, under the null, for a significance level $\alpha^{(i)}$ in diagnostic test i. For example, if only a single test is undertaken so that $i = 1$, then $Pr(R^{(1)}) = \alpha^{(1)}$. If two tests are undertaken, however, then the probability of rejecting in one or both tests is $Pr\left(R^{(1)} \cup R^{(2)}\right)$. As rule of thumb, therefore, to control the overall error, it is recommended that the significance level of each diagnostic test is set equal to α/m, where α is the overall or total significance level targeted by the user, and m is the number of diagnostic tests. This is sometimes referred to as a Bonferroni correction.

To illustrate how the requirements can be met in practice, suppose we would like to ensure the residuals are normal by testing for non-normality with the Shapiro-Wilks test function shapiro.test. In this context, its main argument is the residuals of the estimated model. The list returned by the user-defined estimator named lmFun above, however, does not contain an item with the residuals. The first step, therefore, is to modify the estimator lmFun so that the returned list also contains the residuals:

```
lmFun <- function(y, x, ...){

  ##info needed for estimation:
  result <- list()
  result$n <- length(y)
  if( is.null(x) || NCOL(x)==0 ){
    result$k <- 0
  }else{
    result$k <- NCOL(x)
  }
  result$df <- result$n - result$k
  if( result$k > 0){
    tmp <- lm(y ~ x - 1)
    result$coefficients <- coef(tmp)
    result$vcov <- vcov(tmp)
    result$logl <- as.numeric(logLik(tmp))
  }else{
    result$coefficients <- NULL
    result$vcov <- NULL
    result$logl <- sum(dnorm(y, sd=sqrt(var(y)), log=TRUE))
  }

  ##residuals:
  if( result$k > 0){
    result$residuals <- residuals(tmp)
  }else{
    result$residuals <- y
  }

  return(result)
}
```

Computationally, the only modification appears under ##residuals. We can now make the user-specified diagnostics function:

```
myDiagnostics <- function(x, ...){
  tmp <- shapiro.test(x$residuals) #do the test
  result <- rbind( c(tmp$statistic, NA, tmp$p.value) )
  return(result)
}
```

The following code undertakes GETS modelling with the user-specified estimator defined above, and the user-specified diagnostics function using a 5% significance level for the latter:

```
getsFun(y, x, user.estimator = list(name = "lmFun"),
  user.diagnostics = list(name = "myDiagnostics", pval = 0.05))
```

Note that if the chosen significance level for the diagnostics is sufficiently high, then no specification search is undertaken because the starting model does not pass the non-normality test. With the current data, for example, a little bit of trial and error reveals this is the case for a level of about pval = 0.35.

User-specified goodness-of-fit

User-specified goodness-of-fit is carried out with the gof.function and gof.method arguments. The former indicates which Goodness-of-Fit (GOF) function to use, and the second is a character that indicates whether the best model maximises ("max") or minimises ("min") the GOF criterion in question. The first argument is a list with a structure similar to earlier: It must contain at least one entry, a character named name, with the name of the GOF function to call. An example is:

```
gof.function = list(name = "myGof"))
```

Optionally, also here the list can contain an item named `envir`, a character, which indicates the environment in which the user-specified GOF function resides. Also as earlier, additional items in the list are passed on to the user-specified GOF function as arguments. The default value, for example, `gof.function = list(name = "infocrit",method = "sc")`, means the argument `method = "sc"` is passed on to the function `infocrit`, see the help pages of `infocrit` (type `?infocrit`) for information on the methods available via this function. The user-specified GOF function must satisfy the following:

1. It should be of the form `myGof(result,...)`, where `result` is the list returned from the estimator in question, e.g. that of the user-specified estimator.
2. It should return a single numeric value, i.e. the value of the GOF measure in question.

To illustrate how the requirements can be met in practice, suppose we would like to use the adjusted R^2 as our GOF measure in combination with our user-defined estimator. For the moment, the user-defined estimator `lmFun` does not contain the information necessary to compute the adjusted R^2. In particular, it lacks the regressand `y`. However, this is readily added:

```
lmFun <- function(y, x, ...){

  ##info needed for estimation:
  result <- list()
  result$n <- length(y)
  if( is.null(x) || NCOL(x)==0 ){
    result$k <- 0
  }else{
    result$k <- NCOL(x)
  }
  result$df <- result$n - result$k
  if( result$k > 0){
    tmp <- lm(y ~ x - 1)
    result$coefficients <- coef(tmp)
    result$vcov <- vcov(tmp)
    result$logl <- as.numeric(logLik(tmp))
  }else{
    result$coefficients <- NULL
    result$vcov <- NULL
    result$logl <- sum(dnorm(y, sd=sqrt(var(y)), log=TRUE))
  }

  ##residuals:
  if( result$k > 0){
    result$residuals <- residuals(tmp)
  }else{
    result$residuals <- y
  }

  ##info needed for r-squared:
  result$y <- y

  return(result)
}
```

The added part appears under `##info needed for r-squared`. A GOF function that returns the adjusted R^2 is:

```
myGof <- function(object, ...){
  TSS <- sum((object$y - mean(object$y))^2)
  RSS <- sum(object$residuals^2)
  Rsquared <- 1 - RSS/TSS
  result <- 1 - (1 - Rsquared) * (object$n - 1)/(object$n - object$k)
  return(result)
}
```

The following code undertakes GETS modelling with all the three user-specified functions defined so far:

```
getsFun(y, x, user.estimator = list(name = "lmFun"),
  user.diagnostics = list(name = "myDiagnostics", pval = 0.05),
  gof.function = list(name = "myGof"), gof.method = "max")
```

Incidentally, it leads to the same final model as when the default GOF function is used.

More speed: turbo, max.paths, parallel computing

In multi-path backwards elimination search, one may frequently arrive at a specification that has already been estimated and tested. As an example, consider the following two paths:

```
$paths[[1]]
[1]  2 4 3 1 5

$paths[[2]]
[1]  4 2 3 1 5
```

In path 1, i.e. `paths[[1]]`, regressor no. 2 is the first to be deleted, regressor no. 4 is the second, and so on. In path 2 regressor no. 4 is the first to be deleted, regressor no. 2 is the second, and so on. In other words, after the deletion of the first two variables, the set of remaining variables (i.e. 3, 1 and 5) in the two paths is identical. Accordingly, knowing the result from the first path, in path 2 it is unnecessary to proceed further after having deleted the first two regressors. Setting the argument `turbo` equal to `TRUE` turns such a check on, and thus skips searches, estimations and tests that are unnecessary. The turbo comes at a small computational cost (often less than 1 second), since the check is undertaken at each deletion. This is why the default is `turbo = FALSE`. However, if the estimation time is noticeable, then turning the turbo on can reduce the search time substantially. As a rule of thumb, if each estimation takes 1 second or more, then turning the turbo on will (almost) always reduce the total search time.

Searching more paths may increase the relevance proportion or potency. Whether and to what extent this happens depends on the sample size n, and on the degree of multicolinearity among the regressors $\boldsymbol{x}_t$. If n is sufficiently large, or if the regressors are sufficiently uncorrelated, then searching fewer paths will not reduce the relevance proportion. In many situations, therefore, one may consider reducing the number of paths to increase the speed. This is achieved via the `max.paths` argument. Setting `max.paths = 10`, for example, means a maximum of 10 paths is searched. The paths that are searched are those of the 10 most insignificant variables (i.e. those with the highest p-values) in the starting model.

When implementing ISAT methods, the function `isat` undertakes a multi-round form of GETS modelling. In the first round the variables are split into B blocks, and then GETS modelling is undertaken on each block. This is a socalled "embarassingly parallel" problem. To make `isat` search in parallel during the first round, simply set the argument `parallel.options` equal to an integer greater than 1. The integer determines how many cores/threads to use, and the command `detectCores()` can be used to find out how many cores/threads that are available on the current machine. Remember, it is not recommended to use all the cores/threads available. Within `isat`, parallel-computing is implemented with the `makeCluster` and `parApply` functions from the package **parallel**. If the time required by `makeCluster` to set up parallel computing is negligible relative to the total computing time (on an average computer the setup-time is about 1 second), then the total computing time may – in optimal situations – be reduced by a factor of about $m - 0.8$, where $m > 1$ is the number of cores/threads used for parallel computing.

User-specified GETS and ISAT methods: Illustrations

GETS modelling of Generalised Linear Models (GLMs)

The function `glm` enables the estimation of a large number of specifications within the class of Generalised Linear Models (GLMs). Here, it is illustrated how GETS modelling can be implemented with GLMs. To fix ideas, the illustration is in terms of the logit-model.

Let $y_t \in \{0, 1\}$ denote the regressand of the logit-model given by

$$Pr\left(y_t = 1 | \boldsymbol{x}_t\right) = \frac{1}{1 + \exp\left(-h_t\right)}, \qquad h_t = \boldsymbol{\beta}' \boldsymbol{x}_t. \tag{5}$$

Next, consider the following set of data:

```
n <- 40 #number of observations
k <- 20 #number of Xs
set.seed(123) #for reproducibility
y <- round(runif(40)) #generate Y
x <- matrix(rnorm(n*k), n, k) #create matrix of Xs
```

In other words, one regressand $y_t \in \{0,1\}$ which is entirely independent of the 20 regressors in x_t. The following function enables GETS modelling of logit-models:

```
logitFun <- function(y, x, ...){

  ##create list:
  result <- list()

  ##n, k and df:
  result$n <- length(y)
  if( is.null(x) || NCOL(x)==0 ){
    result$k <- 0
  }else{
    result$k <- NCOL(x)
  }
  result$df <- result$n - result$k

  ##call glm if k > 0:
  if( result$k > 0){
    tmp <- glm(y ~ x - 1, family = binomial(link="logit"))
    result$coefficients <- coef(tmp)
    result$vcov <- vcov(tmp)
    result$logl <- as.numeric(logLik(tmp))
  }else{
    result$coefficients <- NULL
    result$vcov <- NULL
    result$logl <- result$n*log(0.5)
  }

  ##return result:
  return(result)
}
```

To undertake the GETS modelling:

```
getsFun(y, x, user.estimator=list(name="logitFun"))
```

Two variables are retained, namely x_{5t} and x_{11t}, at the default significance level of 5% (i.e. `t.pval = 0.05`). To reduce the chance of retaining irrelevant variables, the significance level can be lowered to, say, 1% by setting `t.pval = 0.01`.

To implement GETS modelling for a different GLM model, only two lines of code needs to be modified in the user-defined function above. The first is the line that specifies the `family`, and the second is the one that contains the log-likelihood associated with the empty model (i.e. the line '`result$logl <-result$n*log(0.5)`').

Creating a `gets` method (S3) for the `"lm"` class of models

The package **gets** provides the generic function `gets`. This enables the creation of GETS methods (S3) for models of arbitrary classes (type `?S3Methods` for more info on S3 methods). Here, this is illustrated for models of class `"lm"`. Our aim is to be able to do the following:

```
mymodel <- lm(y ~ x)
gets(mymodel)
```

That is, to first estimate a model of class `"lm"` where `x` is a matrix of regressors, and then to conveniently undertake GETS modelling by simply applying the code `gets(.)` to the named object `mymodel`. To this end, a function named `gets.lm` that relies on `getsFun` will be created. In doing so, a practical aspect is how to appropriately deal with the intercept codewise. Indeed, as we will see, a notable part of the code in the user-defined function will be devoted to the intercept. The reason for this is that `lm` includes the intercept by default. Another practical aspect is whether to use `lm` or `ols` whenever a model is estimated by OLS (both employ the QR decomposition). The latter is simpler codewise and computationally faster, so we opt for the latter. The function is:

```
gets.lm <- function(object, ...){

  ##make y:
  y <- as.vector(object$model[, 1])
  yName <- names(object$model)[1]

  ##make x:
  x <- as.matrix(object$model[, -1])
  xNames <- colnames(x)
  if(NCOL(x) == 0){
    x <- NULL
    xNames <- NULL
  }else{
    if(is.null(xNames)){
      xNames <- paste0("X", 1:NCOL(x))
      colnames(x) <- xNames
    }
  }

  ##is there an intercept?:
  if(length(coef(object)) > 0){
    cTRUE <- names(coef(object))[1] == "(Intercept)"
    if(cTRUE){
      x <- cbind(rep(1, NROW(y)), x)
      xNames <- c("(Intercept)", xNames)
      colnames(x) <- xNames
    }
  }

  ##do gets:
  myspecific <- getsFun(y, x, ...)

  ##which are the retained regressors?:
  retainedXs <- xNames[myspecific$specific.spec]
  cat("Retained regressors:\n ", retainedXs, "\n")

  ##return result
  return(myspecific)

}
```

Next, recall the Data Generation Process (DGP) of the first experiment:

```
n <- 40 #number of observations
k <- 20 #number of Xs
set.seed(123) #for reproducibility
y <- rnorm(n) #generate Y
x <- matrix(rnorm(n*k), n, k) #create matrix of Xs
```

We can now do GETS modelling on models of class "lm" by simply applying the code 'gets(...)' on the object in question. The following code first stores an estimated model of class "lm" in an object named startmodel, and then applies the newly defined function gets.lm to it:

```
startmodel <- lm(y ~ x)
finallm <- gets(startmodel)
```

The information from the specification search is stored in the object called finallm, and during the search the following is printed:

```
18 path(s) to search
Searching: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
Retained regressors:
X10 X17 X18
```

In other words, the retained regressors are no. 10, 17 and 18. Note that, due to the way the user-defined function has been put together, the intercept is excluded from deletion.

Regression with ARMA error

The function `arima` can be used to estimate a linear regression with deterministic regressors and an error-term that follows an ARMA. An example is

$$y_t = \boldsymbol{\beta}'\boldsymbol{x}_t + \epsilon_t, \qquad \epsilon_t = \phi_1\epsilon_{t-1} + \theta_1 u_{t-1} + u_t, \qquad u_t \sim WN\left(0, \sigma_u^2\right),$$

where $\boldsymbol{x}_t$ is a vector of deterministic regressors and WN is short for White Noise. The error ϵ_t is thus governed by an ARMA(1,1). Let x_t denote a (deterministic) step-shift variable in which the step-shift occurs at observation 30, i.e. $x_t = 1\,(t \geq 30)$. Next, consider the DGP given by

$$y_t = 4x_t + \epsilon_t, \qquad \epsilon_t = 0.4\epsilon_{t-1} + 0.1u_{t-1} + u_t, \qquad u_t \sim N(0,1), \qquad t = 1, \ldots, n \tag{6}$$

with $n = 60$. In other words, the series y_t is non-stationary and characterised by a large location shift at $t = 30$. Figure 1 illustrates the evolution of y_t, which is generated with the following code:

```
set.seed(123) #for reproducibility
eps <- arima.sim(list(ar = 0.4, ma = 0.1), 60) #epsilon
x <- coredata(sim(eps, which.ones = 30)) #step-dummy at t = 30
y <- 4*x + eps #the dgp
plot(y, ylab="y", xlab="t", lwd = 2)
```

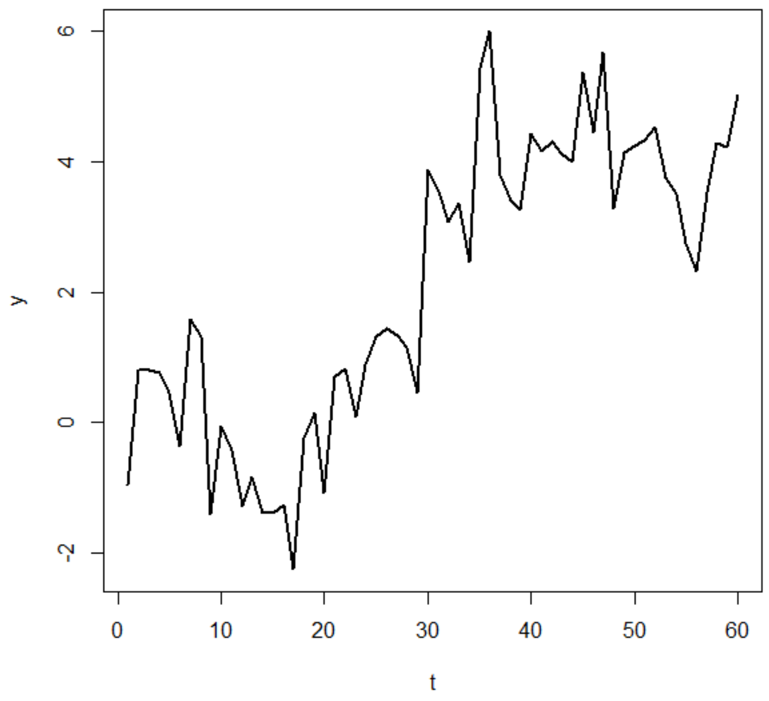


Figure 1: The graph of y_t as given in (6).

By just looking at the graph, it seems clear that there is a location shift, but it is not so clear that it in fact occurs at $t = 30$. I now illustrate how the `arima` function can be used in combination with `getsFun` to automatically search for where the break occurs. The idea is to do GETS modelling over a set or block of step-indicators that cover the period in which the break visually appears to be in. Specifically, the aim is to apply GETS modelling to the following starting model with 11 regressors:

$$y_t = \sum_{i=1}^{11} \beta_i \cdot 1_{\{t \geq 24+i\}} + \epsilon_t, \qquad \epsilon_t = \phi_1\epsilon_{t-1} + \theta_1 u_{t-1} + u_t.$$

To this end, we first need to make the user-specified estimator:

```
myEstimator <- function(y, x){
    ##create list:
    result <- list()

    ##estimate model:
    if( is.null(x) || NCOL(x)==0 ){
      result$k <- 0
      tmp <- arima(y, order = c(1,0,1)) #empty model
    }else{
      result$k <- NCOL(x)
      tmp <- arima(y, order = c(1,0,1), xreg = x)
      result$coefficients <- tmp$coef[-c(1:3)]
      result$vcov <- tmp$var.coef
      result$vcov <- result$vcov[-c(1:3),-c(1:3)]
    }

    ##rename and re-organise things:
    result$n <- tmp$nobs
    result$df <- result$n - result$k
    result$logl <- tmp$loglik

    return(result)
  }
```

Note that the estimator has been put together such that the ARMA(1,1) specification of the error ϵ_t is fixed. As a consequence, the specification search is only over the regressors. The following code first creates the 11 step dummies, and then undertakes the GETS modelling:

```
  xregs <- coredata(sim(eps, which.ones = 25:35)) #11 step-dummies
  getsFun(y, xregs, user.estimator = list(name = "myEstimator"))
```

Two step-dummies are retained, namely those of $t = 30$ and $t = 35$.

Faster ISAT when n is large

ISAT methods are computationally intensive, since at least $n - 1$ indicators are included as regressors. Accordingly, as n grows large, purpose-specific estimators can greatly reduce the computing time. One way of building such an estimator is by using tools from the package **Matrix**, see Bates and Maechler (2018). The code below illustrates this. First it loads the library, and then it creates a function named `olsFaster` that re-produces the structure of the estimation result returned by the function `ols` with `method = 3` (i.e. OLS with the ordinary coefficient-covariance), but with functions from **Matrix**. The code is:

```
  library(Matrix)
  olsFaster <- function(y, x){
    out <- list()
    out$n <- length(y)
    if (is.null(x)){ out$k <- 0 }else{ out$k <- NCOL(x) }
      out$df <- out$n - out$k
      if (out$k > 0) {
        x <- as(x, "dgeMatrix")
        out$xpy <- crossprod(x, y)
        out$xtx <- crossprod(x)
        out$coefficients <- as.numeric(solve(out$xtx,out$xpy))
        out$xtxinv <- solve(out$xtx)
        out$fit <- out$fit <- as.vector(x %*% out$coefficients)
    }else{ out$fit <- rep(0, out$n)        }
    out$residuals <- y - out$fit
    out$residuals2 <- out$residuals^2
    out$rss <- sum(out$residuals2)
    out$sigma2 <- out$rss/out$df
    if (out$k > 0) { out$vcov <- as.matrix(out$sigma2 * out$xtxinv) }
    out$logl <- -out$n * log(2 * out$sigma2 * pi)/2 - out$rss/(2 * out$sigma2)
    return(out)
  }
```

Depending on the data and hardware/software configuration, the estimator may lead to a considerably speed-improvement. In the following example, the function `system.time` suggests a speed improvement of about 20% on the current hardware/software configuration:

```
set.seed(123) #for reproducibility
y <- rnorm(1000)
x <- matrix(rnorm(length(y)*20), length(y), 20)
#with ols:
system.time( finalmodel <- isat(y, mxreg = x, max.paths = 5))
#with olsFaster:
system.time( finalmodel <- isat(y, mxreg = x, max.paths = 5,
  user.estimator = list(name = "olsFaster")) )
```

Summary

In many applications a specific model or estimator is needed, a specific set of diagnostics tests may be required, or a specific fit criterion is preferred. In these situations, if the combination of estimator/model, diagnostics tests and fit criterion is not already offered in a pre-programmed way by publicly available software, the implementation of user-specified GETS and ISAT methods puts a large programming-burden on the user. This article has outlined how recent additions to the package **gets** greatly simplifies the development of user-specified GETS and ISAT methods. The package is the first software – both inside and outside the R universe – to provide a complete set of facilities for user-specified GETS and ISAT methods.

Acknowledgements

I am grateful to the Editor Catherine Hurley, an anonymous reviewer, Damian Clarke, Éric Dubois, Jonas Kurle, Felix Pretis, James Reade, Moritz Schwarz, Gareth Thomas, participants at the UseR! 2019 conference (Toulouse, July), the Statistics Norway seminar (May, 2018) and the Norges Bank seminar (April, 2018) for their helpful comments, suggestions and questions.

Bibliography

D. Bates and M. Maechler. *Matrix: Sparse and Dense Matrix Classes and Methods*, 2018. URL `https://CRAN.R-project.org/package=Matrix`. R package version 1.2-15. [p263]

J. Campos, D. F. Hendry, and N. R. Ericsson, editors. *General-to-Specific Modeling. Volumes 1 and 2*. Edward Elgar Publishing, Cheltenham, 2005. [p251]

J. Castle, J. Doornik, D. F. Hendry, and F. Pretis. Detecting Location Shifts During Model Selection by Step-Indicator Saturation. *Econometrics*, 3:240–264, 2015. URL `https://doi.org/10.3390/econometrics3020240`. [p251]

D. Clarke. General-to-specific modeling in Stata. *The Stata Journal*, 14:895–908, 2014. URL `https://www.stata-journal.com/article.html?article=st0365`. [p252]

J. A. Doornik and D. F. Hendry. *Empirical Econometric Modelling - PcGive 15*. Timberlake Consultants Ltd., London, 2018. [p252]

É. Dubois and E. Michaux. Grocer 1.8: an econometric toolbox for Scilab. 2019. URL `http://dubois.ensae.net/grocer.html`. [p252]

D. F. Hendry, S. Johansen, and C. Santos. Automatic selection of indicators in a fully saturated regression. *Computational Statistics*, 23:317–335, 2008. URL `https://doi.org/10.1007/s00180-007-0054-z`. [p251]

K. D. Hoover and S. J. Perez. Data Mining Reconsidered: Encompassing and the General-to-Specific Approach to Specification Search. *Econometrics Journal*, 2:167–191, 1999. URL `https://doi.org/10.1111/1368-423X.00025`. [p251, 252]

IHS Markit. *EViews Version 11*. IHS Markit, Irvine, 2020. URL `http://www.EViews.com/`. [p252]

M. C. Lovell. Data Mining. *The Review of Economics and Statistics*, 65:1–12, 1983. URL `https://doi.org/10.2307/1924403`. [p251]

F. Pretis, J. Reade, and G. Sucarrat. Automated General-to-Specific (GETS) Regression Modeling and Indicator Saturation for Outliers and Structural Breaks. *Journal of Statistical Software*, 86:1–44, 2018. URL https://doi.org/10.18637/jss.v086.i03. [p251, 252, 254]

G. Schwarz. Estimating the Dimension of a Model. *The Annals of Statistics*, 6:461–464, 1978. [p253, 255]

G. Sucarrat, J. Kurle, F. Pretis, J. Reade, and M. Schwarz. *gets: General-to-Specific (GETS) Modelling and Indicator Saturation (ISAT) Methods*, 2020. URL https://CRAN.R-project.org/package=gets. R package version 0.25. [p252]

Genaro Sucarrat
BI Norwegian Business School
Nydalsveien 37, 0484 Oslo
Norway
genaro.sucarrat@bi.no

Kuhn-Tucker and Multiple Discrete-Continuous Extreme Value Model Estimation and Simulation in R: The rmdcev Package

by Patrick Lloyd-Smith

Abstract This paper introduces the package **rmdcev** in R for estimation and simulation of Kuhn-Tucker demand models with individual heterogeneity. The models supported by **rmdcev** are the multiple-discrete continuous extreme value (MDCEV) model and Kuhn-Tucker specification common in the environmental economics literature on recreation demand. Latent class and random parameters specifications can be implemented and the models are fit using maximum likelihood estimation or Bayesian estimation. The **rmdcev** package also implements demand forecasting and welfare calculation for policy simulation. The purpose of this paper is to describe the model estimation and simulation framework and to demonstrate the functionalities of **rmdcev** using real datasets.

Introduction

Individual choice contexts are often characterized by both extensive (i.e. what alternative to choose) and intensive (i.e. how much of an alternative to consume) margins (Bhat, 2008). These multiple discrete-continuous (MDC) choice situations are pervasive, arising in transportation, marketing, health, and decisions regarding environmental resources (Bhat and Pinjari, 2014). The Kuhn-Tucker (KT) modelling framework is often employed to analyze these MDC situations and substantial progress has been made in improving these econometric modeling structures (von Haefen and Phaneuf, 2005; Bhat and Pinjari, 2014). Despite the large potential applications for KT models, there remains a gap between this potential and actual examples of these models being used. One of the reasons cited for the lack of widespread use of KT models is that estimating and simulating these models is challenging. The explanations of methods used to work with these models are spread across many papers and few user friendly software tools are available. The purpose of this paper is to present a unified account for KT estimation and simulation alongside computer code for easy and efficient implementation.

This paper presents an overview of the R package **rmdcev** which can estimate and simulate KT demand models with discrete or continuous unobserved individual heterogeneity.[1] The common starting point for all KT models is the individual's constrained optimization problem and exploiting the resulting KT first order conditions in estimation. The most popular empirical KT modelling framework is the multiple-discrete continuous extreme value (MDCEV) model as first introduced by Bhat (2008). A separate stream of literature in the environmental economics on recreation demand has developed a closely related set of models and use the term KT to describe the models. In this paper, we use KT to describe the general modelling framework, MDCEV to describe the Bhat (2008) specifications, and KT-EE to describe the environmental economics literature KT specification (von Haefen et al., 2004). One of the main differences between the MDCEV and KT-EE frameworks is how alternative-specific attributes enter the utility function, a point we describe in the paper.

Incorporating preference heterogeneity has been an important advancement in choice modeling. Both the MDCEV and KT-EE specifications can be estimated to incorporate unobserved preference heterogeneity by assuming continuous distributions using random parameters or using a latent class (LC) specification assuming a discrete distribution where people can be divided into distinct segments. The models in **rmdcev** can be fit using maximum likelihood estimation or Bayesian estimation. Besides estimation, the **rmdcev** package also implements demand forecasting and welfare calculation for policy simulation. The two main functions in the **rmdcev** are `mdcev` used to estimate all model specifications and `mdcev.sim` used to simulate both demand and welfare implications. **rmdcev** is available from the Comprehensive R Archive Network (CRAN) at `https://CRAN.R-project.org/package=rmdcev` as well as from GitHub at `https://github.com/plloydsmith/rmdcev`.

While there are several R packages available to estimate discrete choice data such as **apollo** (Hess and Palma, 2019), **mlogit** (Croissant, 2019), and **gmnl** (Sarrias and Daziano, 2017)[2], there are limited options for users interested in estimating and simulating KT models. In addition to **rmdcev**, the

[1] This paper uses version 1.2.4 of the **rmdcev** package.

[2] Sarrias and Daziano (2017) provides a good overview of the different R packages available to estimate discrete choice models

apollo package developed by Stephane Hess and David Palma at the Choice Modelling Centre in Leeds provides a flexible modelling platform for estimating MDCEV models and simulating demand behaviour (Hess and Palma, 2019). **apollo** estimates a full suite of choice models including discrete choice models and is thus more comprehensive and flexible than **rmdcev**. The main advantages for KT modeling in using the **rmdcev** is that it 1) provides functions for calculating welfare implications of policy scenarios, 2) allows the estimation and simulation of the KT formulation used in environmental economics (von Haefen and Phaneuf, 2005), 3) uses the Stan program (Carpenter et al., 2017) for Bayesian estimation and thus the user has access to specialized postestimation commands, and 4) is primarily coded in C++ and thus around 20 times faster than **apollo**. The main advantages of **apollo** compared to **rmdcev** is that 1) it can estimate model specifications without an outside good whereas **rmdcev** only estimates models with an outside good, 2) users have more control over particular parameter specifications such as which parameters are fixed at their starting values and which are allowed to be random parameters, and 3) it allows users to estimate the multiple discrete continuous nested extreme value model and LC-random parameter MDCEV specifications.

The paper first introduces the conceptual framework underlying KT models and the connection to economic theory and welfare measures. Section 2 also describes the various empirical specifications for KT models. Section 3 introduces the **rmdcev** package focusing first on estimation before moving on to discuss how to conduct welfare and demand simulations. Section 4 provides conclusions of the paper.

Models

Conceptual framework

This section describes the underlying conceptual framework for KT models. Each individual i maximizes utility through the choice of the numeraire or outside good (x_{i1}) and the non-numeraire alternatives (x_{ik}) subject to a monetary or non-monetary budget constraint. We assume there is a numeraire good (i.e. essential Hicksian composite good) which is always consumed and has a price of one. The individual's maximization problem is

$$\begin{aligned} &\max_{x_{ik}, x_{i1}} U(x_{ik}, x_{i1}) \\ &s.t. \quad y_i = \sum_{k=2}^{K} p_{ik} x_{ik} + x_{i1}, \quad x_{ik} \geq 0, \quad k = 2, ..., K, \end{aligned} \tag{1}$$

where x_{ik} is the consumption level for alternative k, x_{i1} is consumption of the numeraire, y_i is any arbitrary budget amount (e.g. annual income), and p_{ik} is the unit price of alternative k.

The resulting first-order KT conditions that implicitly define the solution to the optimal consumption bundles of x_{ik} and x_{i1} are

$$\begin{aligned} \frac{U_{x_{ik}}}{U_{x_{i1}}} &\leq p_{ik}, \quad k = 1, ...K, \\ x_{ik}\left[\frac{U_{x_{ik}}}{U_{x_{i1}}} - p_{ik}\right] &= 0, \quad k = 1, ...K. \end{aligned} \tag{2}$$

For alternatives with positive consumption levels, the marginal rate of substitution between these alternatives and the numeraire good is equal to the price of the alternative. For unconsumed alternatives, the marginal rate of substitution between these alternatives and the numeraire good is less than the price of the alternatives. For the rest of the paper, we drop the subscript i for notational simplicity.

These first-order conditions can be used to derive Marshallian and Hicksian demands and welfare measures (von Haefen and Phaneuf, 2005). We assume that alternatives have non-price attribute q_k and the vector of k prices and attributes is denoted as p and q. The Hicksian compensating surplus (CS^H) for a change in price and quality from baseline levels p^0 and q^0 to new 'policy' levels p^1 and q^1 is defined explicitly using an expenditure function

$$CS^H = y - e(p^1, q^1, \bar{U}, \theta, \varepsilon), \tag{3}$$

where θ is the vector of structural parameters ($\psi_k, \alpha_k, \gamma_k$), ε is a vector or matrix of unobserved heterogeneity, and $\bar{U} = V(p^0, q^0, y, \theta, \varepsilon)$ and represents baseline utility.

Multiple discrete-continuous extreme value model (MDCEV)

The **rmdcev** package implements the random utility specification of the MDCEV as introduced by Bhat (2008). The model specifications included in **rmdcev** always assume an outside good (i.e. the numeraire good that is always consumed by every individual). The general utility function is specified as

$$U(x_k, x_1) = \sum_{k=2}^{K} \frac{\gamma_k}{\alpha_k} \psi_k \left[\left(\frac{x_k}{\gamma_k} + 1 \right)^{\alpha_k} - 1 \right] + \frac{\psi_1}{\alpha_1} x_1^{\alpha_1}, \tag{4}$$

where $\gamma_k > 0$, $\psi_k > 0$ and $\alpha_k \leq 1$ for all k are required for this specification to be consistent with the properties of a utility function (Bhat, 2008). Bhat (2008) provides a detailed overview of the parameter interpretation and in brief

- The ψ_k parameters represent the marginal utility of consuming alternative k at the point of zero consumption (i.e. baseline marginal utility).
- The γ_k parameters are translation parameters that allow for corner solutions (i.e. zero consumption levels for alternatives) and also influence satiation. The lower the value of γ_k, the greater the satiation effect in consuming x_k.
- The α_k parameters control the rate of diminishing marginal utility of additional consumption. If α_k equal to one, then there is no satiation effects (i.e. constant marginal utility).

The 'random utility' element of the model is introduced into the baseline utility through a random error term as

$$\psi_k = \psi(z_k, \varepsilon_k) = exp(\beta' z_k + \varepsilon_k), \tag{5}$$

where z_k is a set of variables that can include alternative-specific attributes and individual-specific characteristics, and ε_k is an error term that allows for the utility function to be random over the population. We assume an extreme value distribution that is independently distributed across alternatives for ε_k with an associated scale parameter of σ. For identification, we specify $\psi_1 = e^{\varepsilon_1}$.

To ensure the estimated utility function corresponds to economic theory we specify $\gamma_k = exp(\gamma_k^*)$ such that $\gamma_k > 0$ and $\alpha_k = exp(\alpha_k^*)/(1 + exp(\alpha_k^*))$ such that $0 < \alpha_k < 1$. γ_k^* and α_k^* are estimated in the package and γ_k and α_k are reported to the user. Similarly, we specify $\sigma = exp(\sigma^*)$. Weak complementarity, which is required for deriving unique welfare measures (Mäler, 1974), is imposed in this specification by adding and subtracting one in the non-numeraire part of the utility function.

While the most general form of the MDCEV model includes ψ_k, γ_k, and α_k parameters for each alternative, Bhat (2008) discusses the identification concerns regarding estimating separate γ_k and α_k parameters for each non-numeraire alternative. Typically only a subset of these parameters can be identified and there are four common utility function specifications:

1. α-profile: set all γ_k parameters to 1.

$$U(x_k, x_1) = \sum_{k=2}^{K} \frac{1}{\alpha_k} exp(\beta' z_k + \varepsilon_k) \left[(x_k + 1)^{\alpha_k} - 1 \right] + \frac{exp(\varepsilon_1)}{\alpha_1} x_1^{\alpha_1}. \tag{6}$$

2. γ-profile: set all non-numeraire α_k parameters to 0.

$$U(x_k, x_1) = \sum_{k=2}^{K} \gamma_k exp(\beta' z_k + \varepsilon_k) \ln \left(\frac{x_k}{\gamma_k} + 1 \right) + \frac{exp(\varepsilon_1)}{\alpha_1} x_1^{\alpha_1}. \tag{7}$$

3. hybrid-profile: set all $\alpha_k = \alpha_1 = \alpha$.

$$U(x_k, x_1) = \sum_{k=2}^{K} \frac{\gamma_k}{\alpha} exp(\beta' z_k + \varepsilon_k) \left[\left(\frac{x_k}{\gamma_k} + 1 \right)^{\alpha} - 1 \right] + \frac{exp(\varepsilon_1)}{\alpha} x_1^{\alpha}. \tag{8}$$

4. hybrid0-profile: set all $\alpha_k = \alpha_1 = 0$.

$$U(x_k, x_1) = \sum_{k=2}^{K} \gamma_k exp(\beta' z_k + \varepsilon_k) \ln \left(\frac{x_k}{\gamma_k} + 1 \right) + exp(\varepsilon_1) \ln(x_1). \tag{9}$$

The likelihood function representing the model probability of the consumption pattern where M alternatives are chosen can be expressed as Bhat (2008)

$$P(x_1^*, x_2^* ... x_M^*, 0, ..., 0) = \frac{1}{\sigma^{M-1}} \left(\prod_{m=1}^{M} c_m \right) \left(\sum_{m=1}^{M} \frac{p_m}{c_m} \right) \left(\frac{\prod_{m=1}^{M} e^{V_m/\sigma}}{\left(\sum_{k=1}^{J} e^{V_k/\sigma} \right)^M} \right) (M-1)!, \tag{10}$$

where σ is the scale parameter and $c_m = \frac{1-\alpha_m}{x_m+\gamma_m}$. The V expressions depend on what model specification is used:

1. α-profile: $V_k = \beta' z_k + (\alpha_k - 1)\ln(x_k + 1) - \ln(p_k)$ for $k \geq 2$, and $V_1 = (\alpha_1 - 1)\ln(x_1)$.
2. γ-profile: $V_k = \beta' z_k - \ln\left(\frac{x_k}{\gamma_k} + 1\right) - \ln(p_k)$ for $k \geq 2$, and $V_1 = (\alpha_1 - 1)\ln(x_1)$.
3. hybrid-profile: $V_k = \beta' z_k + (\alpha - 1)\ln\left(\frac{x_k}{\gamma_k} + 1\right) - \ln(p_k)$ for $k \geq 2$, and $V_1 = (\alpha - 1)\ln(x_1)$.
4. hybrid0-profile: $V_k = \beta' z_k - \ln\left(\frac{x_k}{\gamma_k} + 1\right) - \ln(p_k)$ for $k \geq 2$, and $V_1 = -\ln(x_1)$.

Kuhn-Tucker model specifications in Environmental Economics (KT-EE)

The **rmdcev** package also implements the KT-EE specification (von Haefen and Phaneuf, 2005). The utility function in this specification is similar to the γ-profile of the MDCEV specification introduced above and is

$$U(x_k, x_1) = \sum_{k=2}^{K} \psi_k \ln(\phi_k x_k + \gamma_k) + \frac{1}{\alpha_1} x_1^{\alpha_1}, \tag{11}$$

where $\phi_k > 0$.[3]

An important difference between this KT formulation and the MDCEV models is the way weak complementary is imposed. In this KT formulation, weak complementarity is imposed by only including alternative-specific attributes in the ϕ_k parameter and not the ψ_k parameter.[4]

In this formulation, the estimating first-order conditions can be written as

$$\varepsilon_k \leq \frac{1}{\sigma}\left(-\beta' s + \ln(\frac{p_k}{\phi_k}) + \ln(\phi_k x_k + \gamma_k) + (\alpha_1 - 1)\ln(y - p_k * x_k)\right), \ \ \forall k, \tag{12}$$

and the resulting likelihood function as

$$P(x) = |J| \prod_k [exp(-g_k(.))/\sigma]^{1(x_k>0)} \, exp[-exp(-g_k(.))], \tag{13}$$

where $|J|$ is the determinant of the Jacobian of transformation, $g_k(.)$ is the right hand side of Equation (12), and $1(x_k > 0)$ is equal to one if x_k is positive and equal to zero if x_k is zero (von Haefen and Phaneuf, 2005). In previous implementations, the KT formulation used the computationally intensive numerical gradient approach to the calculation of the determinant of the Jacobian of transformation (von Haefen and Phaneuf, 2005).

The **rmdcev** package uses the compact structure of the determinant of the Jacobian as derived by Bhat (2008) and defined as

$$|J| = \frac{(1-\alpha_1)}{x_1}\left[\prod_m \frac{\phi_m}{\phi_m * x_m + \gamma_m}\right]\left[x_1(1-\alpha_1) + \sum_m \frac{(\phi_m * x_m + \gamma_m) * p_m}{\phi_m}\right], \tag{14}$$

where m denotes non-numeraire alternatives with positive consumption levels. Using this analytical gradient approach has the benefit of substantially speeding up estimation by around 70% relative to the numerical gradient approach.

In both the MDCEV and KT-EE specifications described above, the parameters $(\beta, \alpha_k, \gamma_k, \phi_k, \sigma)$ are structural parameters that are assumed to be equal across the population which simplifies estimation. However, these fixed parameter specification is quite restrictive as they can only incorporate preference heterogeneity through interaction terms with observed individual characteristics. Without these interaction terms, the fixed specifications impose the assumption that all individuals have the same tastes for alternatives (i.e. preference homogeneity). This assumption is relaxed in the next two specifications which are able to accommodate both observed and unobserved preference heterogeneity.

Latent class (LC-KT) models

The latent class version of the KT model assumes that an individual belongs to a finite mixture of S segments each indexed by s ($s = 1, 2, ...S$) (Sobhani et al., 2013; Kuriyama et al., 2010). Within each

[3]The environmental economics literature uses slightly different notation as typically θ is used for γ, μ is used for σ, and ρ for α_1. We change the notation slightly for consistency with the MDCEV model specifications.

[4]See Herriges et al. (2004) for more discussion on this point.

segment, the LC specification assumes preference homogeneity. We do not observe which segment an individual belongs to but we can attribute a probability π_{is} that individual i is a member of segment s. We impose that $0 \leq \pi_{is} \leq 1$ and $\sum_{s=1}^{S} \pi_{is} = 1$ through the use of the logit link function as

$$\pi_{is} = \frac{exp(\delta_s' w_i)}{\sum_{s=1}^{S} exp(\delta_s' w_i)}, \tag{15}$$

where w_i is a vector of individual characteristics and δ_s is a vector of coefficients to be estimated. The δ_s coefficients determine how the individual characteristics affect the membership of individual i in segment s. For identification, the δ_1 coefficients for the first segment are set to zero.

The likelihood function can be written as

$$P = \prod_i \pi_{is} P_{is}, \tag{16}$$

where P_{is} has the same form as Equations (10) and Equations (13) but is now class specific.

Random parameters (RP-LC) models

The random parameter specification of the LC models assumes that the structural parameters $\theta = (\beta, \alpha_k, \gamma_k)$ are not necessarily fixed but have an assumed distribution (Bhat, 2008). In **rmdcev**, parameters are distributed multivariate normal with a mean $\bar{\theta}$ and variance covariance matrix $\sum_\theta$ (von Haefen and Phaneuf, 2005). This structure allows for continuous preference heterogeneity and accommodates more flexible correlation patterns between alternatives in a similar fashion to the mixed logit model in discrete choice models. The σ scale parameter is always assumed to be a fixed parameter.

The most flexible model specification is to estimate the full variance covariance matrix and if there are Q parameters in θ then there are $Q(Q+1)/2$ unique variance covariance parameters to estimate in the correlated RP-MDCEV specification. An alternative is to assume the off-diagonal parameters are zero and estimate uncorrelated random parameters by estimating the Q diagonal elements of $\sum_\theta$. If all elements of $\sum_\theta$ are assumed to be zero, the model collapses to the fixed KT structures.

A note on Bayesian versus classical maximum likelihood estimation

The KT model without unobserved heterogeneity can be estimated using Bayesian or classical maximum likelihood techniques. The LC-KT model can only be estimated using classical maximum likelihood techniques as Bayesian approaches are challenged by the 'label switching' problem (Jasra et al., 2005). The RP-KT models can only be estimated using Bayesian techniques as random parameter models require simulated maximum likelihood estimators and these are not implemented in **rmdcev** at this time.

While there are philosophical differences between Bayesian and classical maximum likelihood techniques to estimating models, the Bernstein-von Mises theorem suggests that the Bayesian posterior distribution are asymptotically equivalent to maximum likelihood estimates if the data generating process has been correctly specified (Train, 2009).

The rmdcev package

Data format

The **rmdcev** uses `mdcev.data` function for handling multiple discrete-continuous data while ensuring the data is in the correct format and is suitable for estimation. The **rmdcev** package accepts data in "long" format (i.e. one row per available non-numeraire alternative for each individual). There is no row for the numeraire (i.e. outside) good. If there are I individuals and J non-numeraire alternatives, then the data frame should have IxJ rows.

To illustrate the suitable form of the data, we can load the recreation data included with the **rmdcev** package. This data is from the Canadian Nature Survey and includes choices for number of days spent recreating in 17 different outdoor activities for 2,000 people (Federal, Provincial, and Territorial Governments of Canada, 2014).

```
data(data_rec, package = "rmdcev")
```

Each recreation activity is characterized by the daily costs of participation for each individual. In addition to the recreation behaviour and prices, the data includes information on three individual characteristics: university (a dummy variable if the person has completed a university degree), ageindex (a person's age divided by the average age in sample), and urban (a dummy variable if a person lives in an urban area). Additional details on the data and price construction are provided in Lloyd-Smith (forthcoming). We can summarize the average consumption and price levels for each alternative as:

```
aggregate(cbind(quant, price) ~ alt, data = data_rec, FUN = mean )
```

```
#>                 alt   quant     price
#> 1             beach  6.5375  53.18359
#> 2           birding 14.3835  44.01734
#> 3           camping  2.5125  61.38326
#> 4           cycling  9.4700  45.99470
#> 5              fish  3.3435  86.22383
#> 6            garden 21.5710  38.28073
#> 7              golf  4.0260 134.10374
#> 8            hiking 41.4150  37.53204
#> 9        hunt_birds  0.4855 111.00176
#> 10       hunt_large  0.9480 184.46812
#> 11        hunt_trap  0.6290  95.33228
#> 12 hunt_waterfowl  0.2085 159.66605
#> 13       motor_land  3.7040 123.10169
#> 14      motor_water  2.8390 139.63845
#> 15            photo  8.6415  67.13733
#> 16        ski_cross  2.6450  32.65243
#> 17         ski_down  1.2065 151.01398
```

The data can be transformed into the structure for MDCEV estimation using the mdcev.data function:

```
data_mdcev <- mdcev.data(data_rec,
                       id.var = "id",
                       alt.var = "alt",
                       choice = "quant")
```

```
#> Sorting data by id.var then alt...
#> Checking data...
#> Data is good
```

The id.var argument indicates what variable uniquely identifies individuals in the data set, alt.var indicates the variable that identifies the non-numeraire alternatives, and choice indicates the level of consumption made by the individuals. Two other optional arguments of mdcev.data are price and income indicating the individual-specific price levels for each alternative, and the income level for each individual. These two arguments only need to be explicitly specified if they are not labeled price and income. Alternative-specific attributes and individual-specific characteristics can be included as additional columns and do not need to be specified in mdcev.data.

The mdcev.data function also checks to ensure the data has the necessary variables, and that all individuals spend positive amounts on the numeraire good. If an individual does not have positive expenditures on the numeraire good, an error message is given.

KT model estimation

A general overview of mdcev

The **rmdcev**

All the various KT model specifications are estimated using the mdcev function.

```
args(mdcev)
```

```
#> function (formula = NULL, data, weights = NULL, model = c("alpha",
#>     "gamma", "hybrid", "hybrid0", "kt_ee"), n_classes = 1, fixed_scale1 = 0,
#>     single_scale = 0, trunc_data = 0, psi_ascs = NULL, gamma_ascs = 1,
#>     seed = "123", max_iterations = 2000, jacobian_analytical_grad = 1,
#>     initial.parameters = "random", hessian = TRUE, algorithm = c("MLE",
#>         "Bayes"), flat_priors = NULL, print_iterations = TRUE,
#>     prior_psi_sd = 10, prior_gamma_sd = 10, prior_phi_sd = 10,
#>     prior_alpha_shape = 1, prior_scale_sd = 1, prior_delta_sd = 10,
#>     gamma_nonrandom = 0, alpha_nonrandom = 0, std_errors = "deltamethod",
#>     n_draws = 50, keep_loglik = 0, random_parameters = "fixed",
#>     show_stan_warnings = TRUE, n_iterations = 200, n_chains = 4,
#>     n_cores = 4, max_tree_depth = 10, adapt_delta = 0.8, lkj_shape_prior = 4,
#>     ...)
```

The main arguments are briefly explained below:

- `formula`: Formula for the model to be estimated as described in the next section.
- `data` The (IxJ) data to be used in estimation as described above.
- `weights` An optional vector of length I of sampling or frequency weights.
- `model` A string indicating which model specification to estimate. The four options are presented below:
  - "alpha": α-profile with all γ_k parameters fixed equal to 1 (Equation (6)).
  - "gamma": γ-profile with one estimated α_1 and all non-numeraire α_k parameters equal to 0 (Equation (7)).
  - "hybrid": hybrid-profile with a single estimated α parameter (i.e. $\alpha_1 = \alpha_k = \alpha$) (Equation (8)).
  - "hybrid0": hybrid-profile with all α parameters fixed equal to 1e-3 (Equation (8)).
  - "kt_ee": Environmental economics version of KT model (Equation (11)).
- `n_classes` The number of latent classes. Note that the LC model is automatically estimated as long as the prespecified number of classes is set greater than 1.
- `gamma_ascs` Indicator to include alternative-specific gammas parameters.
- `psi_ascs` Whether to include alternative-specific psi parameters. The first alternative is used as the reference category. Only specify to 1 for MDCEV models.
- `fixed_scale1` Whether to fix the scale parameter at 1.
- `trunc_data` Whether the estimation should be adjusted for truncation of non-numeraire alternatives. This option is useful if the data only includes individuals with positive non-numeraire consumption levels such as recreation data collected on-site. To account for the truncation of consumption, the likelihood is normalized by one minus the likelihood of observing zero consumption for all non-numeraire alternatives (i.e. likelihood of positive consumption) following Englin, Boxall and Watson (1998) and von Haefen (2003).
- `seed` Random seed.
- `algorithm` Either "Bayes" for Bayesian estimation or "MLE" for maximum likelihood estimation. The MLE algorithm uses the Limited-memory BFGS which approximates the Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm but uses less computer memory.
- `flat_priors` indicator if completely uninformative priors should be specified. Defaults to 1 if MLE used and 0 if Bayes used. If using MLE and set flat_priors = 0, penalized MLE is used and the optimizing objective is augmented with the priors.
- `print_iterations` Whether to print intermediate iteration information or not.
- `std_errors` Compute standard errors using the delta method ("deltamethod") or multivariate normal draws ("mvn"). The default is "deltamethod". Note that mvn parameter draws should be used to incorporate parameter uncertainty for demand and welfare simulation. For maximum likelihood estimation only.
- `n_draws` The number of multivariate normal draws for standard error calculations if "mvn" is specified.
- `initial.parameters` The default for fixed and random parameter specifications is to use random starting values (except for the scale parameter with a starting value set to 1). For LC models, the default is to use slightly adjusted MLE point estimates from the single class model. Initial parameter values should be included in a named list. For example, the LC "hybrid" specification initial parameters can be specified as:

  ```
  initial.parameters = list(psi = array(0, dim = c(K, num_psi)),
                            gamma = array(1, dim = c(K, num_alt)),
                            alpha = array(0.5, dim = c(K, 1)),
                            scale = array(1, dim = c(K)))
  ```

  where K is the number of classes (i.e. K = 1 is used for single class models), num_psi is number of psi parameters, and num_alt is number of non-numeraire alternatives.

Formula format

The formula is used to incorporate alternative-specific variables and individual-specific characteristics into the ψ_k parameters, the membership equation of the LC-KT models, and ϕ_k parameters for the KT-EE specification. By default, alternative-specific constants (ASCs) for all non-numeraire alternatives are included in the ψ_k and γ_k parameters. For the ψ_k, the first ASC is fixed at 0 due to identification concerns. They can be omitted using the `psi_ascs = 0` and `gamma_ascs = 0` arguments. Furthermore, the γ_k, α_k, and σ parameters cannot include alternative- or individual specific variables besides ASCs.

The formula is divided in three parts, separated by the symbol `|` and is based on the R package **Formula** (Zeileis and Croissant, 2010). The first part is reserved for the z_k variables in ψ_k as in Equation (5), excluding ASCs. These can include alternative-specific and individual-specific variables. Interaction terms between variables can be included using the normal **Formula** syntax of `z1:z2`. This is particularly useful for creating interaction terms to incorporate observed preference heterogeneity for alternative-specific variables and individual-specific characteristics.

For a model with only ASCs in ψ_k, the formula can be specified as

```
f1 = ~ 0
```

We can add individual-specific variables to the ψ_k parameters as follows

```
f2 = ~ university + ageindex
```

Alternative-specific variables such as `z1` and `z2` can be included in the same way such as

```
f2 = ~ z1 + z2
```

The second part corresponds to individual-specific characteristics that enter in the probability assignment in models with latent classes. The formula will automatically include a constant in the membership equation but this can be omitted if `-1` is used in the formula. For example, a LC model with no alternative-specific variables in the psi_k parameters and `university`, `ageindex` and a constant determine the class membership can be specified as

```
f3 = ~ 0 | university + ageindex
```

The third part is reserved for the q_k variables included in the ϕ_k parameters in the KT-EE model specification ((Equation 11)). For example, if there was an alternative-specific variable named 'q1', it can be included as below

```
f4 = ~ 0 | 0 | q1
```

Estimating KT models using maximum likelihood techniques

We estimate a KT model by first calling `mdcev.data` on the **Recreation** data. For these examples we are going to use a subset of 200 individuals from the data.

```
data_model <- mdcev.data(data_rec, subset = id <= 200,
                       id.var = "id",
                       alt.var = "alt",
                       choice = "quant")

#> Sorting data by id.var then alt...
#> Checking data...
#> Data is good
```

We might think that older people prefer gardening to other activities and so we can include an interaction term between the activity `garden` and the variable `ageindex`. There are no alternative-specific variables besides constant terms to include in ψ and therefore the formula can be specified as

```
data_model$age_garden = ifelse(data_model$alt == "garden",
                               data_model$ageindex,0)
f5 = ~ age_garden
```

We specify the γ-profile of the MDCEV model specification where a single α_1 is estimated for the numeraire alternative and all non-numeraire alternatives are fixed at zero by setting `model = "gamma"`. We use maximum likelihood estimation by setting `algorithm = "MLE"`.

The syntax for the model is the following:

```
mdcev_mle <- mdcev(~ age_garden,
                  data = data_model,
                  model = "gamma",
                  algorithm = "MLE",
                  print_iterations = FALSE)

#> Using MLE to estimate KT model
```

Setting `print_iterations = TRUE` will print out intermediate iteration results as the model converges.

The output of the function can be accessed by calling summary.

```
summary(mdcev_mle)

#> Model run using rmdcev for R, version 1.2.4
#> Estimation method                : MLE
#> Model type                       : gamma specification
#> Number of classes                : 1
#> Number of individuals            : 200
#> Number of non-numeraire alts     : 17
#> Estimated parameters             : 36
#> LL                               : -5119.11
#> AIC                              : 10310.21
#> BIC                              : 10428.95
#> Standard errors calculated using : Delta method
#> Exit of MLE                      : successful convergence
#> Time taken (hh:mm:ss)            : 00:00:0.5
#>
#> Average consumption of non-numeraire alternatives:
#>          beach        birding        camping        cycling           fish
#>           6.70          12.75           2.60           7.89           4.00
#>         garden           golf         hiking     hunt_birds     hunt_large
#>          23.18           5.42          41.62           0.58           1.03
#>      hunt_trap hunt_waterfowl     motor_land    motor_water          photo
#>           0.80           0.24           5.92           3.53          11.00
#>      ski_cross       ski_down
#>           3.12           1.85
#>
#> Parameter estimates --------------------------------
#>                      Estimate Std.err z.stat
#> psi_birding            -0.762   0.113  -6.75
#> psi_camping            -0.534   0.115  -4.64
#> psi_cycling            -0.455   0.110  -4.13
#> psi_fish               -0.162   0.116  -1.39
#> psi_garden             -0.537   0.176  -3.05
#> psi_golf                0.553   0.112   4.94
#> psi_hiking             -0.039   0.107  -0.36
#> psi_hunt_birds         -1.034   0.194  -5.33
#> psi_hunt_large         -0.234   0.160  -1.46
#> psi_hunt_trap          -1.280   0.208  -6.16
#> psi_hunt_waterfowl     -0.886   0.254  -3.49
#> psi_motor_land          0.119   0.126   0.94
#> psi_motor_water         0.458   0.115   3.98
#> psi_photo               0.011   0.105   0.11
#> psi_ski_cross          -1.164   0.122  -9.54
#> psi_ski_down            0.229   0.134   1.71
#> psi_age_garden          0.513   0.155   3.31
#> gamma_beach             8.662   1.457   5.95
#> gamma_birding          22.366   4.945   4.52
#> gamma_camping           7.546   1.482   5.09
#> gamma_cycling          16.182   3.115   5.19
#> gamma_fish             11.831   2.277   5.20
#> gamma_garden           17.763   2.711   6.55
#> gamma_golf             11.082   2.393   4.63
#> gamma_hiking           17.467   2.872   6.08
#> gamma_hunt_birds        9.669   3.688   2.62
#> gamma_hunt_large       12.561   3.589   3.50
#> gamma_hunt_trap        12.714   5.656   2.25
#> gamma_hunt_waterfowl    7.739   4.167   1.86
#> gamma_motor_land       16.277   4.009   4.06
#> gamma_motor_water      11.247   2.352   4.78
#> gamma_photo            14.478   2.635   5.49
#> gamma_ski_cross        10.365   2.387   4.34
#> gamma_ski_down          9.051   2.403   3.77
#> alpha_num               0.667   0.008  83.43
#> scale                   0.607   0.027  22.47
#> Note: All non-numeraire alpha's fixed to 0.
```

The summary includes overall model and estimation information and the parameter estimates. All parameters have been transformed to their original form.[5] Interpreting the parameter estimates of KT models directly is challenging due to the non-linearities implied by the utility function and the partial confounding of α_k and γ_k parameters (see Bhat (2008) for a in-depth discussion). Examining the ψ_k parameters first which represent the marginal utility when consumption is zero, we can see that relative to the beach recreation activity (i.e. the omitted reference category), hunting and trapping and cross country skiing have the largest negative ASCs suggesting these activities are less preferred starting from zero consumption levels. The interaction parameter between age and gardening is positive and significant suggesting that older people gain a higher utility from gardening compared to younger people. Because all non-numeraire α parameters are fixed at zero, the γ_k parameters can be interpreted as capturing satiation and these satiation effects are lowest for the activities with the highest γ_k parameter values such as birding, cycling, and motorized land vehicles. The α_1 is estimated to be less than 1 which also implies satiation in the numeraire good. Bhat (2008); Lloyd-Smith et al. (2019) provide empirical applications of this model.

In the next example, we estimate the α-profile of the MDCEV utility function by changing the model argument to "alpha".

```
mdcev_mle <- mdcev(~ age_garden,
                   data = data_model,
                   model = "alpha",
                   algorithm = "MLE",
                   print_iterations = FALSE)


summary(mdcev_mle)
#> Model run using rmdcev for R, version 1.2.4
#> Estimation method                 : MLE
#> Model type                        : alpha specification
#> Number of classes                 : 1
#> Number of individuals             : 200
#> Number of non-numeraire alts      : 17
#> Estimated parameters              : 36
#> LL                                : -5354.33
#> AIC                               : 10780.67
#> BIC                               : 10899.41
#> Standard errors calculated using : Delta method
#> Exit of MLE                       : successful convergence
#> Time taken (hh:mm:ss)             : 00:00:0.59
#>
#> Average consumption of non-numeraire alternatives:
#>         beach        birding        camping        cycling           fish
#>          6.70          12.75           2.60           7.89           4.00
#>        garden           golf         hiking     hunt_birds     hunt_large
#>         23.18           5.42          41.62           0.58           1.03
#>     hunt_trap hunt_waterfowl     motor_land    motor_water          photo
#>          0.80           0.24           5.92           3.53          11.00
#>     ski_cross       ski_down
#>          3.12           1.85
#>
```

[5] $\gamma_k = exp(\gamma_k^*)$, $\alpha_1 = exp(\alpha_1^*)/(1 + exp(\alpha_1^*))$, and $\sigma = exp(\sigma^*)$, where γ_k^*, α_1^*, and σ^* are estimated but the transformed parameters are returned to users.

```
#> Parameter estimates --------------------------------
#>                      Estimate Std.err z.stat
#> psi_birding            -0.821   0.115  -7.13
#> psi_camping            -0.582   0.117  -4.97
#> psi_cycling            -0.501   0.111  -4.51
#> psi_fish               -0.208   0.117  -1.78
#> psi_garden             -0.481   0.176  -2.73
#> psi_golf                0.492   0.114   4.32
#> psi_hiking              0.127   0.109   1.17
#> psi_hunt_birds         -1.121   0.199  -5.64
#> psi_hunt_large         -0.309   0.164  -1.88
#> psi_hunt_trap          -1.359   0.213  -6.38
#> psi_hunt_waterfowl     -0.976   0.261  -3.74
#> psi_motor_land          0.040   0.129   0.31
#> psi_motor_water         0.396   0.117   3.38
#> psi_photo              -0.031   0.105  -0.29
#> psi_ski_cross          -1.229   0.125  -9.83
#> psi_ski_down            0.158   0.138   1.14
#> psi_age_garden          0.494   0.156   3.17
#> alpha_num               0.658   0.008  82.21
#> alpha_beach             0.593   0.040  14.82
#> alpha_birding           0.720   0.038  18.94
#> alpha_camping           0.596   0.049  12.16
#> alpha_cycling           0.700   0.039  17.94
#> alpha_fish              0.660   0.043  15.34
#> alpha_garden            0.647   0.030  21.55
#> alpha_golf              0.669   0.045  14.87
#> alpha_hiking            0.595   0.030  19.82
#> alpha_hunt_birds        0.665   0.090   7.39
#> alpha_hunt_large        0.701   0.068  10.31
#> alpha_hunt_trap         0.710   0.094   7.55
#> alpha_hunt_waterfowl    0.651   0.132   4.93
#> alpha_motor_land        0.721   0.048  15.02
#> alpha_motor_water       0.663   0.047  14.12
#> alpha_photo             0.680   0.037  18.37
#> alpha_ski_cross         0.661   0.051  12.97
#> alpha_ski_down          0.658   0.060  10.97
#> scale                   0.602   0.034  17.71
#> Note: All non-numeraire gamma's fixed to 1.
```

Estimating alternative-specific α_k parameters and fixing all the non-numeraire γ parameters at 1, allows us to see the heterogeneity in α_k parameters across recreation activities.

The hybrid model specification of the MDCEV model where a single α is estimated for the numeraire and non-numeraire alternatives can be estimated by setting `model = "hybrid"` as the next example demonstrates.

```
mdcev_mle <- mdcev(~ age_garden,
                  data = data_model,
                  model = "hybrid",
                  algorithm = "MLE",
                  print_iterations = FALSE)

#> Using MLE to estimate KT model

summary(mdcev_mle)

#> Model run using rmdcev for R, version 1.2.4
#> Estimation method                : MLE
#> Model type                       : hybrid specification
#> Number of classes                : 1
#> Number of individuals            : 200
#> Number of non-numeraire alts     : 17
#> Estimated parameters             : 36
#> LL                               : -5230.91
#> AIC                              : 10533.81
#> BIC                              : 10652.55
#> Standard errors calculated using : Delta method
#> Exit of MLE                      : successful convergence
#> Time taken (hh:mm:ss)            : 00:00:0.6
#>
#> Average consumption of non-numeraire alternatives:
#>          beach        birding        camping        cycling           fish
#>           6.70          12.75           2.60           7.89           4.00
#>         garden           golf         hiking     hunt_birds     hunt_large
#>          23.18           5.42          41.62           0.58           1.03
#>      hunt_trap hunt_waterfowl     motor_land    motor_water          photo
#>           0.80           0.24           5.92           3.53          11.00
#>      ski_cross       ski_down
#>           3.12           1.85
#>
#> Parameter estimates --------------------------------
#>                      Estimate Std.err z.stat
#> psi_birding            -0.783   0.081  -9.67
#> psi_camping            -0.570   0.082  -6.95
#> psi_cycling            -0.488   0.078  -6.25
#> psi_fish               -0.206   0.083  -2.48
#> psi_garden             -0.580   0.128  -4.53
#> psi_golf                0.565   0.080   7.06
#> psi_hiking             -0.285   0.076  -3.75
#> psi_hunt_birds         -0.832   0.137  -6.08
#> psi_hunt_large         -0.095   0.113  -0.84
#> psi_hunt_trap          -1.029   0.146  -7.05
#> psi_hunt_waterfowl     -0.524   0.178  -2.94
#> psi_motor_land          0.172   0.090   1.91
#> psi_motor_water         0.449   0.082   5.48
#> psi_photo              -0.103   0.074  -1.39
#> psi_ski_cross          -1.112   0.087 -12.78
#> psi_ski_down            0.345   0.095   3.63
#> psi_age_garden          0.312   0.112   2.79
#> gamma_beach             2.198   0.446   4.93
#> gamma_birding           5.722   1.484   3.86
#> gamma_camping           2.669   0.649   4.11
#> gamma_cycling           5.745   1.307   4.40
#> gamma_fish              4.162   1.007   4.13
#> gamma_garden            4.776   0.910   5.25
#> gamma_golf              3.446   0.873   3.95
#> gamma_hiking            3.315   0.719   4.61
#> gamma_hunt_birds        3.719   1.704   2.18
#> gamma_hunt_large        5.533   1.922   2.88
#> gamma_hunt_trap         4.605   2.446   1.88
#> gamma_hunt_waterfowl    3.227   2.029   1.59
#> gamma_motor_land        5.691   1.642   3.47
#> gamma_motor_water       3.941   1.011   3.90
#> gamma_photo             4.723   1.012   4.67
#> gamma_ski_cross         3.593   0.994   3.61
#> gamma_ski_down          3.265   1.027   3.18
#> alpha                   0.648   0.005 129.53
#> scale                   0.431   0.014  30.78
#> Note: Alpha parameter is equal for all alternatives.
```

The same number of parameters are estimated in all three models and the log-likelihood is highest for the γ-profile specification. The ease of estimating different MDCEV model specifications can be used to compare models quickly and help the analyst pick their preferred specification for each empirical application.

We can also estimate the KT-EE specification by changing the formula call and the model call to "kt_ee".

```
kt_mle <- mdcev(~ age_garden | 0 | 0,
                   data = data_model,
                   model = "kt_ee",
                  algorithm = "MLE",
                  print_iterations = FALSE)

summary(kt_mle)

#> Model run using rmdcev for R, version 1.2.4
#> Estimation method                : MLE
#> Model type                       : kt_ee specification
#> Number of classes                : 1
#> Number of individuals            : 200
#> Number of non-numeraire alts     : 17
#> Estimated parameters             : 20
#> LL                               : -5360.46
#> AIC                              : 10760.93
#> BIC                              : 10826.89
#> Standard errors calculated using : Delta method
#> Exit of MLE                      : successful convergence
#> Time taken (hh:mm:ss)            : 00:00:0.27
#>
#> Average consumption of non-numeraire alternatives:
#>          beach        birding        camping        cycling           fish
#>           6.70          12.75           2.60           7.89           4.00
#>         garden           golf         hiking     hunt_birds     hunt_large
#>          23.18           5.42          41.62           0.58           1.03
#>      hunt_trap hunt_waterfowl     motor_land    motor_water          photo
#>           0.80           0.24           5.92           3.53          11.00
#>      ski_cross       ski_down
#>           3.12           1.85
#>
#> Parameter estimates --------------------------------
#>                       Estimate Std.err z.stat
#> psi_age_garden           0.395   0.110   3.59
#> gamma_beach             10.552   1.083   9.74
#> gamma_birding           22.278   2.485   8.97
#> gamma_camping           16.210   1.778   9.12
#> gamma_cycling           16.247   1.744   9.32
#> gamma_fish              12.245   1.360   9.00
#> gamma_garden            16.651   2.167   7.68
#> gamma_golf               6.241   0.700   8.92
#> gamma_hiking            11.918   1.322   9.02
#> gamma_hunt_birds        25.826   4.427   5.83
#> gamma_hunt_large        13.803   2.020   6.83
#> gamma_hunt_trap         32.843   6.100   5.38
#> gamma_hunt_waterfowl    24.635   5.550   4.44
#> gamma_motor_land        10.405   1.282   8.12
#> gamma_motor_water        7.117   0.812   8.76
#> gamma_photo             11.160   1.184   9.43
#> gamma_ski_cross         28.693   3.201   8.96
#> gamma_ski_down           8.405   1.065   7.89
#> alpha_num                0.475   0.007  67.92
#> scale                    0.713   0.025  28.53
```

This model does not include ASCs in the psi_k parameters due to concerns about weak complementarity.

Estimating KT models using Bayesian techniques

The exact same models can be fit using Bayesian estimation by changing the algorithm call to "Bayes". Bayesian estimation is implemented using the Stan programming language (Carpenter et al., 2017). The Bayesian framework requires careful choice of priors for the parameters, especially in data sparse contexts. The specific prior distributions for the fixed parameter specifications is presented below. The user has the ability to change the standard deviation and shape of these priors through these options in the mdcev function:

- `prior_psi_sd` standard deviation for normal prior with mean 0.

- `prior_phi_sd` standard deviation for normal prior with mean 0.
- `prior_gamma_sd` standard deviation for half-normal prior with mean 1.
- `prior_alpha_shape` shape parameter for beta distribution.
- `prior_scale_sd` standard deviation for half-normal prior with mean 0.

For the random parameter model specifications, the priors for the means of all random parameters follow a normal distribution with mean 0 on the unconstrained space.

There are also a number of further options for Bayesian estimation. For example, the number of iterations (n_iterations), number of chains (n_chains), and number of cores (n_cores) for parallel implementation of the chains can also be chosen. The full set of options for Bayesian estimation are presented below.

- `random_parameters` The form of the covariance matrix for the parameters. Options are
  - 'fixed' for no random parameters,
  - 'uncorr for uncorrelated random parameters, or
  - 'corr' for correlated random parameters.
- `n_iterations` The number of iterations to use in Bayesian estimation. The default is for the number of iterations to be split evenly between warmup and posterior draws. The number of warmup draws can be directly controlled using the warmup argument (see rstan::sampling)
- `n_chains` The number of independent Markov chains in Bayesian estimation.
- `n_cores` The number of cores used to execute the Markov chains in parallel in Bayesian estimation. Can set using options(mc.cores = parallel::detectCores()).
- `lkj_shape_prior` Prior for Cholesky matrix for correlated random parameters.

In this example, we estimate the γ-profile of the MDCEV specification using Bayesian techniques. We set the number of iterations to 200 and use 4 independent chains across 4 cores.

```
mdcev_bayes <- mdcev(~ age_garden,
                        data = data_model,
                        model = "gamma",
                        algorithm = "Bayes",
                        n_iterations = 200,
                        n_chains = 4,
                        n_cores = 4,
                        print_iterations = FALSE)
```

The output of the function can be accessed by calling `summary`.

```
summary(mdcev_bayes)

#> Model run using rmdcev for R, version 1.2.4
#> Estimation method                : Bayes
#> Model type                       : gamma specification
#> Number of classes                : 1
#> Number of individuals            : 200
#> Number of non-numeraire alts     : 17
#> Estimated parameters             : 36
#> LL                               : -5137.78
#> Number of chains                 : 4
#> Number of warmup draws per chain : 100
#> Total post-warmup sample         : 400
#> Time taken (hh:mm:ss)            : 00:00:41
#>
#> Average consumption of non-numeraire alternatives:
#>          beach        birding        camping        cycling           fish
#>           6.70          12.75           2.60           7.89           4.00
#>         garden           golf         hiking     hunt_birds     hunt_large
#>          23.18           5.42          41.62           0.58           1.03
#>      hunt_trap hunt_waterfowl     motor_land    motor_water          photo
#>           0.80           0.24           5.92           3.53          11.00
#>      ski_cross       ski_down
#>           3.12           1.85
#>
```

```
#> Parameter estimates --------------------------------
#>                      Estimate Std.dev z.stat n_eff Rhat
#> psi_birding            -0.789   0.113  -7.00   255 0.99
#> psi_camping            -0.574   0.123  -4.67   258 1.01
#> psi_cycling            -0.489   0.118  -4.16   276 1.00
#> psi_fish               -0.206   0.123  -1.68   157 1.01
#> psi_garden             -0.562   0.196  -2.86   347 1.01
#> psi_golf                0.501   0.134   3.74   201 1.01
#> psi_hiking              0.025   0.117   0.21   429 1.00
#> psi_hunt_birds         -1.159   0.200  -5.80   235 1.01
#> psi_hunt_large         -0.344   0.179  -1.91   189 1.02
#> psi_hunt_trap          -1.436   0.245  -5.87   271 1.00
#> psi_hunt_waterfowl     -1.098   0.291  -3.77   187 1.00
#> psi_motor_land          0.060   0.136   0.44   301 1.00
#> psi_motor_water         0.420   0.125   3.37   277 1.00
#> psi_photo              -0.010   0.125  -0.08   312 1.00
#> psi_ski_cross          -1.222   0.135  -9.06   207 1.01
#> psi_ski_down            0.153   0.147   1.04   362 1.00
#> psi_age_garden          0.563   0.175   3.22   481 1.00
#> gamma_beach             8.013   1.362   5.88   261 1.01
#> gamma_birding          17.715   3.298   5.37   516 1.00
#> gamma_camping           7.128   1.347   5.29   459 1.00
#> gamma_cycling          14.506   2.756   5.26   614 1.00
#> gamma_fish             11.002   2.110   5.21   636 1.00
#> gamma_garden           15.587   2.546   6.12   685 0.99
#> gamma_golf             10.145   2.029   5.00   399 1.00
#> gamma_hiking           15.158   2.444   6.20   687 1.00
#> gamma_hunt_birds        9.479   3.372   2.81   254 1.00
#> gamma_hunt_large       11.702   3.016   3.88   320 1.01
#> gamma_hunt_trap        11.582   3.582   3.23   252 1.01
#> gamma_hunt_waterfowl    8.269   3.551   2.33   141 1.01
#> gamma_motor_land       14.258   3.252   4.39   511 1.00
#> gamma_motor_water      10.392   2.260   4.60   485 1.00
#> gamma_photo            13.013   2.398   5.43   595 0.99
#> gamma_ski_cross         9.652   2.310   4.18   527 1.00
#> gamma_ski_down          8.814   2.344   3.76   357 1.00
#> alpha_num               0.668   0.008  82.15   296 1.00
#> scale                   0.654   0.029  22.75   187 1.01
#> Note: All non-numeraire alpha's fixed to 0.
#> Note from Rstan: 'For each parameter, n_eff is a crude measure of effective sample
#> size, and Rhat is the potential scale reduction factor on split chains (at
#> convergence, Rhat=1)'
```

Comparing these parameter values to the maximum likelihood estimates of the γ-profile MDCEV specification, the values are quite similar. As the data set is rather small with only 200 individuals, the priors play a role in reducing the estimates closer to 1 for the γ_k, but this role will lessen in larger data applications.

One benefit of using the Bayesian approach is that one can take advantage of the postestimation commands, interactive diagnostics, and posterior analysis in **rstan**, **bayesplot** (Gabry et al., 2019), and **shinystan** (Muth et al., 2018). For example, the effective sample size reports the estimated number of independent draws from the posterior distribution for each parameter (Stan Development Team, 2019). The interested reader is referred to these packages for additional details.

Estimating LC-KT models

In this example, we estimate a LC-KT model using the **Recreation** data. We set the number of classes equal to 2 and we use data on 1000 individuals. We would like to include the `university`, `ageindex`, and `urban` in the membership equation and we include them in the `formula` interface. The constant for the membership equation is included automatically. The LC model is automatically estimated as long as the prespecified number of classes (`n_classes`) is set greater than 1. The scale parameters are fixed at 1 using `fixed_scale1 = 1`.

```
data_model <- mdcev.data(data_rec, subset = id <= 1000,
                       id.var = "id",
                       alt.var = "alt",
                       choice = "quant")

mdcev_lc <- mdcev(~ 0 | university + ageindex + urban,
                  data = data_model,
                  n_classes = 2,
                  model = "gamma",
                  fixed_scale1 = 1,
                  algorithm = "MLE",
                  print_iterations = FALSE)


summary(mdcev_lc)
#> Model run using rmdcev for R, version 1.2.4
#> Estimation method                : MLE
#> Model type                       : gamma specification
#> Number of classes                : 2
#> Number of individuals            : 1000
#> Number of non-numeraire alts     : 17
#> Estimated parameters             : 72
#> LL                               : -23298.65
#> AIC                              : 46741.3
#> BIC                              : 47094.66
#> Standard errors calculated using : Delta method
#> Exit of MLE                      : successful convergence
#> Time taken (hh:mm:ss)            : 00:00:11.39
#>
#> Average consumption of non-numeraire alternatives:
#>        beach       birding        camping        cycling
#> 6.44         14.34          2.31           8.06
#> fish         garden         golf         hiking
#> 3.15         21.61          4.45          40.03
#> hunt_birds     hunt_large      hunt_trap hunt_waterfowl
#> 0.49           1.01          0.59           0.20
#> motor_land    motor_water      photo      ski_cross
#> 4.03           2.96          9.00           2.48
#> ski_down
#> 1.18
#>
#>
#> Class average probabilities:
#>        class1 class2
#> 0.86   0.14
#> Parameter estimates --------------------------------
#>        Estimate Std.err z.stat
#> class1.psi_birding           -1.268   0.095 -13.35
#> class2.psi_birding           -1.178   0.095 -12.40
#> class1.psi_camping           -0.948   0.089 -10.65
#> class2.psi_camping           -0.646   0.117  -5.52
#> class1.psi_cycling           -0.754   0.080  -9.42
#> class2.psi_cycling           -1.094   0.099 -11.05
#> class1.psi_fish              -1.075   0.085 -12.64
#> class2.psi_fish               1.179   0.546   2.16
#> class1.psi_garden             0.032   0.656   0.05
#> class2.psi_garden            -0.200   1.491  -0.13
#> class1.psi_golf              -0.122   0.549  -0.22
#> class2.psi_golf               0.406   0.118   3.44
#> class1.psi_hiking             0.444   0.111   4.00
#> class2.psi_hiking             0.201   0.086   2.34
#> class1.psi_hunt_birds        -4.348   0.103 -42.21
#> class2.psi_hunt_birds         0.298   0.111   2.69
#> class1.psi_hunt_large        -3.783   0.249 -15.19
#> class2.psi_hunt_large         1.295   0.235   5.51
#> class1.psi_hunt_trap         -6.475   0.253 -25.59
#> class2.psi_hunt_trap         -0.570   0.224  -2.55
#> class1.psi_hunt_waterfowl    -4.140   0.217 -19.08
#> class2.psi_hunt_waterfowl    -0.328   0.234  -1.40
#> class1.psi_motor_land        -0.776   0.212  -3.66
#> class2.psi_motor_land         1.147   0.229   5.01
#> class1.psi_motor_water       -0.397   0.233  -1.70
#> class2.psi_motor_water        1.399   0.246   5.69
#> class1.psi_photo             -0.207   0.266  -0.78
#> class2.psi_photo             -0.653   0.221  -2.95
#> class1.psi_ski_cross         -1.772   0.209  -8.48
#> class2.psi_ski_cross         -1.288   0.247  -5.21
#> class1.psi_ski_down          -0.473   0.245  -1.93
#> class2.psi_ski_down          -0.388   0.282  -1.38
#> class1.gamma_beach            4.112   0.372  11.05
#> class2.gamma_beach            6.337   0.850   7.45
#> class1.gamma_birding         15.129   1.727   8.76
#> class2.gamma_birding          7.732   0.833   9.28
#> class1.gamma_camping          3.497   0.520   6.73
#> class2.gamma_camping          7.827   0.650  12.04
#> class1.gamma_cycling          9.862   1.299   7.59
#> class2.gamma_cycling         13.344   1.185  11.26
#> class1.gamma_fish             4.854   3.539   1.37
#> class2.gamma_fish             3.496   2.701   1.29
#> class1.gamma_garden           9.858  16.725   0.59
#> class2.gamma_garden           8.924   7.287   1.22
#> class1.gamma_golf             7.178   1.151   6.24
#> class2.gamma_golf             4.562   0.662   6.89
#> class1.gamma_hiking           7.107   0.705  10.08
#> class2.gamma_hiking          10.823   1.472   7.35
#> class1.gamma_hunt_birds       2.989   0.445   6.72
#> class2.gamma_hunt_birds       2.673   0.639   4.18
#> class1.gamma_hunt_large       4.752   1.764   2.69
#> class2.gamma_hunt_large       3.361   0.924   3.64
#> class1.gamma_hunt_trap        0.975   0.305   3.20
#> class2.gamma_hunt_trap        5.343   1.146   4.66
#> class1.gamma_hunt_waterfowl   3.842   0.910   4.22
#> class2.gamma_hunt_waterfowl   3.626   1.017   3.57
#> class1.gamma_motor_land       5.807   1.331   4.36
#> class2.gamma_motor_land       7.884   1.757   4.49
#> class1.gamma_motor_water      3.894   0.817   4.77
#> class2.gamma_motor_water      5.414   1.523   3.55
#> class1.gamma_photo            6.970   2.271   3.07
#> class2.gamma_photo            7.877   1.674   4.71
#> class1.gamma_ski_cross        4.951   1.039   4.77
#> class2.gamma_ski_cross        4.932   1.480   3.33
#> class1.gamma_ski_down         3.887   1.107   3.51
#> class2.gamma_ski_down         4.667   1.677   2.78
#> class1.alpha_num              0.679   0.006 113.18
#> class2.alpha_num              0.676   0.017  39.78
#> class2.(Intercept)           -1.187   0.366  -3.24
#> class2.university            -0.506   0.257  -1.97
#> class2.ageindex               0.129   0.281   0.46
#> class2.urban                 -0.752   0.260  -2.89
#> Note: Scale parameter fixed to 1.
#> Note: All non-numeraire alpha's fixed to 0.
#> Note: The membership equation parameters for class 1 are normalized to 0.
```

In this LC example, we assume that there are two types of people that have different preferences for recreation. The probability of class assignment depends on unobserved factors and the three sociodemographic factors included in the membership equation with only urban having a statistically significant effect on class probability. People living in urban areas are less likely to be in class 2. The summary output reports the average class probabilities as being 85% for class 1 and 15% for class 2. The ψ parameters across classes are similar although there are some noticeable differences such as the hunting and trapping preferences. The γ parameters, on the other hand, show that satiation between classes is quite different. Sobhani et al. (2013); Kuriyama et al. (2010) provide empirical applications of these models.

If `initial.parameter` are not provided, the default is to use slightly adjusted parameter estimates of the MDCEV model as starting values when estimating the LC-MDCEV model to assist speed and convergence issues.[6] The MDCEV model output can be accessed from `mdcev_lc[["mdcev_fit"]]` object for comparison.

Estimating RP-KT models

Random parameter models require defining and parameterizing the variance covariance matrix. For uncorrelated random parameters, the diagonal elements of the variance covariance matrix are estimated and the off-diagonal elements are assumed to be zero. For correlated random parameters, the variance covariance matrix is fully estimated and can be parameterized in many ways. The **rmdcev** package defines the variance covariance matrix in terms of Cholesky factors of the correlation matrix and a vector of standard deviations for numerical stability. Thus the variance covariance matrix is specified as

$$\sum = diag(\tau)\ x\ LL^T\ x\ diag(\tau), \tag{17}$$

where τ is a vector of standard deviations, and L is the cholesky factors of the correlation matrix.

In this example, we estimate an uncorrelated random parameters γ-specification of the MDCEV model without any ψ_k parameters. We set the argument `random_parameters = "uncorr"` to indicate that uncorrelated random parameters will be estimated. As noted earlier, all random parameters follow a normal distribution. We change the `psi_ascs = 0` to omit the ASCs in the ψ_k parameters.

```
data_model <- mdcev.data(data_rec, subset = id <= 200,
                       id.var = "id",
                       alt.var = "alt",
                       choice = "quant")

mdcev_rp <- mdcev(~ 0,
                    data = data_model,
                    model = "gamma",
                    algorithm = "Bayes",
                    n_chains = 4,
                    psi_ascs = 0,
                    fixed_scale1 = 1,
                    n_iterations = 200,
                    random_parameters = "uncorr",
                    print_iterations = FALSE)

summary(mdcev_rp)

#> Model run using rmdcev for R, version 1.2.4
#> Estimation method                 : Bayes
#> Model type                        : gamma specification
#> Number of classes                 : 1
#> Number of individuals             : 200
#> Number of non-numeraire alts      : 17
#> Estimated parameters              : 36
#> LL                                : -5363.25
#> Random parameters                 : uncorrelated random parameters
#> Number of chains                  : 4
#> Number of warmup draws per chain : 100
#> Total post-warmup sample          : 400
#> Time taken (hh:mm:ss)             : 00:01:51.2
#>
#> Average consumption of non-numeraire alternatives:
#>          beach       birding       camping       cycling          fish
#>           6.70         12.75          2.60          7.89          4.00
#>         garden          golf        hiking    hunt_birds    hunt_large
#>          23.18          5.42         41.62          0.58          1.03
```

[6]In particular, the estimated ψ_k and γ_k parameters from the MDCEV model are randomly adjusted by 0.02.

```
#>      hunt_trap hunt_waterfowl     motor_land    motor_water          photo
#>           0.80           0.24           5.92           3.53          11.00
#>      ski_cross       ski_down
#>           3.12           1.85
#>
#> Parameter estimates --------------------------------
#>                         Estimate Std.dev z.stat n_eff Rhat
#> gamma_beach                5.678   1.066   5.32   302 1.00
#> gamma_birding              8.121   1.925   4.22   472 0.99
#> gamma_camping              3.791   0.812   4.67   463 1.00
#> gamma_cycling              8.235   1.615   5.10   416 0.99
#> gamma_fish                 7.203   1.788   4.03   543 1.00
#> gamma_garden              12.485   1.889   6.61   362 0.99
#> gamma_golf                 6.480   1.464   4.43   319 1.00
#> gamma_hiking              15.217   2.411   6.31   525 1.00
#> gamma_hunt_birds           4.108   2.014   2.04   341 1.01
#> gamma_hunt_large           7.419   2.542   2.92   352 1.00
#> gamma_hunt_trap            5.671   4.956   1.14   423 1.00
#> gamma_hunt_waterfowl       5.014   7.314   0.69   388 1.00
#> gamma_motor_land           8.620   2.381   3.62   526 1.00
#> gamma_motor_water          6.577   1.421   4.63   390 1.00
#> gamma_photo                8.662   1.777   4.88   197 1.04
#> gamma_ski_cross            3.333   0.785   4.25   440 1.00
#> gamma_ski_down             4.916   1.621   3.03   456 1.00
#> alpha_num                  0.725   0.008  94.68   459 1.00
#> sd.gamma_beach             1.239   0.222   5.58   319 1.01
#> sd.gamma_birding           1.894   0.665   2.85   179 1.03
#> sd.gamma_camping           1.254   0.254   4.94   326 1.00
#> sd.gamma_cycling           1.293   0.262   4.93   404 1.00
#> sd.gamma_fish              1.268   0.234   5.43   640 1.00
#> sd.gamma_garden            1.262   0.234   5.40   268 1.02
#> sd.gamma_golf              1.532   0.453   3.38   306 0.99
#> sd.gamma_hiking            1.319   0.261   5.05   200 1.00
#> sd.gamma_hunt_birds        1.764   0.963   1.83   551 1.01
#> sd.gamma_hunt_large        1.418   0.445   3.19   840 0.99
#> sd.gamma_hunt_trap         2.359   2.573   0.92   362 1.00
#> sd.gamma_hunt_waterfowl    3.875  12.454   0.31   312 1.02
#> sd.gamma_motor_land        1.502   0.456   3.29   334 1.00
#> sd.gamma_motor_water       1.420   0.335   4.24   398 1.00
#> sd.gamma_photo             1.311   0.272   4.82   455 0.99
#> sd.gamma_ski_cross         1.439   0.367   3.92   276 1.00
#> sd.gamma_ski_down          1.569   0.544   2.88   419 1.00
#> sd.alpha_num               0.514   0.010  51.01   183 1.02
#> Note: Scale parameter fixed to 1.
#> Note: All non-numeraire alpha's fixed to 0.
#> Note from Rstan: 'For each parameter, n_eff is a crude measure of effective sample
#> size, and Rhat is the potential scale reduction factor on split chains (at
#> convergence, Rhat=1)'
```

The results show the means of the random parameters followed by the estimated standard deviations. The standard deviations that are estimated to be different from zero suggest there is heterogeneity in preference parameters. The correlated random parameters specification can be estimated by setting `random_parameters = "corr"`. Bhat and Sen (2006) provide an empirical application of this type of model.

Computational and estimation issues

KT models are notoriously tricky to estimate relative to standard discrete choice models. This section provides some guidance for estimating these models and common convergence issues:

- **Starting values:** Model parameter estimates can be sensitive to starting values, especially the more complex LC-KT specification. Users should use several different initial parameter values for model estimation to ensure robust results and a global maxima is found rather than a local maxima. The default behaviour for LC-KT models is to use KT parameters as starting values.

In practice the author has found this to be quite effective at finding global maxima. However, users are encouraged to use random starting values as a robustness check.

- **Identification issues:** Depending on the model specification and included variables the model may not be properly identified. If you receive an error such as `Error in chol.default(-H) : the leading minor of order 9 is not positive definite` or `In sqrt(diag(cov_mat)) : NaNs produced`, this usually suggests an identification issue. Users should double check all variables included in the model are appropriate. One solution is to start with a simpler model first and then slowly add variables to help locate any problematic variables.
- **Parameter estimates near boundaries:** Interpret models with parameter estimates that are near the boundaries (e.g. α close to 1) with caution. Users are recommended to re-estimate the model with starting values far from this boundary.
- **Bayesian estimation**: For models estimated using Bayesian estimation, users should consult the **rstan** User Guide for additional guidance on model estimation options and postestimation checks (Stan Development Team, 2019). Additional information is available by typing `help(rstan)`.

Simulating KT demand and welfare scenarios

The **rmdcev** package includes simulation functions for calculating welfare measures and forecasting demand under alternative policy scenarios. The overall approach used for simulation is first introduced and then code examples are given.

Overview of simulation steps

Once the model parameters are estimated, there are two steps to simulation in KT models. In the first step we draw simulated values for the unobserved heterogeneity term (ε) using Monte Carlo techniques. The second step uses these error draws, the previously estimated model parameters, and the underlying data to calculate Marshallian demands for forecasting or Hicksian demands for welfare analysis. These two steps are described below.

Step 1: simulating unobserved heterogeneity

Monte Carlo simulation techniques can be employed to draw simulated values of the unobserved heterogeneity (ε) using either unconditional or conditional draws.

1. Unconditional error draws: draw from the entire distribution of unobserved heterogeneity using the following formula

$$\varepsilon_k = -log(-log(draw(0,1))) * \sigma, \tag{18}$$

where $draw(0,1)$ is a draw between 0 and 1 and σ is the scale parameter. **rmdcev** allows errors to be drawn using uniform draws or the Modified Latin Hypercube Sampling algorithm (Hess et al., 2006).

2. Conditional error draws: draw errors terms to reflect behaviour and dependent on whether alternative is consumed or not (von Haefen, 2003; von Haefen et al., 2004):

- If $x_k > 0$, set $\varepsilon_k = (V_1 - V_k)/\sigma$ for the MDCEV specifications where V_1 and V_k depend on the model specification as detailed above. If using the environmental economics KT model specification ("kt_ee"), set $\varepsilon_k = g_k(.)$ from Equation (12).
- If $x_k = 0$, $\varepsilon_k < (V_1 - V_k)/\sigma$ and simulate ε_k from the truncated type I extreme value distribution such that

$$\varepsilon_k = -log(-log(draw(0,1) * exp(-exp(\frac{V_1 - V_k}{\sigma})))) * \sigma \text{ for the MDCEV specifications, or} \tag{19}$$

$$\varepsilon_k = -log(-log(draw(0,1) * exp(-exp(-g_k(.))))) * \sigma \text{ for the KT-EE specification.} \tag{20}$$

In the conditional error draw approach, we normalize $\varepsilon_1 = 0$.

The main differences between these two error draw approaches is that in the conditional approach, errors are drawn such that the model perfectly predicts the observed consumption patterns in the baseline state (von Haefen and Phaneuf, 2005). The conditional approach uses observed behaviour by individuals to characterize unobserved heterogeneity and can be useful for scenario simulation as the baseline matches observed behavior. This is especially true if poor in-sample behavioral predictions is found using the unconditional approach (von Haefen, 2003). The unconditional approach draws all errors based on distributional assumptions and is necessary for out-of-sample forecasting. If the model correctly specifies the data generating process, the sample means of the conditional and unconditional approaches should converge in expectation. Another difference between the two approaches is that the unconditional approach uses more computation time as there is a need to calculate consumption patterns in the baseline state as well as simulate the entire distribution of unobserved heterogeneity.

Step 2: Calculating welfare measures and demand forecasts

With the error draws in hand, the second step is to simulate demand or welfare changes. Compared to welfare measures in discrete choice models, welfare calculation in KT models is more challenging because of the two KT conditions in Equation (2). For a given policy scenario, a priori, we do not know which alternatives have a positive or zero consumption level. **rmdcev** implements the Pinjari and Bhat (2011) efficient demand forecasting routine for simulating demand behaviour for MDCEV models which relies on calculating Marshallian demands. For welfare calculations, we need to calculate the expenditure function in Equation (3) which relies on Hicksian demands. These are calculated using the approach described by Lloyd-Smith (2018) and the **rmdcev** extends these approaches to the environmental economics KT model specifications. The demand and welfare simulation approaches share a lot of commonalities and thus only the approach used for welfare calculations are fully described in the appendix. The specific steps for demand simulation is explained in-depth in Pinjari and Bhat (2011) and the interested reader is encouraged to read Section 4 of the paper for the exact details.

Welfare analysis

In **rmdcev**, the functions for welfare and demand simulation have been divided into 3 steps to allow users to parallelize operations as necessary.

We first estimate the model using `mdcev` and we set `std_errors = "mvn"` to generate multivariate normal draws as these will be required to generate standard errors for calculations.

```
mdcev_mle <- mdcev(~ageindex,
                  data = data_model,
                  model = "hybrid",
                  algorithm = "MLE",
                  std_errors = "mvn",
                  print_iterations = FALSE)

#> Using MLE to estimate KT model
```

1. Define policy scenarios In the first step, we define the number of alternative policy scenarios to use in simulation and then specify changes to the ψ variables and prices of alternatives. The CreateBlankPolicies function has been created to easily set-up the required lists for the simulation. These policies can then be manually edited according to the specific policy scenario. For prices, **rmdcev** is set up to accept additive changes in prices that impact all individuals the same. For the ψ and ϕ variable changes, the package is set up to accept any new values for these variables. Depending on the number of individuals and number of policies, the generated policies list can be quite large. If the user is only interested in assessing price changes, then you can use `price_change_only = TRUE` which ensures duplicate ψ and ϕ data is not created.

In this example, we are interested in two separate policies. The first policy increases the costs of all recreation activities by $1 and the second policy increases the cost of all four hunting activities by $10. The policy set-up for these two scenarios is demonstrated below.

```
nalts <- mdcev_mle$stan_data[["J"]]
npols <- 2

policies<-  CreateBlankPolicies(npols = npols,
                                model = mdcev_mle,
                                price_change_only = TRUE)

policies$price_p[[1]] <- c(0, rep(1, nalts))
policies$price_p[[2]][10:13] <- rep(10, 4)
```

For policy scenarios that involve changes in the ψ or ϕ variables, the user can change the dat_psi or dat_phi list of the policies object. For example, the following code will increase the value of the first ψ variable by 20% in policy scenario 2.

```
policies_2 <-  CreateBlankPolicies(npols = npols,
                                model = mdcev_mle,
                                price_change_only = FALSE)

policies_2$dat_psi_p[[2]][, 1] <- policies_2$dat_psi_p[[2]][, 1] * 1.2
```

2. Prepare simulation data The second step is to combine the parameter estimates, data, and policy scenarios into a data format for simulation. The PrepareSimulationData function uses the model fit and the user defined policy scenarios to create this specific data format. This function separates the output into individual-specific data (df_indiv), data common to all individuals (df_common), and simulation options (sim_options).

```
df_sim <- PrepareSimulationData(mdcev_mle, policies)
```

3. Simulate MDCEV model The third step is to simulate the policy scenario using the formatted data and the mdcev.sim function. The specific steps for the simulation algorithms are described in Appendix A. The user chooses the type of error draws (unconditional or conditional as described above), the number of error draws, and whether to simulate the demand or welfare changes.

```
welfare <- mdcev.sim(df_sim$df_indiv,
                     df_common = df_sim$df_common,
                     sim_options = df_sim$sim_options,
                     cond_err = 1,
                     nerrs = 25,
                     sim_type = "welfare")

#> Using hybrid approach in simulation...
#> 3.00e+05simulations finished in0.07minutes.(75377per second)

summary(welfare)

#> # A tibble: 2 x 5
#>   policy    mean std.dev `ci_lo2.5%` `ci_hi97.5%`
#>   <chr>    <dbl>   <dbl>       <dbl>        <dbl>
#> 1 policy1 -126.    0.189      -126.        -126.
#> 2 policy2  -20.6   0.458       -21.3        -19.7
```

The output of the mdcev.sim for welfare analysis is an object of class mdcev.sim which contains a list of matrices where each element of the list is for an individual and the matrix consists of rows for each policy scenario and columns for each parameter simulation.

The summary function computes summary statistics across all individuals. For example, the average welfare change for a $1 daily increase in all recreation costs (i.e. Policy 1) is -$126.

The reason these last two steps are separate is to allow users to parallelize the simulation step as the last step can be computationally intensive. The number of simulations is a multiplicative function of the number of individuals, number of policies, number of parameter estimate simulations, and the number of error draws (I x *npols* x *nsims* x *nerrs*). Even for modestly sized data, the total number of simulations can easily reach well into the millions or billions. All simulations are conducted at the individual level which allows the user to easily parallelize the mdcev.sim function using the **parallel** package or similar packages.

Demand forecasting

This section demonstrates the demand forecasting capabilities of **rmdcev**. Please refer to the previous section for an overview of the three steps to simulation.

```
policies <- CreateBlankPolicies(npols = 2, model = mdcev_mle)

policies$price_p[[1]] <- c(0, rep(1, nalts))
policies$price_p[[2]][10:13] <- rep(10, 4)
```

```
df_sim <- PrepareSimulationData(mdcev_mle, policies)

demand <- mdcev.sim(df_sim$df_indiv,
                        df_common = df_sim$df_common,
                        sim_options = df_sim$sim_options,
                        cond_err = 1,
                        nerrs = 25,
                        sim_type = "demand")

#> Using hybrid approach in simulation...
#> 5.40e+06simulations finished in0.07minutes.(1360202per second)

summary(demand)

#> # A tibble: 36 x 6
#> # Groups:   policy [2]
#>    policy    alt     mean std.dev `ci_lo2.5%` `ci_hi97.5%`
#>    <chr>   <int>    <dbl>   <dbl>       <dbl>        <dbl>
#>  1 policy1     0 68946.      8.62    68933.       68962.
#>  2 policy1     1     6.24    0.02        6.2          6.28
#>  3 policy1     2    11.2     0.05       11.1         11.2
#>  4 policy1     3     2.34    0.02        2.31         2.37
#>  5 policy1     4     7.24    0.04        7.18         7.3
#>  6 policy1     5     3.76    0.02        3.72         3.78
#>  7 policy1     6    20.7     0.06       20.7         20.8
#>  8 policy1     7     5.29    0.01        5.28         5.3
#>  9 policy1     8    36.1     0.15       35.8         36.4
#> 10 policy1     9     0.55    0           0.53         0.55
#> # ... with 26 more rows
```

The output of the demand simulation a `mdcev.sim` object with a list of I elements, one for each individual. Within each element there are nsim lists each containing a matrix of demands. The rows of the matrix are for each policy scenario and the columns represent each alternative. The summary function computes summary statistics.

Generating simulated data

The **rmdcev** package has the capability to simulate KT data. Simulated KT data can be easily created for model assessment and Monte Carlo analysis using the `GenerateMDCEVData` function. The following example will generate a simulated data set with 1,000 individuals, 10 non-numeraire alternatives, and particular parameter values.

```
model = "gamma"
nobs = 1000
nalts = 10
sim.data <- GenerateMDCEVData(model = model,
                  nobs = nobs,
                  nalts = nalts,
                  psi_j_parms = c(-5, 0.5, 2), # alt-specific variables
                  psi_i_parms = c(-1.5, 3, -2, 1, 2), # individual-specific variables
                  gamma_parms = stats::runif(nalts, 1, 10),
                  alpha_parms = 0.5,
                  scale_parms = 1)

#> Sorting data by id.var then alt...
#> Checking data...
#> Data is good
```

Next, we can estimate the model using maximum likelihood techniques to recover the parameter estimates.

```
mdcev_mle <- mdcev(formula = ~ b1 + b2 + b3 + b4 + b5 + b6 + b7 + b8,
                           data = sim.data$data,
                           model = model,
                           psi_ascs = 0,
                           algorithm = "MLE",
                           print_iterations = FALSE)
```

Conclusions

The **rmdcev** package implements several Kuhn-Tucker model specifications including MDCEV with heterogeneity that can be continuous (i.e. random parameters) or discrete (i.e. latent classes). Models can be estimated using maximum likelihood or Bayesian techniques. This paper demonstrates the use of the package to estimate several model specifications and to derive demand forecasts and welfare implications of policy scenarios. To my knowledge, there is no other available statistical package that can estimate welfare implications of policy scenarios using MDCEV models. I hope that the publication of **rmdcev** will make KT modeling available to a wider audience.

Appendix A: Specific steps for simulating KT models

Welfare and demand simulation follow similar approaches and this section details the welfare simulation approach. There are two algorithms that differ depending on the model specification. If a single α parameter is estimated (i.e. model = "hybrid" or "hybrid0"), then we can use the hybrid approach to welfare simulation. If there are heterogeneous α parameters (i.e. model = "gamma", "alpha", or "kt_ee"), then we can use the general approach to welfare simulation. The hybrid approach is less computationally intensive and provides an exact analytical solution but the general approach can be used with all utility specifications. The specific steps for both algorithms are described below. Additional details are provided in Lloyd-Smith (2018).

Steps in algorithm for hybrid-profile MDCEV utility specifications

Step 0: Assume that only the numeraire alternative is chosen and let the number of chosen alternatives equal one (M=1).

Step 1: Using the data, model parameters, and either conditional or unconditional simulated error term draws, calculate the price-normalized baseline utility values (ψ_k/p_k) for all alternatives. Sort the K alternatives in the descending order of their price-normalized baseline utility values. Note that the numeraire alternative is in the first place. Go to step 2.

Step 2: Compute the value of λ^E using the following equation:

$$\frac{1}{\lambda^E} = \left[\frac{\alpha \bar{U} + \sum_{m=2}^{M} \gamma_m \psi_m}{\sum_{m=2}^{M} \gamma_m \psi_m \left(\frac{p_m}{\psi_m} \right)^{\frac{\alpha}{\alpha-1}} + \psi_1 \left(\frac{p_1}{\psi_1} \right)^{\frac{\alpha}{\alpha-1}}} \right]^{\frac{\alpha-1}{\alpha}}. \tag{21}$$

Go to step 3.

Step 3: If $\frac{1}{\lambda^E} > \frac{\psi_{M+1}}{p_{M+1}}$, go to step 4. Else if $\frac{1}{\lambda^E} < \frac{\psi_{M+1}}{p_{M+1}}$, set $M = M + 1$. If $M < K$, go back to step 2. If $M = K$, go to step 4.

Step 4: Compute the optimal Hicksian consumption levels for the first I alternatives in the above descending order using the following equations

$$x_1 = \left(\frac{p_1}{\lambda^E \psi_1} \right)^{\frac{1}{\alpha_1 - 1}}, \text{ and} \tag{22}$$

$$x_m = \left[\left(\frac{p_m}{\lambda^E \psi_m} \right)^{\frac{1}{\alpha_m - 1}} - 1 \right] \gamma_m, \quad \text{if } \; x_m > 0. \tag{23}$$

Set the remaining alternative consumption levels to zero and stop.

Steps in algorithm for general utility specifications

In this context, there is no closed-form expressions for λ^E and we need to conduct a numerical bisection routine. The following routine describes the approach for the MDCEV utility specifications. The approach used for the KT-EE specification is omitted due to space, but the overall strategy is the same with the only differences being the definitions for utility functions and optimal demands. Let $\hat{\lambda^E}$ and $\hat{U}$ be estimates of λ^E and U and let tol_λ and tol_U be the tolerance levels for estimating λ^E and U

that can be arbitrarily small. The algorithm works as follows:

Step 0: Assume that only the numeraire is chosen and let the number of chosen alternatives equal one (M=1).

Step 1: Using the data, model parameters, and either conditional or unconditional simulated error term draws, calculate the price-normalized baseline utility values (ψ_k / p_k) for all alternatives. Sort the K alternatives in the descending order of their price-normalized baseline utility values. Note that the numeraire is in the first place. Go to step 2.

Step 2: Let $\frac{1}{\hat{\lambda^E}} = \frac{\psi_{M+1}}{p_{M+1}}$ and substitute $\hat{\lambda^E}$ into the following equation to obtain an estimate of $\hat{U}$.

$$\bar{U} = \sum_{M=2}^{M} \frac{\gamma_m}{\alpha_m} \psi_m \left[\left(\frac{p_m}{\lambda^E \psi_m} \right)^{\frac{\alpha_m}{\alpha_m - 1}} - 1 \right] + \frac{\psi_1}{\alpha_1} \left(\frac{p_1}{\lambda^E \psi_1} \right)^{\frac{\alpha_1}{\alpha_1 - 1}} . \tag{24}$$

Step 3: If $\hat{U} < \bar{U}$, go to step 4. Else, if $\hat{U} \geq \bar{U}$, set $\frac{1}{\lambda_l^E} = \frac{\psi_{M+1}}{p_{M+1}}$ and $\frac{1}{\lambda_u^E} = \frac{\psi_M}{p_M}$. Go to step 5.

Step 4: Set $M = M + 1$. If $M < K$, go to step 2. Else if $M = K$, set $\frac{1}{\lambda_l^E} = 0$ and $\frac{1}{\lambda_u^E} = \frac{\psi_K}{p_K}$. Go to step 5.

Step 5: Let $\hat{\lambda^E} = (\lambda_l^E + \lambda_u^E)/2$ and substitute $\hat{\lambda^E}$ into the equation of step 2 to obtain an estimate of $\hat{U}$. Go to step 6.

Step 6: If $|\lambda_l^E - \lambda_u^E| \leq \ tol_\lambda$ or $|\hat{U} - \bar{U}| \leq \ tol_U$, go to step 7. Else if $\hat{U} < \bar{U}$, update $\lambda_u^E = (\lambda_l^E + \lambda_u^E)/2$ and go to step 5. Else if $\hat{U} > \bar{U}$, update $\lambda_l^E = (\lambda_l^E + \lambda_u^E)/2$ and go to step 5.

Step 7: Compute the optimal Hicksian consumption levels for the first M alternatives in the above descending order using Equation (22). Set the remaining alternative consumption levels to zero and stop.

Acknowledgements

Thanks to Joshua K Abbott and Allen Klaiber whose codes were helpful in putting this package together.

Bibliography

C. Bhat and A. Pinjari. Multiple discrete-continuous choice models: A reflective analysis and a prospective view. In S. Hess and A. Daly, editors, *Handbook of Choice Modelling*, chapter 19, pages 427–454. Edward Elgar Publishing, 2014. [p266]

C. R. Bhat. The multiple discrete-continuous extreme value (MDCEV) model: Role of utility function parameters, identification considerations, and model extensions. *Transportation Research Part B: Methodological*, 42(3):274–303, 2008. ISSN 0191-2615. URL `10.1016/j.trb.2007.06.002`. [p266, 268, 270, 275]

C. R. Bhat and S. Sen. Household vehicle type holdings and usage: an application of the multiple discrete-continuous extreme value (MDCEV) model. *Transportation Research Part B: Methodological*, 40(1):35–53, Jan. 2006. ISSN 0191-2615. URL `10.1016/j.trb.2005.01.003`. [p284]

B. Carpenter, A. Gelman, M. Hoffman, D. Lee, B. Goodrich, M. Betancourt, M. Brubaker, J. Guo, P. Li, and A. Riddell. Stan: A Probabilistic Programming Language. *Journal of Statistical Software, Articles*, 76(1):1–32, 2017. ISSN 1548-7660. URL `10.18637/jss.v076.i01`. [p267, 278]

Y. Croissant. *mlogit: Multinomial Logit Models*, 2019. URL `https://CRAN.R-project.org/package=mlogit`. R package version 1.0-1. [p266]

Federal, Provincial, and Territorial Governments of Canada. 2012 Canadian Nature Survey: Awareness, participation, and expenditures in nature-based recreation, conservation, and subsistence activities. Technical report, Canadian Councils of Resource Ministers, Ottawa, ON, 2014. [p270]

J. Gabry, D. Simpson, A. Vehtari, M. Betancourt, and A. Gelman. Visualization in bayesian workflow. *J. R. Stat. Soc. A*, 182:389–402, 2019. URL `10.1111/rssa.12378`. [p280]

J. A. Herriges, C. L. Kling, and D. J. Phaneuf. What's the use? welfare estimates from revealed preference models when weak complementarity does not hold. *Journal of Environmental Economics* *and Management*, 47(1):55–70, 2004. URL https://doi.org/10.1016/S0095-0696(03)00058-5. Publisher: Elsevier. [p269]

S. Hess and D. Palma. Apollo: A flexible, powerful and customisable freeware package for choice model estimation and application. *Journal of Choice Modelling*, page 100170, June 2019. ISSN 1755-5345. URL 10.1016/j.jocm.2019.100170. [p266, 267]

S. Hess, K. E. Train, and J. W. Polak. On the use of a Modified Latin Hypercube Sampling (MLHS) method in the estimation of a Mixed Logit Model for vehicle choice. *Transportation Research Part B: Methodological*, 40(2):147–163, Feb. 2006. ISSN 0191-2615. URL 10.1016/j.trb.2004.10.005. [p285]

A. Jasra, C. C. Holmes, and D. A. Stephens. Markov chain monte carlo methods and the label switching problem in bayesian mixture modeling. *Statist. Sci.*, 20(1):50–67, 02 2005. URL https://doi.org/10.1214/088342305000000016. [p270]

K. Kuriyama, M. Hanemann, and J. Hilger. A latent segmentation approach to a Kuhn-Tucker model: An application to recreation demand. *Journal of Environmental Economics and Management*, 60(3): 209–220, Nov. 2010. ISSN 0095-0696. URL 10.1016/j.jeem.2010.05.005. [p269, 283]

P. Lloyd-Smith. A new approach to calculating welfare measures in kuhn-tucker demand models. *Journal of Choice Modelling*, 26:19 – 27, 2018. ISSN 1755-5345. URL https://doi.org/10.1016/j.jocm.2017.12.002. [p286, 289]

P. Lloyd-Smith. The Economic Benefits of Recreation in Canada. *Canadian Journal of Economics*, forthcoming. [p271]

P. Lloyd-Smith, J. K. Abbott, W. Adamowicz, and D. Willard. Decoupling the Value of Leisure Time from Labor Market Returns in Travel Cost Models. *Journal of the Association of Environmental and Resource Economists*, 6(2):215–242, Jan. 2019. ISSN 2333-5955. URL 10.1086/701760. [p275]

C. Muth, Z. O. Jonah, and Gabry. User-friendly bayesian regression modeling: A tutorial with rstanarm and shinystan. *The Quantitative Methods for Psychology*, 14(2):99–119, 2018. URL 10.20982/tqmp.14.2.p099. [p280]

K. Mäler. *Environmental Economics: A Theoretical Inquiry*. Johns Hopkins University Press for Resources for the Future, 1974. [p268]

A. R. Pinjari and C. R. Bhat. Computationally efficient forecasting procedures for Kuhn-Tucker consumer demand model systems: Application to residential energy consumption analysis. Technical report, Department of Civil and Environmental Engineering, University of South Florida, 2011. [p286]

M. Sarrias and R. Daziano. Multinomial Logit Models with Continuous and Discrete Individual Heterogeneity in R: The gmnl Package. *Journal of Statistical Software*, 79(1):1–46, July 2017. ISSN 1548-7660. URL http://dx.doi.org/10.18637/jss.v079.i02. [p266]

A. Sobhani, N. Eluru, and A. Faghih-Imani. A latent segmentation based multiple discrete continuous extreme value model. *Transportation Research Part B: Methodological*, 58:154–169, Dec. 2013. ISSN 0191-2615. URL 10.1016/j.trb.2013.07.009. [p269, 283]

Stan Development Team. RStan: the R interface to Stan. R package version 2.19, 2019. URL https://mc-stan.org/rstan. [p280, 285]

K. Train. *Discrete Choice Methods with Simulation*. Cambridge University Press, 2009. [p270]

R. H. von Haefen. Incorporating observed choice into the construction of welfare measures from random utility models. *Journal of Environmental Economics and Management*, 45(2):145–165, Mar. 2003. ISSN 0095-0696. URL 10.1016/S0095-0696(02)00047-5. [p285, 286]

R. H. von Haefen and D. J. Phaneuf. Kuhn-Tucker Demand System Approaches to Non-Market Valuation. In R. Scarpa and A. Alberini, editors, *Applications of Simulation Methods in Environmental and Resource Economics*, pages 135–157. Springer Netherlands, Dordrecht, 2005. ISBN 978-1-4020-3684-2. [p266, 267, 269, 270, 285]

R. H. von Haefen, D. J. Phaneuf, and G. R. Parsons. Estimation and Welfare Analysis with Large Demand Systems. *Journal of Business & Economic Statistics*, 22(2):194–205, 2004. ISSN 0735-0015. [p266, 285]

A. Zeileis and Y. Croissant. Extended Model Formulas in R: Multiple Parts and Multiple Responses. *Journal of Statistical Software*, 34(1):1–13, Apr. 2010. ISSN 1548-7660. URL 10.18637/jss.v034.i01. [p273]

Patrick Lloyd-Smith
University of Saskatchewan
Department of Agricultural and Resource Economics
Global Institute for Water Security
Room 3D34, Agriculture Building 51 Campus Drive
Saskatoon, SK S7N 5A8 Canada
https://plloydsmith.github.io/
patrick.lloydsmith@usask.ca

NTS: An R Package for Nonlinear Time Series Analysis

by Xialu Liu, Rong Chen, and Ruey Tsay

Abstract Linear time series models are commonly used in analyzing dependent data and in forecasting. On the other hand, real phenomena often exhibit nonlinear behavior and the observed data show nonlinear dynamics. This paper introduces the R package **NTS** that offers various computational tools and nonlinear models for analyzing nonlinear dependent data. The package fills the gaps of several outstanding R packages for nonlinear time series analysis. Specifically, the **NTS** package covers the implementation of threshold autoregressive (TAR) models, autoregressive conditional mean models with exogenous variables (ACMx), functional autoregressive models, and state-space models. Users can also evaluate and compare the performance of different models and select the best one for prediction. Furthermore, the package implements flexible and comprehensive sequential Monte Carlo methods (also known as particle filters) for modeling non-Gaussian or nonlinear processes. Several examples are used to demonstrate the capabilities of the **NTS** package.

Introduction: nonlinear time series analysis in R

Time series analysis investigates the dynamic dependence of data observed over time or in space. While linear time series analysis has been extensively studied in the literature with many software packages widely available, nonlinear time series analysis only attracts limited attention. Although there exist some software packages for analyzing nonlinear time series focusing on different sets of tools, there are still significant gaps in capability. The **NTS** (Tsay et al., 2020), a recent R package, provides a number of functions for simulating, analyzing, and predicting nonlinear time series data. The available models include univariate and multivariate TAR models, conditional intensity models, nonlinear state-space models, and functional time series models. The package also features various nonlinearity tests and sequential Monte Carlo (SMC) methods. While **NTS** package does not intend to be comprehensive, it fills the important missing parts of the existing packages, providing users valuable tools for analyzing dependent data with nonlinear dynamics. The package is now available from the Comprehensive R Archive Network at `http://CRAN.R-project.org/package=NTS`.

NTS incorporates the latest developments in statistical methods and algorithms for analyzing nonlinear time series data, and it makes the following contributions: (1) **NTS** offers various computational tools with a wide range of applications, and it fills the gaps left by the existing R functions. There are several R packages focusing on nonlinear time series. The **nonlinearTseries** (Garcia and Sawitzki, 2020) package implements the methods based on information theory, the **NlinTS** (Youssef, 2020) package introduces functions for causality detection and neural networks, and the **nlts** (Bjornstad, 2018) package emphasizes nonparametric autoregression and tests. **NTS**, providing computational tools for TAR models, ACMx models, convolutional functional autoregressive (CFAR) models, and non-Gaussian and nonlinear state-space models, consists of some of the missing pieces in the current coverage, hence making a more completed toolkit for nonlinear time series analysis in R. Other well-known modern methods for nonlinear data such as smoothing, deep learning and random forest that have been implemented in packages **sm** (Bowman and Azzalini, 2018), **tree** (Ripley, 2019) and **randomForest** (Breiman et al., 2018) can be adopted for nonlinear time series analysis, even though they are mainly designed for independent data. Hence, they are not included in this package. (2) **NTS** provides complete solutions with superior performance for the nonlinear models entertained. For example, **NTS** implements estimation, prediction, model building and model comparison procedures for TAR models. It allows the threshold variable in the model to be a lag variable or an exogenous variable, while the **TAR** (Zhang and Nieto, 2017) package, using Markov Chain Monte Carlo and Bayesian methods aiming to deal with missing values, assumes the threshold variable is exogenous. Threshold estimation methods in **NTS**, which perform recursive least squares or nested sub-sample searching, are more computationally efficient than the conditional least squares methods implemented in package **tsDyn** (Narzo et al., 2020). Furthermore, the threshold nonlinearity test proposed by Tsay (1989) in **NTS** is specifically designed for self-exciting TAR (SETAR) models while the existing R package **nonlinearTseries** just conducts general nonlinearity tests. In addition, **NTS** utilizes the out-of-sample forecasting to evaluate different TAR models to avoid overfitting, while other R packages such as **tsDyn** just compare TAR models based on AIC and residuals. (3) **NTS** offers additional options to existing packages with more flexibility. Specifically, **NTS** offers R functions to fit the ACMx model for time series analysis of count data, which allow the conditional distribution to be double Poisson, while the **tscount** (Liboschik et al., 2020) package uses the generalized linear models and only considers Poisson and negative binomial distributions. Another example is that

NTS implements the estimation and prediction procedures of CFAR models proposed by Liu et al. (2016), which give an intuitive and direct interpretation for functional time series analysis and provide more flexibility for estimation to deal with irregular observation locations compared to functional autoregressive models developed by Bosq (2000) introduced in the **ftsa** (Hyndman and Shang, 2020) package. (4) **NTS** provides easy access to SMC methods with various options for statistical inference. It contains different R functions which can be easily implemented for filtering and smoothing and are much more user-friendly, while the **SMC** (Goswami, 2011) package only writes a generic function for SMC and requires more effort from users.

The goal of this paper is to highlight the main functions of the **NTS** package. In the paper, we first consider different models for nonlinear time series analysis, and provide an overview of the available functions for parameter estimation, prediction and nonlinearity tests in the **NTS** package. Then we discuss the functions for SMC methods and demonstrate their applications via an example. Conclusions are given at the end.

Models and methods available in NTS

TAR models

TAR models are a piecewise extension of the autoregressive (AR) model proposed by Tong (1978). It has been widely used in many scientific fields, such as economics (Tong and Lim, 1980; Tiao and Tsay, 1989), finance (Domian and Louton, 1997; Narayan, 2006), among others (Chen, 1995). The models are characterized by partitioning the Euclidean space into non-overlapping regimes via a threshold variable and fitting a linear AR model in each regime (Li and Tong, 2016). The partition is by various thresholds in the domain of the threshold variable.

Let $\{r_i \mid i = 0, \ldots, m\}$ be a sequence of real numbers satisfying

$$r_0 = -\infty < r_1 < r_2 < \ldots < r_{m-1} < r_m = \infty.$$

A time series $\{y_t | t = 1, \ldots n\}$ follows an m-regime TAR model with threshold variable z_t, threshold delay $d > 0$, and order $(p_1, \ldots, p_m)$, if

$$y_t = \begin{cases} \phi_{0,1} + \sum_{i=1}^{p_1} \phi_{i,1} y_{t-i} + \sigma_1 \epsilon_t, & \text{if } z_{t-d} \le r_1, \\ \phi_{0,2} + \sum_{i=1}^{p_2} \phi_{i,2} y_{t-i} + \sigma_2 \epsilon_t, & \text{if } r_1 < z_{t-d} \le r_2, \\ \cdots \\ \phi_{0,m} + \sum_{i=1}^{p_m} \phi_{i,m} y_{t-i} + \sigma_m \epsilon_t, & \text{if } r_{m-1} < z_{t-d}, \end{cases} \tag{1}$$

where $\phi_{i,j}$ are real numbers, $\sigma_1, \ldots, \sigma_m$ are positive real numbers, and ϵ_t are i.i.d random variates with mean 0 and variance 1. If the threshold variable $z_t = y_t$ for $t = 1, \ldots, n$, Model (1) is called a SETAR model with delay d. The coefficients $\phi_{i,j}$ must satisfy certain conditions for the stationarity of y_t. These conditions are complicated in general, but some special cases are available in the literature. See, for instance, Chen and Tsay (1991) and the references therein. In particular, it is interesting to point out that the stationarity of each marginal model in (1) is not needed for the stationarity of y_t. As a matter of fact, Model (1) would become more interesting when some of the marginal models are nonstationary.

Threshold estimation for two-regime TAR models

In this subsection, we introduce three algorithms for estimation of two-regime TAR models.

The two-regime TAR model can be rewritten as

$$y_t = (\boldsymbol{\beta}_1' \mathbf{x}_{t,1} + \sigma_1 \epsilon_t) I(z_{t-d} \le r_1) + (\boldsymbol{\beta}_2' \mathbf{x}_{t,2} + \sigma_2 \epsilon_t) I(z_{t-d} > r_1), \tag{2}$$

where $I(\cdot)$ is the indicator function, $\mathbf{x}_{t,j} = (1, y_{t-1}, \ldots, y_{t-p_j})'$, and $\boldsymbol{\beta}_j = (\phi_{0,j}, \phi_{1,j}, \ldots, \phi_{p_j,j})'$ collects the AR coefficients in regime j, for $j = 1, 2$.

Define $p = \max\{p_1, p_2, d\}$, and $\mathbf{x}_j = (\mathbf{x}_{p+1,j}, \ldots, \mathbf{x}_{n,j})'$ for $j = 1, 2$. Write $\mathbf{x}_1(r) = \mathbf{x}_1 * I(z_{t-d} \le r)$, and $\mathbf{x}_2(r) = \mathbf{x}_2 * I(z_{t-d} > r)$, where $*$ denotes the Hadamard product operator of matrices. Then Equation (2) can be re-expressed in a matrix form

$$\mathbf{y} = \mathbf{x}_1(r_1)\boldsymbol{\beta}_1 + \mathbf{x}_2(r_1)\boldsymbol{\beta}_2 + \boldsymbol{\varepsilon}, \tag{3}$$

where $\mathbf{y} = (y_{p+1}, \ldots, y_n)'$, $\boldsymbol{\varepsilon} = (\tilde{\epsilon}_{p+1}, \ldots, \tilde{\epsilon}_n)'$, and $\tilde{\epsilon}_t = [I(z_{t-d} \le r_1)\sigma_1 + I(z_{t-d} > r_1)\sigma_2]\epsilon_t$ for $t = p + 1, \ldots, n$.

(Conditional) least squares: For each fixed threshold candidate r, least squares method can be used to estimate the AR coefficients $\boldsymbol{\beta}_1$ and $\boldsymbol{\beta}_2$,

$$\hat{\boldsymbol{\beta}}_1(r) = [\mathbf{x}_1(r)'\mathbf{x}_1(r)]^{-1}\mathbf{x}_1(r)'\mathbf{y}, \quad \hat{\boldsymbol{\beta}}_2(r) = [\mathbf{x}_2(r)'\mathbf{x}_2(r)]^{-1}\mathbf{x}_2(r)'\mathbf{y}. \tag{4}$$

It yields the following error function

$$S_n(r) = \mathbf{y}'\mathbf{y} - \hat{\boldsymbol{\beta}}_1(r)'\mathbf{x}_1(r)'\mathbf{x}_1(r)\hat{\boldsymbol{\beta}}_1(r) - \hat{\boldsymbol{\beta}}_2(r)'\mathbf{x}_2(r)'\mathbf{x}_2(r)\hat{\boldsymbol{\beta}}_2(r). \tag{5}$$

To get sufficient number of observations in each regime for estimation, we assume that the threshold value r_1 lies in a bounded interval $[\underline{r}, \bar{r}]$. Then it can be estimated as

$$\hat{r}_1 = \arg \min_{r \in \{z_{p-d+1}, \ldots, z_{n-d}\} \cap [\underline{r}, \bar{r}]} S_n(r). \tag{6}$$

Recursive least squares: Recursive least squares method provides an efficient way to update the least squares solution with new observations, and is much less computationally expensive than the ordinary least squares method. When we traverse all possible thresholds and calculate $S_n(r)$ in (5), recursive least squares can be used to estimate $\boldsymbol{\beta}_1$ and $\boldsymbol{\beta}_2$ in (4) helping us effectively reduce the computational cost.

Let $\mathcal{S} = \{z_{p-d+1}, \ldots, z_{n-d}\} \cap [\underline{r}, \bar{r}]$ be the set containing all candidates for the threshold value, n_0 be the number of elements in $\mathcal{S}$, $z_{(j)}$ be the j-th largest value in set $\mathcal{S}$, and $t_{(j)}$ be the time index for $z_{(j)}$. In other words, $z_{(j)} = z_{t_{(j)}}$.

Here is the algorithm of recursive least squares for TAR model estimation:

1. When $z_{(1)}$ is used as a tentative threshold value to estimate $\boldsymbol{\beta}_1$,

$$\mathbf{P}_1(1) = [\mathbf{x}_1(z_{(1)})'\mathbf{x}_1(z_{(1)})]^{-1}, \quad \hat{\boldsymbol{\beta}}_1(z_{(1)}) = \mathbf{P}_1(1)\mathbf{x}_1(z_{(1)})'\mathbf{y}.$$

   When $z_{(n_0)}$ is used as a tentative threshold value to estimate $\boldsymbol{\beta}_2$,

$$\mathbf{P}_2(n_0) = [\mathbf{x}_2(z_{(n_0)})'\mathbf{x}_2(z_{(n_0)})]^{-1}, \quad \hat{\boldsymbol{\beta}}_2(z_{(n_0)}) = \mathbf{P}_2(n_0)\mathbf{x}_2(z_{(n_0)})'\mathbf{y}.$$

2. For $k = 2, \ldots, n_0$, we estimate the AR coefficients in regime 1 with the following

$$\begin{aligned}\mathbf{K}_1(k) &= \mathbf{P}_1(k-1)\mathbf{x}_{t_{(k)}+d,1}/[1 + \mathbf{x}'_{t_{(k)}+d,1}\mathbf{P}_1(k-1)\mathbf{x}_{t_{(k)}+d,1}],\\ \mathbf{P}_1(k) &= \mathbf{P}_1(k-1) - \mathbf{K}_1(k)\mathbf{x}'_{t_{(k)}+d,1}\mathbf{P}_1(k-1),\\ \hat{\boldsymbol{\beta}}_1(z_{(k)}) &= \hat{\boldsymbol{\beta}}_1(z_{(k-1)}) + \mathbf{K}_1(k)[y_{t_{(k)}+d} - \hat{\boldsymbol{\beta}}_1(z_{(k-1)})'\mathbf{x}_{t_{(k)}+d,1}].\end{aligned}$$

   For $k = n_0 - 1, \ldots, 1$, we estimate the AR coefficients in regime 2 with the following

$$\begin{aligned}\mathbf{K}_2(k) &= \mathbf{P}_2(k+1)\mathbf{x}_{t_{(k)}+d,2}/[1 + \mathbf{x}'_{t_{(k)}+d,2}\mathbf{P}_2(k+1)\mathbf{x}_{t_{(k)}+d,2}],\\ \mathbf{P}_2(k) &= \mathbf{P}_2(k+1) - \mathbf{K}_2(k)\mathbf{x}'_{t_{(k)}+d,2}\mathbf{P}_2(k+1),\\ \hat{\boldsymbol{\beta}}_2(z_{(k)}) &= \hat{\boldsymbol{\beta}}_2(z_{(k+1)}) + \mathbf{K}_2(k)[y_{t_{(k)}+d} - \hat{\boldsymbol{\beta}}_2(z_{(k+1)})'\mathbf{x}_{t_{(k)}+d,2}].\end{aligned}$$

3. With $\hat{\boldsymbol{\beta}}_1(z_{(j)})$ and $\hat{\boldsymbol{\beta}}_2(z_{(j)})$ for $j = 1, \ldots, n_0$, we can obtain $S_n(z_{(j)})$ and then estimate r_1 with (6).

Nested sub-sample search (NeSS) algorithm: NeSS algorithm proposed by Li and Tong (2016) produces a much faster way to search threshold candidates, and reduce the computational complexity dramatically.

Li and Tong (2016) shows that there exists a positive constant C depending only on $\mathbf{y}$, p_1 and p_2, such that

$$\sup_{r \in [\underline{r}, \bar{r}]} \left| \frac{C - S_n(r)}{n} - J(r) \right| \xrightarrow{p} 0,$$

where $J(r)$ is a non-stochastic continuous function over $[\underline{r}, \bar{r}]$, and it is strictly monotonically increasing in $[\underline{r}, r_1]$ and strictly monotonically deceasing in $[r_1, \bar{r}]$. It implies that $S_n(r)$ may have only one minimum value over the set $\{k\Delta : k \in \mathbb{Z}\} \cap [\underline{r}, \bar{r}]$ for some $\Delta > 0$. This provides theoretical support for the following NeSS algorithm to seek the minimizer of $S_n(r)$.

NeSS algorithm:

0. Get the initial feasible set $\mathcal{S} = \{z_{p-d+1}, \ldots, z_{n-d}\} \cap [\underline{r}, \bar{r}]$ for the threshold value estimation.
1. Obtain the 25th, 50th, and 75th percentiles of the feasible set, and define them as q_1, q_2 and q_3, respectively. Calculate $S_n(q_1)$, $S_n(q_2)$, and $S_n(q_3)$.

Table 1: Comparison among various R functions for SETAR model estimation (200 replicates)

Function	Sample size 200		Sample size 2000	
	Elapsed time	MSE	Elapsed time	MSE
`uTAR` with recursive least squares	2.802s	0.0017	44.020s	2.08e-05
`uTAR` with NeSS algorithm	20.360s	0.0017	25.878s	2.08e-05
`setar`	7.147s	0.0017	286.226s	2.08e-05

2. If $S_n(q_1) \leq S_n(q_2)$ and $S_n(q_1) \leq S_n(q_3)$, the feasible set is updated as $\mathcal{S} \cap (-\infty, q_2]$.
   If $S_n(q_2) < S_n(q_1)$ and $S_n(q_2) \leq S_n(q_3)$, the feasible set is updated as $\mathcal{S} \cap [q_1, q_3]$.
   Otherwise, the feasible set is updated as $\mathcal{S} \cap [q_2, +\infty)$.

   Repeat Steps 1-2 until the number of elements in the new feasible set is less than a pre-specified positive integer k_0.
3. Minimize $S_n(r)$ over the new feasible set and get $\hat{r}_1$.

Comparing to the standard search algorithm which traverses all the threshold candidates, NeSS algorithm reduces the number of least squares operations from $O(n)$ to $O(\log n)$.

R functions for TAR models in NTS

In the R package **NTS**, the function `uTAR` implements recursive least squares estimation or the NeSS algorithm for TAR model estimation. The two methods both have lower computational complexity than the existing R function `setar` designed for SETAR model estimation in the **tsDyn** package, which performs least squares estimation and adopts a single grid search algorithm.

To illustrate, we use the following data generating process to compare the performance of the three methods[1].

$$y_t = \begin{cases} 1 - 0.3y_{t-1} + 0.5y_{t-2} + \epsilon_t, & \text{if } y_{t-2} \leq 0.2, \\ -1 + 0.6y_{t-1} + 0.3y_{t-2} + \epsilon_t, & \text{if } y_{t-2} > 0.2. \end{cases} \tag{7}$$

Table 1 summarizes the average elapsed time and mean squared error (MSE) of the estimated threshold value for 200 replications. Recursive least squares method and NeSS algorithm implemented by `uTAR` both take shorter time than `setar` when sample size is large. It is also seen that when sample size is large, NeSS algorithm is the fastest, but when the sample size is relatively small, the recursive least squares method is the fastest.

Besides threshold value estimation for univariate time series, the **NTS** package implements data generating, forecasting, model checking, and model comparison procedures for both univariate and multivariate time series into user-friendly computational tools. Table 2 lists these functions of **NTS** related to TAR models. In the following we will demonstrate the usage of functions for univariate time series through the data generating process in Model (7).

Table 2: List of R functions about TAR models in the package **NTS**

	Function	Description
Univariate TAR model	`uTAR.sim`	Generate a univariate SETAR process for up to 3 regimes
	`uTAR`	Estimate univariate two-regime TAR models including threshold
	`uTAR.est`	Estimate multiple regimes TAR models with known threshold(s)
	`uTAR.pred`	Predict a fitted univariate TAR model
	`thr.test`	Test for threshold nonlinearity of a scalar series
Multivariate TAR model	`mTAR.sim`	Generate a multivariate two-regime SETAR process
	`mTAR`	Estimate multivariate two-regime TAR models including threshold
	`mTAR.est`	Estimate multivariate multiple-regime TAR models
	`ref.mTAR`	Refine a fitted multivariate two-regime TAR model
	`mTAR.pred`	Predict a fitted multivariate TAR model

The function `uTAR.sim` generates data from a given univariate SETAR model for up to three regimes with following arguments: `nob` is the sample size of the generated data, `arorder` specifies the AR orders for different regimes, `phi` is a real matrix containing the AR coefficients with one row for a

[1]The program is run on a personal computer with a 2.30GHz Intel Core(TM)i5-8259U CPU, 16GB RAM and 64-bit Operating system.

regime, d is the time delay, cnst is a vector of constant terms for the regimes, and sigma is a vector containing the standard deviations of the innovation process of the regimes. It also allows users to customize the burn-in period with option ini. The function returns a list of components including the generated data from the specified TAR model (series) and the innovation series (at).

We simulate the data generating process in Model (7) with the following code. Figure 1 shows the time series plot of the first 200 observations of the simulated data.

```
R> set.seed(1687)
R> y <- uTAR.sim(nob = 2000, arorder = c(2,2), phi = t(matrix(c(-0.3, 0.5, 0.6,
+    -0.3), 2, 2)), d = 2, thr = 0.2, cnst = c(1, -1), sigma = c(1, 1))
```

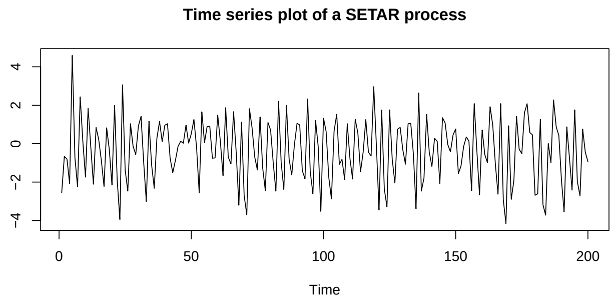


Figure 1: Time series plot of the first 200 observations generated from the SETAR model in Equation (7).

Estimation of the threshold value of the two-regime SETAR process can be done via the function uTAR as illustrated below:

```
R> thr.est<- uTAR(y = y$series, p1 = 2, p2 = 2, d = 2, thrQ = c(0, 1), Trim = c(0.1,
+    0.9), include.mean = T, method = "NeSS", k0 = 50)
Estimated Threshold:  0.1951103
Regime 1:
     Estimate Std. Error   t value     Pr(>|t|)
X1  1.0356009 0.04902797  21.12265 8.946275e-85
X2 -0.3017810 0.01581242 -19.08506 2.383743e-71
X3  0.4890477 0.02707987  18.05945 7.230880e-65
nob1 & sigma1: 1236 1.017973
Regime 2:
     Estimate Std. Error    t value     Pr(>|t|)
X1 -1.1352678 0.07222915 -15.717585 2.107275e-48
X2  0.5560001 0.03177212  17.499622 7.360494e-58
X3 -0.2122922 0.04641671  -4.573616 5.596852e-06
nob2 & sigma2:  762 1.034592

Overall MLE of sigma:  1.024343
Overall AIC:  101.8515

R> thr.est <- uTAR(y = y$series, p1 = 2, p2 = 2, d = 2, thrQ = c(0,1), Trim = c(0.1,
+     0.9), include.mean = T, method = "RLS")
Estimated Threshold:  0.1951103
Regime 1:
     Estimate Std. Error   t value     Pr(>|t|)
X1  1.0356009 0.04902797  21.12265 8.946275e-85
X2 -0.3017810 0.01581242 -19.08506 2.383743e-71
X3  0.4890477 0.02707987  18.05945 7.230880e-65
nob1 & sigma1:  1236 1.017973
Regime 2:
     Estimate Std. Error    t value     Pr(>|t|)
X1 -1.1352678 0.07222915 -15.717585 2.107275e-48
X2  0.5560001 0.03177212  17.499622 7.360494e-58
X3 -0.2122922 0.04641671  -4.573616 5.596852e-06
nob2 & sigma2:  762 1.034592
```

```
Overall MLE of sigma:  1.024343
Overall AIC:  101.8515
```

uTAR has the following arguments: y is a vector of observed time seres, p1 and p2 are the AR order of regime 1 and regime 2, respectively, d is the delay, and thrV contains the external threshold variable z_t which should have the same length as that of y. For SETAR models, thrV is not needed and should be set to NULL. thrQ determines the lower and upper quantiles to search for threshold value. Trim defines the lower and upper trimmings to control the minimum sample size in each regime and determine $[\underline{r}, \overline{r}]$ for estimation. include.mean is a logical value for including the constant term in each linear model. method decides the way to search the threshold value, and there are two choices, "RLS" for recursive least squares and "NeSS" for NeSS algorithm. k0 is only used when NeSS algorithm is selected to controls the maximum sub-sample size.

From the output, the estimated threshold value is 0.195, which is close to the true value 0.2. The estimated constant terms for regime 1 and regime 2 are 1.036 and -1.135, respectively. The estimated AR coefficients for regime 1 and regime 2 are -0.302, 0.489, 0.556, and -0.212, respectively. The estimated standard deviations of the innovation processes in two regimes are 1.018 and 1.035. As expected, all estimates are significant and close their true parameters.

Here we provide an incomplete list of the returned values of the function uTAR:

- residuals: estimated innovations or residuals series.
- coefs: a 2-by-$(p+1)$ matrix. The first row and second row show the estimated coefficients in regime 1 and 2, respectively.
- sigma: estimated covariances of the innovation process in regime 1 and regime 2.
- thr: estimated threshold value.

Estimation of a multiple-regime TAR model with pre-specified threshold values can be done by the function uTAR.est.

```
R> est <- uTAR.est(y = y$series, arorder = c(2, 2), thr = thr.est$thr, d = 2,
+    output = FALSE)
```

Here aroder is a row vector of positive integers containing the AR orders of all the regimes. thr collects the threshold values whose length should be the number of regimes minus 1. output is a logical value for printing out the estimation results with default being TRUE. The function uTAR.est returns the following components: coefs is a matrix with m rows in which each row contains the estimated parameters for one regime, sigma contains the estimated innovation variances for different regimes, residuals collects the estimated innovations, and sresi shows the standardized residuals.

The following R code provides one-step-ahead prediction with function uTAR.pred.

```
R> set.seed(12)
R> pred <- uTAR.pred(model = est, orig = 2000, h = 1, iteration = 100, ci = 0.95,
+    output = TRUE)
Forecast origin:  2000
Predictions: 1-step to  1 -step
     step  forecast
[1,]    1 -1.429635
Pointwise  95  % confident intervals
    step      Lowb      Uppb
int    1 -2.991667 0.6531542
```

The output above shows that the one-step ahead prediction for y_{2001} is -1.43. Various options in the function uTAR.pred provide users the flexibility to customize the forecasting origin with orig, forecast horizon with h, number of iterations with iterations, and confidence level with ci. The function uTAR.pred returns the prediction with pred.

The R function thr.test in the **NTS** package implements the F test designed for SETAR models and proposed by Tsay (1989). The test helps users detect the existence of nonlinear dynamics in the data. Below is the R code and output when we perform the nonlinearity tests with thr.test.

```
R> thr.test(y$series, p = 2, d = 2, ini = 40, include.mean = T)
SETAR model is entertained
Threshold nonlinearity test for (p,d):  2 2
F-ratio and p-value:  213.0101 1.511847e-119
```

ini is the initial number of data to start the recursive least square estimation. The output shows that p-value is very small, and it indicates that there is nonlinearity in the series y$series.

Back-testing can be used to evaluate the forecasting performance of a model and to conduct model comparison between different models. Back-testing for a univariate SETAR model is implemented through the function backTAR with syntax:

```
R> backTAR(model, orig, h = 1, iter = 3000)
```

where model is an object returned by uTAR or uTAR.est, h is the forecast horizon, and iter controls the number of simulation iterations in prediction.

The function returns the model, out-of-sample rolling prediction errors and predicted states. It also provides information for model comparison. The following example shows the out-of-sample forecasting performance of SETAR models with delay 2 and 1, respectively. It shows that the root MSE, mean absolute error, and biases of the model with delay 2 are all smaller than those of the model with delay 1. Hence, as expected, the model with delay 2 is preferred.

```
R> set.seed(11)
R> backTAR(est, 50, 1, 3000)
Starting forecast origin:  50
1-step to  1 -step out-sample forecasts
RMSE:  1.02828
 MAE:  0.8172728
Bias:  -0.001337478
Performance based on the regime of forecast origins:
Summary Statistics when forecast origins are in State:  1
Number of forecasts used:  1204
RMSEj:  1.029292
 MAEj:  0.8172963
Biasj:  0.00259177
Summary Statistics when forecast origins are in State:  2
Number of forecasts used:  746
RMSEj:  1.026645
 MAEj:  0.817235
Biasj:  -0.007679051

R> thr.est2 <- uTAR(y = y$series, p1 = 2, p2 = 2, d = 1, thrQ = c(0, 1),
+    Trim=c(0.1, 0.9), include.mean = T, method = "RLS")
R> est2 <- uTAR.est(y = y$series, arorder = c(2, 2), thr = thr.est2$thr, d = 1)
R> set.seed(11)
R> backTAR(est2, 50, 1, 3000)
Starting forecast origin:  50
1-step to  1 -step out-sample forecasts
RMSE:  1.38731
 MAE:  1.105443
Bias:  -0.006635381
Performance based on the regime of forecast origins:
Summary Statistics when forecast origins are in State:  1
Number of forecasts used:  1112
RMSEj:  1.360347
 MAEj:  1.090989
Biasj:  0.2462278
Summary Statistics when forecast origins are in State:  2
Number of forecasts used:  838
RMSEj:  1.4223
 MAEj:  1.124622
Biasj:  -0.3421769
```

The usage of functions for multivariate two-regime TAR models listed in Table 2, including mTAR.sim, mTAR, mTAR.pred, is similar to that of the univariate counterpart functions discussed before. The only exception is that these multivariate functions take different arguments to define the vector autoregressive(VAR) coefficients:

- phi1,phi2: VAR coefficient matrices of regime 1 and regime 2.
- sigma1,sigma2: innovation covariance matrices of regime 1 and regime 2.
- c1,c2: constant vectors of regime 1 and regime 2.
- delay: two elements (i, d) with "i" being the index of the component to be used as the threshold variable and "d" the delay for threshold variable.

The function mTAR conducts the nested sub-sample search algorithm and provides different choices of criterion for threshold selection with the option score, namely (AIC, det(RSS)). It has less computational cost, but only applies to two-regime models. mTAR.est can handle multiple regimes. They both return a list of components with the estimated VAR coefficients in beta, estimated innovation covariance matrices in sigma, and estimated innovations in residuals.

Analysis of non-Gaussian time series

Autoregressive conditional mean (ACM) models are designed for time series of count data, starting with the autoregressive conditional Poisson models, and various extensions of ACM models were investigated. The **NTS** includes a function ACMx for the estimation of ACMx models. Let y_t be the time series of interest, $\mathbf{x}_t$ be a vector containing the exogenous variables, and $\mathcal{F}_t = \{y_{t-1}, y_{t-2}, \ldots; \mathbf{x}_t, \mathbf{x}_{t-1}, \ldots\}$. The ACMx models postulate

$$y_t \mid \mathcal{F}_t \sim F(\cdot \mid \mu_t),$$

where $\mu_t = \mathrm{E}(y_t \mid \mathcal{F}_t) = \exp(\mathbf{x}_t'\boldsymbol{\beta})\lambda_t$, and λ_t follows the model

$$\lambda_t = \omega + \sum_{i=1}^{p} \alpha_i \left[\frac{y_{t-i}}{\exp(\mathbf{x}_{t-i}'\boldsymbol{\beta})} \right] + \sum_{j=1}^{q} \gamma_j \lambda_{t-j},$$

p and q are nonnegative integers, $\omega > 0$, and α_i and γ_j are parameters satisfying certain conditions so that λ_t is always positive and finite. The conditional distribution $F(y_t \mid \mathcal{F}_t)$ can be Poisson, negative binomial, or double Poisson (Tsay and Chen, 2018).

The estimation of ACMx models is implemented via the function ACMx with syntax:

```
R> ACMx(y, order = c(1, 1), X = NULL, cond.dist = "po", ini = NULL)
```

where y is the series of count data, X is the matrix of exogenous variables, order specifies the values for p and q, cond.dist determines the conditional distribution with options: "po" for Poisson, "nb" for negative binomial, and "dp" for double Poisson, and ini collects initial parameter estimates designed for use with "nb" or "dp".

We illustrate the function ACMx with an example below:

```
R> set.seed(12)
R> x <- rnorm(1000)*0.1
R> y <- matrix(0, 1000, 1)
R> y[1] <- 2
R> lambda <- matrix(0, 1000, 1)
R> for (i in 2:1000){
+    lambda[i] <- 2 + 0.2*y[i-1]/exp(x[i-1]) + 0.5*lambda[i-1]
+    set.seed(i)
+    y[i] <- rpois(1, exp(x[i]) * lambda[i])
+  }
R> ACMx(y, order = c(1, 1), x, "po")
Initial estimates:  1.056732 1.738874 0.05 0.5
loB:  -1.056732 1e-06 1e-06 1e-06
upB:  3.170195 19.12762 0.5 0.999999
Maximized log-likehood:  -2373.08

Coefficient(s):
       Estimate  Std. Error  t value   Pr(>|t|)
beta  1.0562836   0.1274853  8.28553 2.2204e-16 ***
omega 2.6696378   0.5569954  4.79293 1.6437e-06 ***
alpha 0.1579050   0.0265997  5.93634 2.9145e-09 ***
gamma 0.4427157   0.0913361  4.84711 1.2528e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Here a time series following the ACMx model with Poisson conditional distribution, order (1,1), $\beta = 1$, $\omega = 2$, $\alpha = 0.2$ and $\gamma = 0.5$ is generated. The R output reports the estimated coefficients which are all significant and close to their true values.

Functional time series

Functional time series analysis has received much attention since the pioneering work of Bosq (2000), and has been widely applied in many fields, including environmental science (Hormann and Kokoszka, 2010), social science (Hyndman and Shang, 2009), and finance (Diebold and Li, 2006; Horváth et al., 2013). Liu et al. (2016) proposed a new class of models called the CFAR models, which has an intuitive and direct interpretation of the dynamics of a stochastic process. The **NTS** encompasses functions to implement the method proposed by Liu et al. (2016).

Before presenting these R functions, we briefly introduce the CFAR model and its estimation procedure. A sequence of square integrable random functions $\{X_t \mid t = 1, \dots, T\}$ defined on $[0, 1]$ follows a CFAR model of order p if

$$X_t(s) = \sum_{i=1}^{p} \int_0^1 \phi_i(s-u) X_{t-i}(u) du + \epsilon_t(s), \quad s \in [0,1],$$

where $\phi_i(\cdot)$ are square integrable and defined on $[-1, 1]$ $(i = 1, \dots, p)$ and are called the convolutional coefficient functions, and ϵ_t are i.i.d. Ornstein-Uhlenbeck (O-U) processes defined on [0,1] satisfying the stochastic differential equation, $d\epsilon_t(s) = -\rho\epsilon_t(s)ds + \sigma dW_s$, $\rho > 0$, and W_s is a Wiener process.

In practice, $X_t(\cdot)$ is usually observed only at discrete points, $s_i = i/N$, $i = 0, \dots N$ for time $t = 1, \dots T$. Liu et al. (2016) recovers the function $X_t(\cdot)$ by linear interpolation,

$$\widetilde{X}_t(s) = \frac{(s_i - s)X_t(s_{i-1}) + (s - s_{i-1})X_t(s_i)}{1/N}, \text{ for } s_{i-1} \le s < s_i,$$

and approximates $\phi_i(\cdot)$ by cubic B-splines,

$$\phi_i(\cdot) \approx \widetilde{\phi}_i(\cdot) = \sum_{j=1}^{k} \beta_{k,i,j} B_{k,j}(\cdot), \text{ for } i = 1, \dots p,$$

where $\{B_{k,j}, j = 1, \dots, k\}$ are uniform cubic B-spline basis functions with k degrees of freedom.

With the above approximation, the B-spline coefficients $\boldsymbol{\beta} = \{\beta_{k,i,j}\}$, ρ, and σ^2 can be estimated by maximizing the approximated log-likelihood function. Specifically

$$(\hat{\boldsymbol{\beta}}, \hat{\rho}, \hat{\sigma}^2) = \arg\max Q(\boldsymbol{\beta}, \rho, \sigma^2),$$

where

$$Q(\boldsymbol{\beta}, \rho, \sigma^2) = C + \frac{(N+1)(T-p)}{2} \ln\left(\frac{\pi\sigma^2}{\rho}\right) - \frac{N(T-p)}{2} \ln(1 - e^{-2\rho/N}) - \frac{1}{2}\sum_{t=1}^{T} \mathbf{e}_t' \boldsymbol{\Sigma}^{-1} \mathbf{e}_t,$$

where C is a constant, $\mathbf{e}_t = (e_{t,0}, \dots e_{t,N})'$, $e_{t,\ell} = X_t(\ell/N) - \sum_{i=1}^{p}\sum_{j=1}^{k} \beta_{k,i,j} \int_0^1 B_{k,j}(\ell/N - u)\widetilde{X}_t(s)du$ for $\ell = 0, \dots, N$, and $\boldsymbol{\Sigma}$ is an $(N+1)$-by-$(N+1)$ matrix with $\sigma^2 e^{-\rho|i-j|/N}$ as its (i, j)-th entry.

The convolutional functions are estimated by

$$\hat{\phi}_i(\cdot) = \sum_{j=1}^{k} \hat{\beta}_{k,i,j} B_{k,j}(\cdot).$$

In the **NTS** package, model specification and estimation of a CFAR process can be carried out by the functions `F_test_cfar` and `est_cfar` with the syntax:

```
R> F_test_cfar(f, p.max = 6, df_b = 10, grid = 1000)
R> est_cfar(f, p = 3, df_b = 10, grid = 1000)
```

The observed functional time series is stored in `f`, which is a T-by-$(N+1)$ matrix, where T is the length of time, and $(N+1)$ is the number of discrete observations of the functional data at a given time period. `p` specifies the CFAR order. `df_b` determines the degrees of freedom k for the natural cubic splines, and `grid` is the number of grid points used to construct the functional time series and noise process.

The function `F_test_cfar` returns the test statistics and the p-values for a sequence of F-tests with $H_0 : \phi_k(\cdot) = 0, \phi_i(\cdot) \neq 0, i = 1 \dots, k-1$ versus $H_a : \phi_i(\cdot) \neq 0, i = 1, \dots, k$ for $k = 1, \dots,$ `p.max`. The function `est_cfar` returns the following values: `phi_coef` collects the estimated spline coefficients $\hat{\beta}_{k,i,j}$ for the convolutional function(s) which is a $(\texttt{df\_b} + 1) \times p$ matrix, and `phi_func` contains the estimated convolutional function values which is a $(2 \times \texttt{grid} + 1) \times p$ matrix. `rho` is the estimated ρ in the O-U process, and `sigma` is estimated standard deviation of the noise process, respectively.

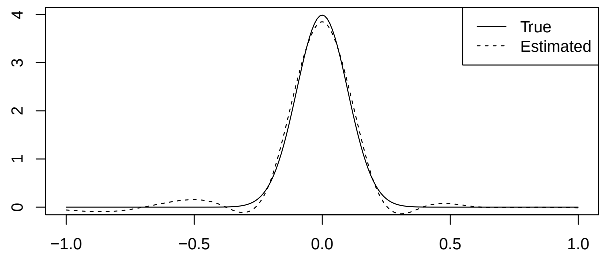


Figure 2: Plot of true convolution function $\phi(\cdot)$ and estimated convolution function $\hat{\phi}(\cdot)$ of the functional time series generated from Equation (8).

The function g_cfar in the **NTS** package generates a CFAR process with argument tmax being the length of time, rho is the parameter for the O-U process, sigma is the standard deviation of the O-U process, phi_list is the convolutional function(s), and ini is the burn-in period. It returns a list with two components. One is cfar, a tmax-by-(grid+1) matrix following the CFAR(p) model, and the other one epsilon is the innovation at time tmax. Function p_cfar provides the forecasts of a CFAR model with argument model as a result of an est_cfar fit and argument m as the forecasting horizon.

Let us consider a CFAR(1) process with the following convolutional coefficient function

$$\phi(s) = \frac{10}{\sqrt{2\pi}} e^{-50s^2}, \quad s \in [-1, 1], \tag{8}$$

where $\phi(\cdot)$ is the probability density function of a Gaussian random variable with mean 0 and standard deviation 0.1 truncated in the interval $[-1, 1]$. For the O-U process, $\rho = 5$ and its standard deviation is 1. The following R code simulates such a CFAR(1) process specified in Equation (8) with $N = 50$, $T = 1000$, and burn-in period 100, conducts an F test, performs the estimation procedure, and provides a one-step ahead prediction.

```
R> phi_func <- function(x){
+    return(dnorm(x, mean = 0, sd = 0.1))
+  }
R> t <- 1000; N <- 50
R> x <- g_cfar(t, rho = 5, phi_func, sigma = 1, ini = 100)
R> f <- x$cfar[, seq(1, 1001, 1000/N)]
R> F_test_cfar(f, p.max = 2, df_b = 10, grid = 1000)
Test and p-value of Order 0 vs Order 1:
[1] 1368.231    0.000
Test and  p-value of Order 1 vs Order 2 :
[1] 0.6848113 0.7544222
R> model <- est_cfar(f, p = 1, df_b = 10, grid = 1000)
R> print(c(model$rho, model$sigma))
[1] 4.940534 1.005315
R> pred <- p_cfar(model, f, m = 3)
```

From the output, the p-values for F tests suggest that we choose CFAR(1) model for the data. $\hat{\rho} = 4.941$ and the standard deviation of the noise process is estimated as 1.005, and they are both close to their true values. Figure 2 plots the estimated convolutional coefficient function (dashed line) and true function $\phi(\cdot)$ (solid line). It can be seen that est_cfar performs well.

F_test_cfarh and est_cfarh in **NTS** can deal with heteroscedasticity and irregular observation locations, while the existing R package **ftsa** designed for functional time series assumes that the functional data are observed at regular locations. The two functions come with following arguments: weight is the heteroscedasticity weight function of the noise process with grid+1 elements, num_obs is a t-by-1 vector and collects the numbers of observations at different times, and x_pos is a t-by$(N+1)$ matrix and shows the observation locations, where $(N+1)$ is the maximum number of observation at a time. The R code below will yield the same results as the previous one does. Hence, the output is omitted.

```
R> num_obs <- rep(N+1, t); x_pos <- matrix(rep(seq(0, 1, 1/N), each = t), t, N+1);
R> weight0 <- function(x){return(rep(1, length(x)))}
```

```
R> F_test_cfarh(f, weight0, p.max = 2, df_b = 10, grid = 1000, num_obs, x_pos)
R> modelh <- est_cfarh(f, weight0, p = 1, df_b = 10, grid = 1000, num_obs, x_pos)
```

State-space modelings via SMC methods

It is challenging to derive analytic solutions for filtering or smoothing of nonlinear or non-Gaussian state-space models. The SMC approach fully utilizes the dynamic nature of the model, and is an effective way to solve such complex problems (Tsay and Chen, 2018). Consider the state-space model:

$$\begin{aligned} \text{State equation:} x_t &= s_t(x_{t-1}, \epsilon_t) \qquad && \text{or } x_t \sim q_t(\cdot \mid x_{t-1}), \\ \text{Observation equation:} y_t &= h_t(x_t, e_t) && \text{or } y_t \sim f_t(\cdot \mid x_t), \end{aligned}$$

where x_t is the unobservable state variable and y_t is the observation ($t = 1, \ldots T$). The underlying states evolve through the known function $s_t(\cdot)$ and the state innovation ϵ_t, following a known conditional distribution $q_t(\cdot)$. The information on the underlying states is observed indirectly through y_t via the known function $h_t(\cdot)$ and with observational noise e_t. The function $h_t(\cdot)$ and the distribution of e_t are known with possibly unknown parameters to be estimated.

In general, there are four main statistical inference objectives associated with a state-space model:

1. Filtering: obtain the marginal posterior distribution of the current state x_t given the entire history of the observations up to the current time, that is, $p(x_t \mid y_1, \ldots y_t)$.
2. Prediction: obtain the marginal posterior distribution of the future state given the current available information, that is, $p(x_{t+1} \mid y_1, \ldots, y_t)$.
3. Smoothing: obtain the posterior distribution of the state at the time t given the entire available information, that is, $p(x_t \mid y_1, \ldots, y_T)$ for $t < T$.
4. Likelihood and parameter estimation. SMC uses a set of weighted samples $\{x_t^{(j)}, w_t^{(j)}\}$ to evaluate the likelihood function $L(\theta) = p(y_1, \ldots, y_T \mid \theta) = \int p(x_1, \ldots, x_T, y_1, \ldots, y_T \mid \theta) dx_1 \ldots dx_T$.

SMC is a recursive procedure with three components:

- **Propagation step**: At time t for $j = 1, \ldots, m$: draw $x_t^{(j)}$ from a trial distribution $g_t(x_t \mid \mathbf{x}_{t-1}^{(j)}, y_t)$, where m is the Monte Carlo sample size. Attach it to $\mathbf{x}_{t-1}^{(j)}$ to form $\mathbf{x}_t^{(j)} = (\mathbf{x}_{t-1}^{(j)}, x_t^{(j)})$. Compute the new weight for $x_t^{(j)}$.
- **Resampling step**: Sample a set of indices $\{I_1, \ldots, I_m\}$, where $I_k \in \{1, \ldots, m\}$ according to a set of priority scores $\alpha_t^{(j)}, j = 1, \ldots, m$. Replace the sample with $\{x_t^{(I_j)}, \tilde{w}_t^{(I_j)} = w_t^{(I_j)} / \alpha_t^{(I_j)}\}$.
- **Inference Step**: Estimation of $\mathrm{E}_{\pi_t}[(h(\mathbf{x}_t)]$ for some integrable function $h(\cdot)$ using the generated weighted samples $(\mathbf{x}_t^{(j)}, w_t^{(j)}), j = 1, \ldots, m$, where $\pi_t(\cdot)$ is the target distribution.

The selection of the propagation trial distribution $g_t(x_t \mid \mathbf{x}_{t-1}, y_t)$ plays a key role for an efficient implementation of SMC. Since the efficiency of SMC is determined by the variance of weight distribution, one would naturally want to choose $g_t(\cdot)$ so that the incremental weight is as close to a constant as possible. Liu and Chen (1995, 1998) proposed the trial distribution

$$g_t(x_t \mid \mathbf{x}_{t-1}) = p(x_t \mid \mathbf{x}_{t-1}, \mathbf{y}_t) \quad \propto \quad q_t(x_t \mid x_{t-1}) f_t(y_t \mid x_t).$$

The proposed distribution utilizes information from both the state and observation equations, hence is termed a full information propagation step.

Algorithm: Full information propagation step
At time t, for $j = 1, \ldots, m$:

1. Draw $x_t^{(j)}$ from the local posterior distribution,

$$g_t(x_t \mid x_{t-1}^{(j)}, y_t) = p(x_t \mid x_{t-1}^{(j)}, y_t) \propto f_t(y_t \mid x_t) q_t(x_t \mid x_{t-1}^{(j)}).$$

2. Compute the incremental weight

$$u_t^{(j)} = \int f_t(y_t \mid x_t) q_t(x_t \mid x_{t-1}^{(j)}) dx_t,$$

and the new weight $w_t^{(j)} = w_{t-1}^{(j)} u_t^{(j)}$.

Because using full information propagation allows for more efficient estimation procedure with Rao-Blackwellization, in **NTS** we provide a specific function for it, different from the more general function using any user designed propagation function.

It is shown that the variance of the weight distributions stochastically increases over time. Continuing to propagate the samples with small weights forward is a waste of computational resources since inference is based on weighted average of the samples. One solution is to use resampling schemes to duplicate the importance samples and to eliminate the ones with low weights. Hence resampling steps are essential in SMC implementations. Another issue to consider is how to make inference as efficient as possible. Rao-Blackwellization is one of the effective tools that can be used.

Smoothing is another important inference to make when we analyze data with state-space models. Delayed estimation (e. g. , making inference on $p(x_{t-d} \mid y_1, \ldots, y_t)$) is a special case of smoothing. It can be achieved simply by using the weighted sample $\{(x_{t-d}, w_t^{(j)})\}$. However, when d is large, this sampling approach does not work well because resampling reduces the number of unique ancestors and thus increases the estimation errors. A more efficient algorithm is to calculate the backward smoothing weights, after obtaining the forward filtering weighted samples $\{(x_t^{(j)}, w_t^{(j)}), j = 1, \ldots, m\}$. The resulting weighted sample $\{(x_t^{(j)}, \tilde{w}_t^{(j)}), j = 1, \ldots, m\}$ is properly weighted.

Algorithm: Weight marginalization SMC smoother

Let $\tilde{w}_T^{(j)} = w_T^{(j)}, j = 1, \ldots, m$. For $t = T-1, T-2, \ldots, 1$ and $j = 1, \ldots, m$:
Calculate

$$\tilde{u}_t^{(j)} = \sum_{i=1}^{m} \frac{q_{t+1}(x_{t+1}^{(i)} \mid x_t^{(j)})}{\sum_{k=1}^{m} q_{t+1}(x_{t+1}^{(i)} \mid x_t^{(k)}) w_t^{(k)}} \tilde{w}_{t+1}^{(i)},$$

and the smoothing weight $\tilde{w}_t^{(j)} = w_t^{(j)} \tilde{u}_t^{(j)}$.

Table 3 lists functions in **NTS** that implement the aforementioned SMC procedures. Compared to the existing R package **SMC** coming with one generic function, **NTS** provides various functions for statistical inference and are much more user-friendly.

Table 3: List of R functions about SMC in package **NTS**

Usage	Function	Description
Generic function	`SMC`	SMC method with delay but not using a full information propagation step
	`SMC.Smooth`	SMC smoothing method
	`SMC.Full`	SMC method using a full information propagation step
	`SMC.Full.RB`	SMC method using a full information propagation step with Rao-Blackwellization estimate and no delay

The `SMC` function can be called by:

```
R> SMC(Sstep, nobs, yy, mm, par, xx.init, xdim, ydim, resample.sch,
+    delay = 0, funH = identity)
```

The following arguments need to be specified for `SMC`.

- `Sstep`: A function that performs one step propagation using a proposal distribution. Its input variables include (`mm,xx,logww,yyy,par,xdim,ydim`), where `xx` and `logww` are the prior iteration samples and their corresponding log weights, and `yyy` is the observation at current time step. It returns a list that contains `xx` (the sample x_t) and `logww` (their corresponding log weights).
- `nobs`: the number of observations, T.
- `yy`: the observations with T columns and `ydim` rows.
- `mm`: the Monte Carlo sample size m.
- `par`: a list of parameter values to pass to `Sstep`.
- `xx.init`: the initial samples of x_0.
- `xdim,ydim`: the dimension of the state variable x_t and the observation y_t.
- `resample.sch`: a binary vector of length `nobs`, reflecting the resampling schedule. `resample.sch[i]=1` indicates resample should be carried out at step i.
- `delay`: the maximum delay lag for delayed wighting estimation. Default is zero.
- `funH`: a user supplied function $h(\cdot)$ for estimating $\mathrm{E}(h(x_t) \mid y_{t+d})$. Default is identify function for estimating the mean with no delay. The function should be able to take vector or matrix as input and operates on each element of the input.

The function returns `xhat`, an array with dimensions (`xdim,nobs,delay+1`) and the scaled log-likelihood value `loglike`. The functions `SMC.Smooth`, `SMC.Full` and `SMC.Full.RB` have similar inputs and outputs, except that `SMC.Smooth` needs another input function for the backward smoothing step and `funH` is a function for estimating $\mathrm{E}(h(x_t) \mid y_1, \ldots, y_T)$.

Here is an example demonstrating how to implement SMC methods using the **NTS** package. Passive sonar is often used in military surveillance systems to track a target and to reduce the chance to be detected by the target. Without using an active sonar, it collects the signals generated by the motion of the target, and thus only the direction (or bearing) of the target is observed with error. With multiple detectors, the location of the target can be identified (Peach, 1995; Kronhamn, 1998; Arulampalam et al., 2004; Tsay and Chen, 2018).

Suppose the target is moving in a two-dimensional plane and there are two stationary detectors located on the same plane at (η_{i1}, η_{i2}) $(i = 1, 2)$ corresponding to a Cartesian coordinate. At each time t the observations consist of two angles ϕ_{it} $(i = 1, 2)$ of the target related to the detectors with noise. Assume that the target is moving with random acceleration in both directions. We use d_{1t} and d_{2t} to denote the true locations at time t in x axis and y axis respectively, and s_{1t} and s_{2t} to denote the speed at time t in x axis and in y axis respectively. Let ΔT be the time duration between two consecutive observations and we assume that the target maintains a random but constant acceleration between two consecutive observations. ϵ_{1t} and ϵ_{2t} are the total acceleration within the period in x and y directions which are assumed to follow $N(0, q_1^2)$ and $N(0, q_2^2)$, respectively. The motion model is

$$\begin{aligned} d_{it} &= d_{it-1} + s_{t-1}\Delta T + 0.5\Delta T \epsilon_{it}, \quad \epsilon_{it} \sim N(0, q_i^2), \\ s_{it} &= s_{i,t-1} + \epsilon_{it}, \quad \text{for } i = 1, 2. \end{aligned}$$

The unobserved state variable is $x_t = (d_{1t}, d_{2t}, s_{1t}, s_{2t})'$. One observes only the directions of the target with observational errors. Assume $\Delta T = 1$, the system can be rewritten as

$$\text{State equation} \quad x_t = \mathbf{H}x_{t-1} + \mathbf{W}w_t, \tag{9}$$

$$\text{Observation equation} \quad \phi_{it} = \arctan\left(\frac{d_{2t} - \eta_{i2}}{d_{1t} - \eta_{i1}}\right) + e_{it}, \text{ for } i = 1, 2, \tag{10}$$

where

$$\mathbf{H} = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad \mathbf{W} = \begin{pmatrix} 0.5q_1 & 0 \\ 0 & 0.5q_2 \\ q_1 & 0 \\ 0 & q_2 \end{pmatrix},$$

$w_t = [w_{t1}\, w_{t2}]'$, $w_{it} \sim N(0, 1)$ and $e_{it} \sim N(0, r^2)$ for $i = 1, 2$. We restrict $\phi_{it} \in [-\pi/2, \pi/2]$. This problem is highly nonlinear.

In this example, the sensors are placed as $(\eta_{11}, \eta_{12}) = (0, 0)$ and $(\eta_{21}, \eta_{22}) = (20, 0)$, and the measurement errors e_{1t} and e_{2t} follow $N(0, 0.02^2)$. A target moves with an initial value $x_0 = (10, 0, 0.01, 0.01)$ and a random acceleration with variance $q_1^2 = q_2^2 = 0.03^2$ in both directions.

We use the following code to generate data.

```
R> simPassiveSonar <- function(nn = 300, q, r, W, V, s2, start, seed){
+    set.seed(seed)
+    x <- matrix(nrow = 4, ncol = nn)
+    y <- matrix(nrow = 2, ncol = nn)
+    for(ii in 1:nn){
+      if(ii == 1) x[, ii] <- start
+      if(ii > 1) x[, ii] <- H%*%x[, ii - 1] + W%*%rnorm(2)
+      y[1, ii] <- atan(x[2, ii]/x[1, ii])
+      y[2, ii] <- atan(x[2, ii]/(x[1, ii] - s2))
+      y[, ii] <- (y[, ii] + V%*%rnorm(2) + 0.5*pi)%%pi -0.5*pi
+    }
+    return(list(xx = x, yy = y, H = H, W = W, V = V))
+  }

R> s2 <- 20; nobs <- 300
R> q <- c(0.03, 0.03); r <- c(0.02, 0.02)
R> start <- c(10, 10, 0.01, 0.01)
R> H <- c(1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0,  0,1)
R> H <- matrix(H, ncol = 4, nrow = 4, byrow = T)
R> W <- c(0.5*q[1], 0, 0, 0.5*q[2], q[1], 0, 0, q[2])
R> W <- matrix(W, ncol = 2, nrow = 4, byrow = T)
R> V <- diag(r)
R> simu_out <- simPassiveSonar(nobs, q, r, W, V, s2, start, seed = 2000)
R> plot(simu_out$xx[1, ], simu_out$xx[2, ], xlab = 'x', ylab = 'y', type = "l")
```

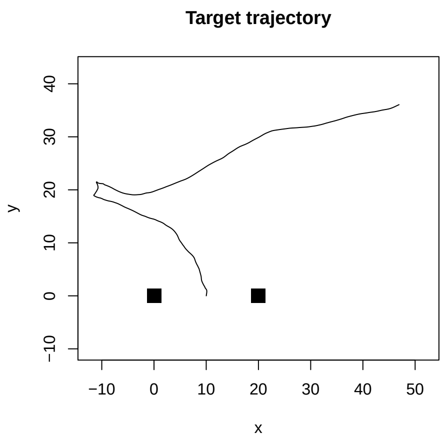


Figure 3: Simulated path of the SMC example generated from Equation (9) and Equation (10).

```
R> points(c(0, 20), c(0, 0), pch = 15, cex = 2)
```

Figure 3 shows a simulated trajectory for $t = 1, \ldots, 300$, which contains two sharp turns in the middle of the observational period.

We consider using the state equation as the proposal distribution. Specifically, given $x_{t-1}^{(i)}$, generate $x_t^{(j)}$ using (9). The incremental weight $u_t^{(j)} \propto p(y_t \mid x_t^{(j)}) = \sum_{i=0}^m p(y_t \mid x_t^{(j)})$ becomes

$$u_t^{(j)} \exp\left\{ -\frac{(\phi_{1t} - \hat{\phi}_{1t}^{(j)})^2 + (\phi_{2t} - \hat{\phi}_{2t}^{(j)})^2}{2r^2} \right\}.$$

The following statements are user generated functions for SMC implementation.

```
R> SISstep.Sonar <- function(mm, xx, logww, yy, par, xdim = 1, ydim = 1){
+    H <- par$H; W <- par$W; V <- par$V; s2 <- par$s2;
+    xx <- H%*%xx + W%*%matrix(rnorm(2*mm), nrow = 2, ncol = mm)
+    y1 <- atan(xx[2, ]/xx[1, ])
+    y2 <- atan(xx[2,]/(xx[1,] - s2))
+    res1 <- (yy[1] - y1 + 0.5*pi)%%pi - 0.5*pi
+    res2 <- (yy[2] - y2 + 0.5*pi)%%pi - 0.5*pi
+    uu <- -res1**2/2/V[1, 1]**2 - res2**2/2/V[2, 2]**2
+    logww <- logww + uu
+    return(list(xx = xx, logww = logww))
+  }

R> SISstep.Smooth.Sonar <- function(mm, xxt, xxt1, ww, vv, par){
+    H <- par$H; W <- par$W;
+    uu <- 1:mm
+    aa <- 1:mm
+    xxt1p <- H%*%xxt
+    for(i in 1:mm){
+      res1 <- (xxt1[1, i] - xxt1p[1, ])/W[1,1]
+      res2 <- (xxt1[2, i] - xxt1p[2, ])/W[2, 2]
+      aa[i] <- sum(exp(-0.5*res1**2 - 0.5*res2**2)*ww)
+    }
+    for(j in 1:mm){
+      res1 <- (xxt1[1, ] - xxt1p[1, j])/W[1, 1]
+      res2 <- (xxt1[2, ] - xxt1p[2, j])/W[2, 2]
+      uu[j] <- sum(exp(-0.5*res1**2 - 0.5*res2**2)*vv/aa)
+    }
+    vv <- ww*uu
+    return(list(vv = vv))
+ }
```

Now we are ready to run the Monte Carlo sample with size $m = 10000$.

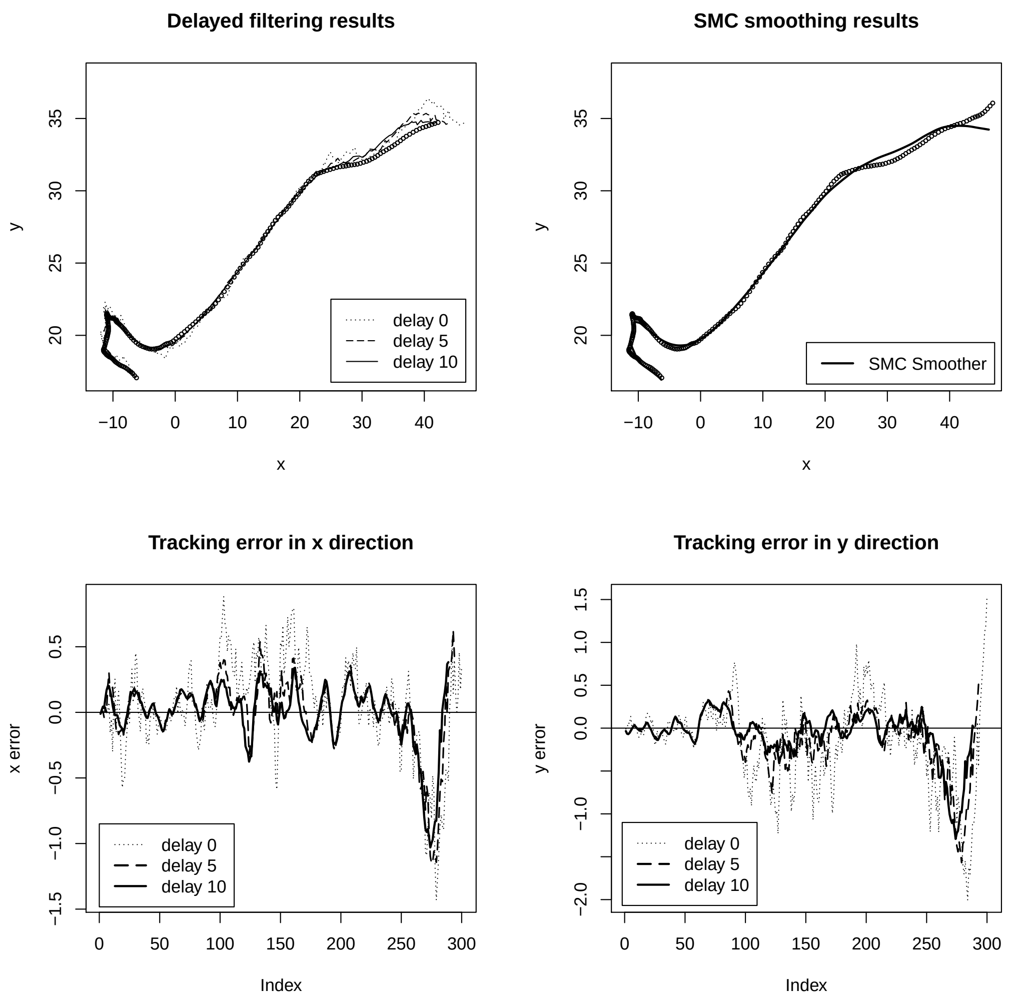


Figure 4: Delayed filtering and smoothing results for the SMC example generated from Equation (9) and Equation (10).

```
R> mm <- 100000
R> set.seed(1)
R> resample.sch <- rep(1,nobs)
R> xdim <- 4; ydim <- 2
R> mu0 <- start; SS0 <- diag(c(1, 1, 1, 1))*0.01
R> xx.init <- mu0 + SS0%*%matrix(rnorm(mm*4), nrow = 4, ncol = mm)
R> par <- list(H = H, W = W, V = V, s2 = s2)
R> delay <- 10
R> out <- SMC(SISstep.Sonar, nobs, yy, mm, par, xx.init, xdim, ydim, resample.sch, delay)
R> tt <- 100:nobs
R> plot(simu_out$xx[1, tt], simu_out$xx[2, tt], xlab = 'x', ylab = 'y')
R> for(dd in c(1, 6, 11)){
+      tt <- 100:(nobs - dd)
+      lines(out$xhat[1, tt, dd], out$xhat[2, tt, dd], lty = 23 - 2*dd, lwd = 1)
+    }
> legend(25, 22.5, legend = c("delay 0", "delay 5", "delay 10"), lty = c(21, 11, 1))
```

The top left panel in Figure 4 shows the delayed estimation using the delay weighting method with delay $d = 0, 5$, and 10, and Monte Carlo sample size $m = 10000$. The bottom panels in Figure 4 plot the estimation error in the x and y directions. The benefit of using delayed estimation can be clearly seen.

SMC smoothing can be implemented with the following R code. The top right panel in Figure 4 plots the smoothing results, and it shows that the SMC smoothing function performs very well.

```
R> set.seed(1)
```

```
R> mm <- 5000
R> par <- list(H = H, W = W, V = V, s2 = s2)
R> xx.init <- mu0 + SS0%*%matrix(rnorm(mm*4), nrow = 4, ncol = mm)
R> out.s5K <- SMC.Smooth(SISstep.Sonar, SISstep.Smooth.Sonar, nobs, yy, mm, par,
+      xx.init, xdim, ydim, resample.sch)
R> plot(simu_out$xx[1, tt], simu_out$xx[2, tt], xlab = 'x', ylab = 'y')
R> lines(out.s5K$xhat[1, tt], out.s5K$xhat[2, tt], lty = 1, lwd = 2)
+      legend(17, 19.5, legend = c("SMC Smoother"), lty = 1, lwd = 2)
```

Conclusion

The paper introduces the R package **NTS** which offers a broad collection of functions for the analysis of nonlinear time series data. We briefly review various nonlinear time series models, including TAR models, ACMx models, CFAR models, and state-space models. The associated estimation, identification, and forecasting procedures are discussed. The **NTS** package provides computational tools to fit these models, to evaluate their performance, and to provide predictions. Furthermore, the functions can be used, extended, and modified within the package to analyze larger univariate/multivariate, Gaussian/non-Gaussian time series. These features enable users to carry out a comprehensive and complex analysis of time series without the constraints from software availability.

Bibliography

D. Arulampalam, B. Ristic, N. Gordon, and T. Mansell. Bearings-only tracking of maneuvering targets using particle filters. *EURASIP Journal of Applied Signal Processing*, 2004:2351–2365, 2004. URL https://doi.org/10.1155/S1110865704405095. [p305]

O. Bjornstad. *nlts: NonlinearTime Series Analysis*, 2018. URL https://CRAN.R-project.org/package=nlts. R package version 1.0-2. [p293]

D. Bosq. *Linear Processes in Function Spaces, Theory and Applications*. Springer-Verlag, New York, 2000. URL http://doi.org/10.1007/978-1-4612-1154-9. [p294, 301]

A. Bowman and A. Azzalini. *sm: Smoothing Methods for Nonparametric Regression and Density Estimation*, 2018. URL https://CRAN.R-project.org/package=sm. R package version 2.2-5.6. [p293]

L. Breiman, A. Cutler, A. Liaw, and M. Wiener. *randomForest: Breiman and Cutler's Random Forests for Classification and Regression*, 2018. URL https://CRAN.R-project.org/package=randomForest. R package version 4.6-14. [p293]

R. Chen. Threshold variable selection in open-loop threshold autoregressive models. *Journal of Time Series Analysis*, 16:461–481, 1995. URL https://doi.org/10.1111/j.1467-9892.1995.tb00247.x. [p294]

R. Chen and R. Tsay. On the ergodicity of tar(1) processes. *The Annals of Applied Probability*, 1:613–634, 1991. URL http://doi.org/10.1214/aoap/1177005841. [p294]

F. X. Diebold and C. Li. Forecasting the term structure of government bond yields. *Journal of Econometrics*, 130:337–364, 2006. URL https://doi.org/10.1016/j.jeconom.2005.03.005. [p301]

D. Domian and D. Louton. A threshold autoregressive analysis of stock returns and real economic activity. *International Review of Economics and Finance*, 6:167–179, 1997. URL https://doi.org/10.1016/S1059-0560(97)90022-8. [p294]

C. Garcia and G. Sawitzki. *nonlinearTseries: Nonlinear Time Series Analysis*, 2020. URL https://CRAN.R-project.org/package=nonlinearTseries. R package version 0.2.10. [p293]

G. Goswami. *SMC: Sequential Monte Carlo (SMC) Algorithm*, 2011. URL https://CRAN.R-project.org/package=SMC. R package version 1.1. [p294]

S. Hormann and P. a. Kokoszka. Weakly dependent functional data. *The Annals of Statistics*, 38:1845–1884, 2010. URL http://doi.org/10.1214/09-aos768. [p301]

L. Horváth, P. Kokoszka, and R. Reeder. Estimation of the mean of functional time series and a two-sample problem. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 75:103–122, 2013. URL https://doi.org/10.1111/j.1467-9868.2012.01032.x. [p301]

R. Hyndman and H. L. Shang. *ftsa: Functional Time Series Analysis*, 2020. URL https://CRAN.R-project.org/package=ftsa. R package version 6.0. [p294]

R. J. Hyndman and H. L. Shang. Forecasting functional time series. *Journal of the Korean Statistical Society*, 38:199–211, 2009. URL https://doi.org/10.1016/j.jkss.2009.06.002. [p301]

T. Kronhamn. Bearings-only target motion analysis based on a multihypothesis kalman filter and adaptive ownship motion control. *IEE Proceedings- Radar, Sonar, and Navigation*, 145:247–252, 1998. URL http://doi.org/10.1049/ip-rsn:19982130. [p305]

D. Li and H. Tong. Nested sub-sample search algorithm for estimation of threshold models. *Statistica Sinica*, 26:1543–1554, 2016. URL http://doi.org/10.5705/ss.2013.394t. [p294, 295]

T. Liboschik, R. Fried, K. Fokianos, P. Probst, and J. Rathjens. *tscount: Analysis of Count Time series*, 2020. URL https://CRAN.R-project.org/package=tscount. R package version 1.4.3. [p293]

J. Liu and R. Chen. Blind deconvolution via sequential imputations. *Journal of the American Statistical Association*, 90(430):567–576, 1995. URL https://doi.org/10.2307/2291068. [p303]

J. Liu and R. Chen. Sequential monte carlo methods for dynamic systems. *Journal of the American Statistical Association*, 93:1032–1044, 1998. URL https://doi.org/10.2307/2669847. [p303]

X. Liu, R. Chen, and H. Xiao. Convolutional autoregressive models for functional time series. *Journal of Econometrics*, 194:263–282, 2016. URL https://doi.org/10.1016/j.jeconom.2016.05.006. [p294, 301]

P. Narayan. The behavior of us stock prices: Evidence from a threshold autoregressive model. *Mathematics and Computers in Simulation*, 71:103–108, 2006. URL https://doi.org/10.1016/j.matcom.2005.11.016. [p294]

A. F. D. Narzo, J. L. Aznarte, M. Stigler, and H. Tsung-wu. *tsDyn: Nonlinear Time series Models with Regime Switching*, 2020. URL https://CRAN.R-project.org/package=tsDyn. R package version 10-1.2. [p293]

N. Peach. Bearings-only tracking using a set of range-parameterized extended kalman filter. *IEE Proceedings- Control Theory and Applications*, 142:73–80, 1995. URL http://doi.org/10.1049/ip-cta:19951614. [p305]

B. Ripley. *tree: Classification and Regression Trees*, 2019. URL https://CRAN.R-project.org/package=tree. R package version 1.0-40. [p293]

G. Tiao and R. Tsay. Model specification in multivariate time series. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 51:157–213, 1989. URL http://doi.org/10.1111/j.2517-6161.1989.tb01756.x. [p294]

H. Tong. On a threshold model. *Pattern Recognition and Signal Processing*, 1978. Sihhoff& Noordhoof, Amsterdam. [p294]

H. Tong and K. Lim. Threshold autoregression, limit cycles and cyclical data. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 42(3):245–292, 1980. URL http://doi.org/10.1111/j.2517-6161.1980.tb01126.x. [p294]

R. Tsay. Testing and modeling threshold autoregressive process. *Journal of the American Statistical Association*, 84:231–240, 1989. URL https://doi.org/10.2307/2289868. [p293, 298]

R. Tsay and R. Chen. *Nonlinear Time Series Analysis*. John Wiley & Sons, New Jersey, 2018. ISBN 978-1-119-26407-1. URL http://doi.org/10.1002/9781119514312. [p300, 303, 305]

R. Tsay, R. Chen, and X. Liu. *NTS: Nonlinear Time Series Analysis*, 2020. URL https://CRAN.R-project.org/package=NTS. R package version 1.1.2. [p293]

H. Youssef. *NlinTS: Models for Non Linear Causality Detection in Time Series*, 2020. URL https://CRAN.R-project.org/package=NlinTS. R package version 1.4.4. [p293]

H. Zhang and F. Nieto. *TAR: Bayesian Modeling of Autoregressive Threshold Time Series Models*, 2017. URL https://CRAN.R-project.org/package=TAR. R package version 1.0. [p293]

Xialu Liu
Department of Management Information Systems
San Diego State University
5500 Campanile Drive, San Diego, CA 92182
USA
xialu.liu@sdsu.edu

Rong Chen
Department of Statistics
Rutgers University
57 US Highway 1, New Brunswick, NJ 08901
USA
rongchen@stat.rutgers.edu

Ruey Tsay
Booth School of Business
University of Chicago
5807 S. Woodlawn Ave, Chicago, IL 60637
USA
ruey.tsay@chicagobooth.edu

Chen's research is supported in part by National Science Foundation grants DMS-1503409, DMS-1737857, IIS-1741390 and CCF-1934924. Tsay's research is supported in part by the Booth School of Business, University of Chicago.

Species Distribution Modeling using Spatial Point Processes: a Case Study of Sloth Occurrence in Costa Rica

by Paula Moraga

Abstract

Species distribution models are widely used in ecology for conservation management of species and their environments. This paper demonstrates how to fit a log-Gaussian Cox process model to predict the intensity of sloth occurrence in Costa Rica, and assess the effect of climatic factors on spatial patterns using the **R-INLA** package. Species occurrence data are retrieved using **spocc**, and spatial climatic variables are obtained with **raster**. Spatial data and results are manipulated and visualized by means of several packages such as **raster** and **tmap**. This paper provides an accessible illustration of spatial point process modeling that can be used to analyze data that arise in a wide range of fields including ecology, epidemiology and the environment.

Introduction

Species distribution models are widely used in ecology to predict and understand spatial patterns, assess the influence of climatic and environmental factors on species occurrence, and identify rare and endangered species. These models are crucial for the development of appropriate strategies that help protect species and the environments where they live. In this paper, we demonstrate how to formulate spatial point processes for species distribution modeling and how to fit them with the **R-INLA** package (Rue et al., 2009) (`http://www.r-inla.org/`).

Point processes are stochastic models that describe locations of events of interest and possibly some additional information such as marks that inform about different types of events (Diggle, 2013; Moraga and Montes, 2011). These models can be used to identify patterns in the distribution of the observed locations, estimate the intensity of events (i.e., mean number of events per unit area), and learn about the correlation between the locations and spatial covariates. The simplest theoretical point process model is the homogeneous Poisson process. This process satisfies two conditions. First, the number of events in any region A follows a Poisson distribution with mean $\lambda|A|$, where λ is a constant value denoting the intensity and $|A|$ is the area of region A. And second, the number of events in disjoint regions are independent. Thus, if a point pattern arises as a realization of an homogeneous Poisson process, an event is equally likely to occur at any location within the study region, regardless of the locations of other events.

In many situations, the homogeneous Poisson process is too restrictive. A more interesting point process model is the log-Gaussian Cox process which is typically used to model phenomena that are environmentally driven (Diggle et al., 2013). A log-Gaussian Cox process is a Poisson process with a varying intensity which is itself a stochastic process of the form $\Lambda(s) = exp(Z(s))$ where $Z = \{Z(s) : s \in \mathbb{R}^2\}$ is a Gaussian process. Then, conditional on $\Lambda(s)$, the point process is a Poisson process with intensity $\Lambda(s)$. This implies that the number of events in any region A is Poisson distributed with mean $\int_A \Lambda(s)ds$, and the locations of events are an independent random sample from the distribution on A with probability density proportional to $\Lambda(s)$. The log-Gaussian Cox process model can also be easily extended to include spatial explanatory variables providing a flexible approach for describing and predicting a wide range of spatial phenomena.

In this paper, we formulate and fit a log-Gaussian Cox process model for sloth occurrence data in Costa Rica that incorporates spatial covariates that can influence the occurrence of sloths, as well as random effects to model unexplained variability. The model allows to estimate the intensity of the process that generates the data, understand the overall spatial distribution, and assess factors that can affect spatial patterns. This information can be used by decision-makers to develop and implement conservation management strategies.

The rest of the paper is organized as follows. First, we show how to retrieve sloth occurrence data using the **spocc** package (Chamberlain, 2018) and spatial climatic variables using the **raster** package (Hijmans, 2019). Then, we detail how to formulate the log-Gaussian Cox process and how to use **R-INLA** to fit the model. Then, we inspect the results and show how to obtain the estimates of the model parameters, and how to create create maps of the intensity of the predicted process. Finally, the conclusions are presented.

Sloth occurrence data

Sloths are tree-living mammals found in the tropical rain forests of Central and South America. They have an exceptionally low metabolic rate and are noted for slowness of movement. There are six sloth species in two families: two-toed and three-toed sloths. Here, we use the R package **spocc** (Chamberlain, 2018) to retrieve occurrence data of the three-toed brown-throated sloth in Costa Rica.

The **spocc** package provides functionality for retrieving and combining species occurrence data from many data sources such as the Global Biodiversity Information Facility (GBIF) (`https://www.gbif.org/`), and the Atlas of Living Australia (ALA) (`https://www.ala.org.au/`). We use the `occ()` function from **spocc** to retrieve the locations of brown-throated sloths in Costa Rica recorded between 2000 and 2019 from the GBIF database (GBIF.org, 2020; GBIF: The Global Biodiversity Information Facility, 2020). In the function, we specify arguments `query` with the species scientific name (*Bradypus variegatus*), `from` with the name of the database (GBIF), and `date` with the start and end dates (2000-01-01 to 2019-12-31). We also specify we wish to retrieve occurrences in Costa Rica by setting `gbifopts` to a named list with `country` equal to the 2-letter code of Costa Rica (CR). Moreover, we only retrieve occurrence data that have coordinates by setting `has_coords = TRUE`, and specify `limit` equal to 1000 to retrieve a maximum of 1000 occurrences.

```
library("spocc")
df <- occ(query = "Bradypus variegatus", from = "gbif",
          date = c("2000-01-01", "2019-12-31"),
          gbifopts = list(country = "CR"),
          has_coords = TRUE, limit = 1000)
```

`occ()` returns an object with slots for each of data sources. We can see the slot names by typing `names(df)`.

```
names(df)

## [1] "gbif" "bison" "inat" "ebird" "ecoengine" "vertnet" "idigbio" "obis" "ala"
```

In this case, since we only retrieve data from GBIF, the only slot with data is `df$gbif` while the others are empty. `df$gbif` contains information about the species occurrence and also other details about the retrieval process. We can use the `occ2df()` function to combine the output of `occ()` and create a single data frame with the most relevant information for our analysis, namely, the species name, the decimal degree longitude and latitude values, the data provider, and the dates and keys of the occurrence records.

```
d <- occ2df(df)
```

A summary of the data can be seen with `summary(d)`. We observe the data contain 707 locations of sloths occurred between 2000-01-24 and 2019-12-30.

```
summary(d)

##      name             longitude        latitude         prov
## Length:707         Min.   :-85.51   Min.   : 8.340   Length:707
## Class :character   1st Qu.:-84.15   1st Qu.: 9.391   Class :character
## Mode  :character   Median :-84.01   Median : 9.795   Mode  :character
##                    Mean   :-83.87   Mean   : 9.902
##                    3rd Qu.:-83.51   3rd Qu.:10.450
##                    Max.   :-82.62   Max.   :11.038
##       date               key
## Min.   :2000-01-24   Length:707
## 1st Qu.:2014-01-12   Class :character
## Median :2017-05-30   Mode  :character
## Mean   :2015-12-19
## 3rd Qu.:2019-01-18
## Max.   :2019-12-30
```

We can visualize the locations of sloths retrieved in Costa Rica using several mapping packages such as **tmap** (Tennekes, 2018), **ggplot2** (Wickham, 2016), **leaflet** (Cheng et al., 2018), and **mapview** (Appelhans et al., 2019). Here, we choose to create maps using **tmap**. First, we use the `SpatialPoints()` function from the **sp** package (Pebesma and Bivand, 2005) to create a `SpatialPoints` object called `dpts` with the coordinates of the sloth locations.

```
library(sp)
dpts <- SpatialPoints(d[, c("longitude", "latitude")])
```

Then we create the map plotting the locations of `dpts`. **tmap** allows to create both static and interactive maps by using `tmap_mode("plot")` and `tmap_mode("view")`, respectively. Here, we create an interactive map using use a basemap given by the OpenStreetmap provider, and plot the sloth locations with `tm_shape(dpts) + tm_dots()`.

```
library(tmap)
tmap_mode("view")
tm_basemap(leaflet::providers$OpenStreetMap) +
  tm_shape(dpts) + tm_dots()
```

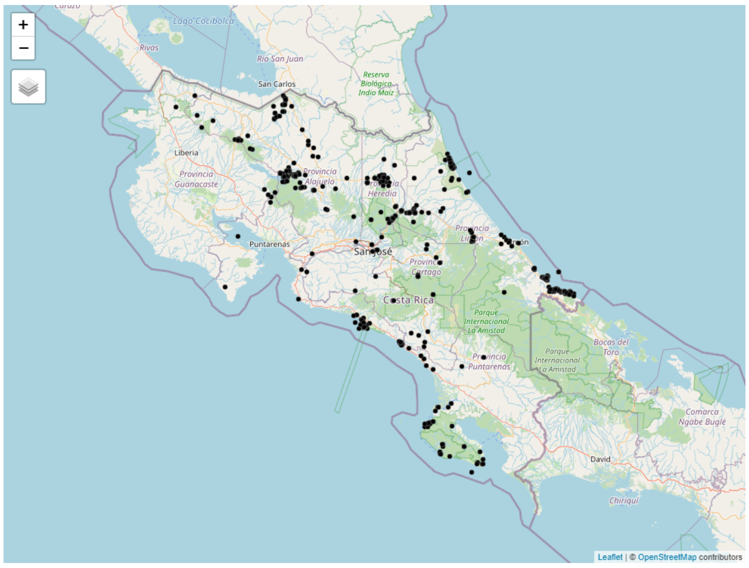


Figure 1: Snapshot of the interactive map depicting sloth locations in Costa Rica. The map shows some areas with no sloths and other areas with sloth aggregations.

The map created is shown in Figure 1. The map shows an inhomogeneous pattern of sloths with concentrations in several locations of Costa Rica. We will use a log-Gaussian Cox point process model to predict the intensity of the process that generates the sloth locations and assess the potential effect of climatic variables on the occurrence pattern.

Spatial climatic covariates

In the model, we include a spatial explanatory variable that can potentially affect sloth occurrence. Specifically, we include a variable that denotes annual minimum temperature observed in the study region. This variable can be obtained using the **raster** package (Hijmans, 2019) from the WorldClim database (`http://www.worldclim.org/bioclim`). We use the `getData()` function of the **raster** package by specifying the name of the database (`"worldclim"`), the variable name (`"tmin"`), and a resolution of 10 minutes of a degree (`"10"`). `getData()` returns a `RasterStack` with minimum temperature observations with degree Celsius x 10 units for each month. We average the values of the `RasterStack` and compute a raster that represents annual average minimum temperature.

```
library(raster)
rmonth <- getData(name = "worldclim", var = "tmin", res = 10)
rcov <- mean(rmonth)
```

Implementing and fitting the spatial point process model

Log-Gaussian Cox process model

We assume that the spatial point pattern of sloth locations in Costa Rica, $\{x_i : i = 1, \ldots, n\}$, has been generated as a realization of a log-Gaussian Cox process with intensity given by $\Lambda(s) = exp(\eta(s))$.

This model can be easily fitted by approximating it by a latent Gaussian model by means of a gridding approach (Illian et al., 2012). First, we discretize the study region into a grid with $n_1 \times n_2 = N$ cells $\{s_{ij}\}$, $i = 1, \ldots, n_1$, $j = 1, \ldots, n_2$. In the log-Gaussian Cox process, the mean number of events in cell s_{ij} is given by the integral of the intensity over the cell, $\Lambda_{ij} = \int_{s_{ij}} exp(\eta(s))ds$, and this integral can be approximated by $\Lambda_{ij} \approx |s_{ij}|exp(\eta_{ij})$, where $|s_{ij}|$ is the area of the cell s_{ij}. Then, conditional on the latent field η_{ij}, the observed number of locations in grid cell s_{ij}, y_{ij}, are independent and Poisson distributed as follows,

$$y_{ij}|\eta_{ij} \sim Poisson(|s_{ij}|exp(\eta_{ij})).$$

In our example, we model the log-intensity of the Poisson process as

$$\eta_{ij} = \beta_0 + \beta_1 \times cov(s_{ij}) + f_s(s_{ij}) + f_u(s_{ij}).$$

Here, β_0 is the intercept, $cov(s_{ij})$ is the covariate value at s_{ij}, and β_1 is the coefficient of $cov(s_{ij})$. `f_s()` is a spatially structured random effect reflecting unexplained variability that can be specified as a second-order two-dimensional CAR-model on a regular lattice. `f_u()` is an unstructured random effect reflecting independent variability in cell s_{ij}.

Computational grid

In order to fit the model, we create a regular grid that covers the region of Costa Rica. First, we obtain a map of Costa Rica using the `ne_countries()` function of the **rnaturalearth** package (South, 2017). In the function we set `type = "countries"`, `country = "Costa Rica"` and `scale = "medium"` (`scale` denotes the scale of map to return and possible options are small, medium and large).

```
library(rnaturalearth)
map <- ne_countries(type = "countries", country = "Costa Rica", scale = "medium")
```

Then, we create a raster that covers Costa Rica using `raster()` where we provide the map of Costa Rica and set `resolution = 0.1` to create cells with size of 0.1 decimal degrees. This creates a raster with 31 $\times$ 33 = 1023 cells, each having an area equal to 0.1^2 decimal degrees^2 (or 11.132 Km^2 at the equator).

```
resolution <- 0.1
r <- raster(map, resolution = resolution)
(nrow <- nrow(r))
## [1] 31
(ncol <- ncol(r))
## [1] 33
nrow*ncol
## [1] 1023
```

We initially set to 0 the values of all the raster cells by using `r[] <-0`. Then, we use `cellFromXY()` to obtain the number of sloths in each of the cells, and assign these counts to each of the cells of the raster.

```
r[] <- 0
tab <- table(cellFromXY(r, dpts))
r[as.numeric(names(tab))] <- tab
```

Finally, we convert the raster `r` to a `SpatialPolygonsDataFrame` object called `grid` using `rasterToPolygons()`. This grid will be used to fit the model with the **R-INLA** package.

```
grid <- rasterToPolygons(r)
```

Data

Now, we add to `grid` the data needed for modeling. Since the spatial model that will be used in **R-INLA** assumes data are sorted by columns, we first transpose `grid`. Then, we add variables `id` with the id of the cells, `Y` with the number of sloths, and `cellarea` with the cell areas.

```
grid <- grid[as.vector(t(matrix(1:nrow(grid), nrow = ncol, ncol = nrow))), ]

grid$id <- 1:nrow(grid)
grid$Y <- grid$layer
grid$cellarea <- resolution*resolution
```

We also add a variable cov with the value of the minimum temperature covariate in each of the cells obtained with the extract() function of **raster**.

```
grid$cov <- extract(rcov, coordinates(grid))
```

Finally, we delete the cells of grid that lie outside Costa Rica. First, we use raster::intersect() to know which cells lie within the map, and then subset these cells in the grid object.

```
gridmap <- raster::intersect(grid, map)
grid <- grid[grid$id %in% gridmap$id, ]
```

A summary of the data can be seen as follows,

```
summary(grid)

## Object of class SpatialPolygonsDataFrame
## Coordinates:
##          min       max
## x -85.908008 -82.60801
## y   8.089453  11.18945
## Is projected: FALSE
## proj4string :
## [+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0]
## Data attributes:
##      layer             id                Y            cellarea         cov
##  Min.   : 0.00   Min.   :   3.0   Min.   : 0.00   Min.   :0.01   Min.   : 78.0
##  1st Qu.: 0.00   1st Qu.: 291.5   1st Qu.: 0.00   1st Qu.:0.01   1st Qu.:176.7
##  Median : 0.00   Median : 566.0   Median : 0.00   Median :0.01   Median :202.4
##  Mean   : 1.38   Mean   : 533.1   Mean   : 1.38   Mean   :0.01   Mean   :189.0
##  3rd Qu.: 0.00   3rd Qu.: 762.5   3rd Qu.: 0.00   3rd Qu.:0.01   3rd Qu.:211.2
##  Max.   :95.00   Max.   :1009.0   Max.   :95.00   Max.   :0.01   Max.   :223.2
##                                                                  NA's   :2
```

We observe that the minimum temperature covariate has 2 missing values. We decide to impute these missing values with a simple approach where we set these values equal to the values of the cells next to them.

```
indNA <- which(is.na(grid$cov))
indNA

## [1]   6 220

grid$cov[indNA] <- grid$cov[indNA+1]
```

We use **tmap** to create maps of the number of sloths (Y) and the covariate values (cov). In the maps, we plot the border of grid that we obtain with the gUnaryUnion() function of the **rgeos** package (Bivand and Rundel, 2019).

```
library(rgeos)
gridborder <- gUnaryUnion(grid)
```

We use tm_facets(ncol = 2) to plot maps in the same row and two columns, and tm_legend() to put the legends in the left-bottom corner of the plots (Figure 2).

```
tmap_mode("plot")
tm_shape(grid) +
  tm_polygons(col = c("Y", "cov"), border.col = "transparent") +
  tm_shape(gridborder) + tm_borders() +
  tm_facets(ncol = 2) + tm_legend(legend.position = c("left", "bottom"))
```

Fitting the model using R-INLA

We fit the log-Gaussian Cox process model to the sloths data using the **R-INLA** package. This package implements the integrated nested Laplace approximation (INLA) approach that permits to perform approximate Bayesian inference in latent Gaussian models (Rue et al., 2009; Moraga., 2019). **R-INLA**

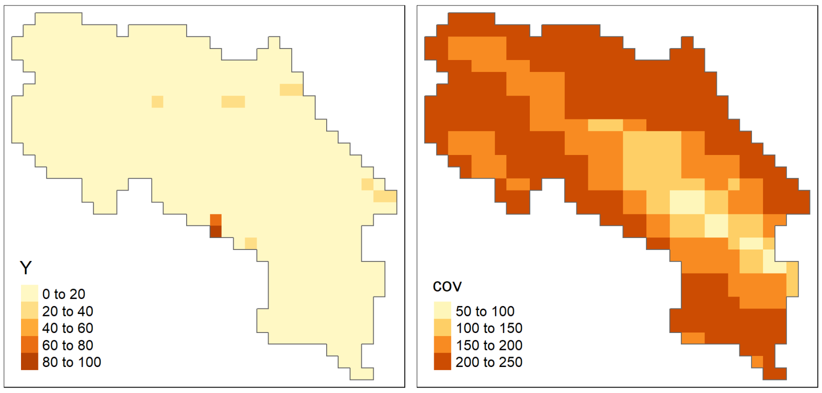


Figure 2: Maps with the number of sloths (left) and minimum temperature values (right) in Costa Rica. The intensity of sloths occurrence is modeled using minimum temperature as a covariate.

is not on CRAN because it uses some external C libraries that make difficult to build the binaries. Therefore, when installing the package, we need to specify the URL of the R-INLA repository. We also need to add the `https://cloud.r-project.org` repository to enable the installation of CRAN dependencies as follows,

```
install.packages("INLA", repos = c("https://inla.r-inla-download.org/R/stable",
                                   "https://cloud.r-project.org"), dep = TRUE)
```

Note that the **R-INLA** package is large and its installation may take a few minutes. Moreover, **R-INLA** suggests the **graph** and **Rgraphviz** packages which are part of the Bioconductor project. These packages have to be installed by using their tools, for example, by using `BiocManager::install(c("graph","Rgraphviz"),dep = TRUE))`.

To fit the model in INLA we need to specify a formula with the linear predictor, and then call the `inla()` function providing the formula, the family, the data, and other options. The formula is written by writing the outcome variable, then the $\sim$ symbol, and then the fixed and random effects separated by + symbols. By default, the formula includes an intercept. The outcome variable is `Y` (the number of occurrences in each cell) and the covariate is `cov`. The random effects are specified with the `f()` function where the first argument is an index vector specifying which elements of the random effect apply to each observation, and the other arguments are the model name and other options. In the formula, different random effects need to have different indices vectors. We use `grid$id` for the spatially structured effect, and create an index vector `grid$id2` with the same values as `grid$id` for the unstructured random effect. The spatially structured random effect is specified with the index vector `id`, the model name that corresponds to ICAR(2) (`"rw2d"`), and the number of rows (`nrow`) and columns (`ncol`) of the regular lattice. The unstructured random effect is specified with the index vector `id2` and the model name `"iid"`.

```
library(INLA)

grid$id2 <- grid$id

formula <- Y ~ 1 + cov +
  f(id, model="rw2d", nrow = nrow, ncol = ncol) +
  f(id2, model="iid")
```

Finally, we call `inla()` where we provide the formula, the family (`"poisson"`) and the data (`grid@data`). We write `E = cellarea` to denote that the expected values in each of the cells are in variable `cellarea` of the data. We also write `control.predictor = list(compute = TRUE)` to compute the marginal densities for the linear predictor.

```
res <- inla(formula, family = "poisson", data = grid@data,
            E = cellarea, control.predictor = list(compute = TRUE))
```

Results

The execution of inla() returns an object res that contains information about the fitted model including the posterior marginals of the parameters and the intensity values of the spatial process. We can see a summary of the results as follows,

```
summary(res)

## Fixed effects:
##               mean    sd 0.025quant 0.5quant 0.975quant   mode kld
## (Intercept) -1.904 2.426     -6.832   -1.852      2.734 -1.754   0
## cov          0.016 0.009     -0.002    0.016      0.035  0.016   0
##
## Random effects:
##   Name          Model
##     id Random walk 2D
##    id2 IID model
##
## Model hyperparameters:
##                    mean    sd 0.025quant 0.5quant 0.975quant  mode
## Precision for id  0.474 0.250      0.158    0.420       1.11 0.328
## Precision for id2 0.287 0.055      0.193    0.282       0.41 0.272
##
## Expected number of effective parameters(stdev): 196.11(7.06)
## Number of equivalent replicates : 2.61
##
## Marginal log-Likelihood:  -1620.60
## Posterior marginals for the linear predictor and the fitted values are computed
```

The intercept $\hat{\beta}_0 = -1.904$ with 95% credible interval $(-6.832, 2.734)$, the minimum temperature covariate has a positive effect on the intensity of the process with a posterior mean $\hat{\beta}_1 = 0.016$ and 95% credible interval $(-0.002, 0.035)$. We can plot the posterior distribution of the coefficient of the covariate $\hat{\beta}_1$ with **ggplot2** (Figure 3). First, we calculate a smoothing of the marginal distribution of the coefficient with inla.smarginal() and then call ggplot() specifying the data frame with the marginal values.

```
library(ggplot2)
marginal <- inla.smarginal(res$marginals.fixed$cov)
marginal <- data.frame(marginal)
ggplot(marginal, aes(x = x, y = y)) + geom_line() +
  labs(x = expression(beta[1]), y = "Density") +
  geom_vline(xintercept = 0, col = "black") + theme_bw()
```

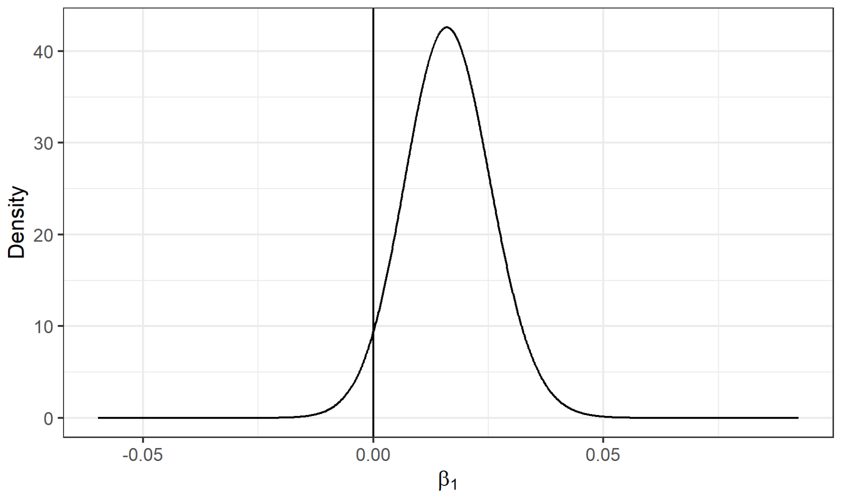


Figure 3: Posterior distribution of the coefficient of covariate minimum temperature and a vertical line at 0. The posterior mean is positive and the 95% credible interval includes 0.

The estimated spatially structured effect is in `res$summary.random$id`. This object contains 1023 elements that correspond to the number of cells in the regular lattice. We can add to the `grid` object the posterior mean of the spatial effect corresponding to each of the cells in Costa Rica as follows,

```
grid$respa <- res$summary.random$id[grid$id, "mean"]
```

We can also obtain the posterior mean of the unstructured random effect as follows,

```
grid$reiid <- res$summary.random$id2[, "mean"]
```

Then we can create maps of the random effects with **tmap**.

```
tm_shape(grid) +
  tm_polygons(col = c("respa", "reiid"), style = "cont", border.col = "transparent")  +
  tm_shape(gridborder) + tm_borders() +
  tm_facets(ncol = 2) + tm_legend(legend.position = c("left", "bottom"))
```

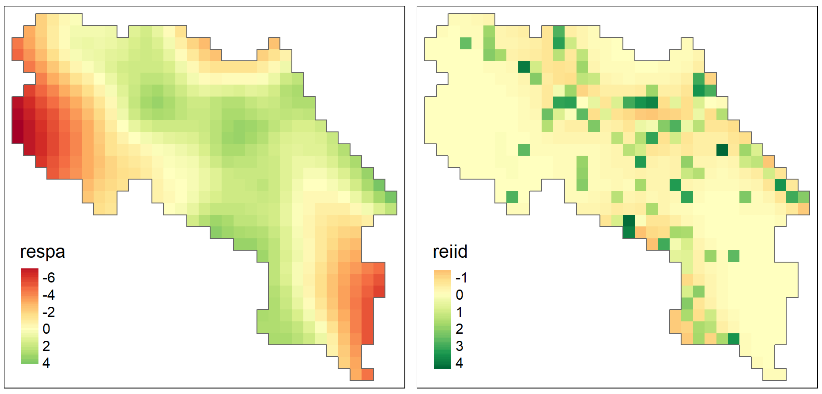


Figure 4: Maps with the values of the spatially structured (left) and unstructured (right) random effects. Maps show there is spatially structured and unstructured residual variation.

Figure 4 shows the maps of the spatially structured and unstructured random effects. We observe a non-constant pattern of the spatially structured random effect suggesting that the intensity of the process that generates the sloth locations may be affected by other spatial factors that have not been considered in the model. Morevoer, the unstructured random effect shows several locations with high values that modify the intensity of the process in individual cells independently from the rest.

The mean and quantiles of the predicted intensity (mean number of events per unit area) in each of the grid cells are in `res$summary.fitted.values`. In the object `grid`, we add a variable `NE` with the mean number of events of each cell by assigning the predicted intensity multiplied by the cell areas. We also add variables `LL` and `UL` with the lower and upper limits of 95% credible intervals for the number of events by assigning quantiles 0.025 and 0.975 multiplied by the cell areas.

```
cellarea <- resolution*resolution
grid$NE <- res$summary.fitted.values[, "mean"] * cellarea
grid$LL <- res$summary.fitted.values[, "0.025quant"] * cellarea
grid$UL <- res$summary.fitted.values[, "0.975quant"] * cellarea
```

We use **tmap** to create maps with the mean and lower and upper limits of 95% credible intervals for the number of sloths in each of the cells. We plot the three maps with a common legend that has breaks from 0 to the maximum number of cases in `grid$UL`.

```
tm_shape(grid) +
  tm_polygons(col = c("NE", "LL", "UL"),
              style = 'fixed', border.col = "transparent",
              breaks = c(0, 10, 50, 100, ceiling(max(grid$UL)))) +
  tm_shape(gridborder) + tm_borders() +
  tm_facets(ncol = 3) + tm_legend(legend.position = c("left", "bottom"))
```

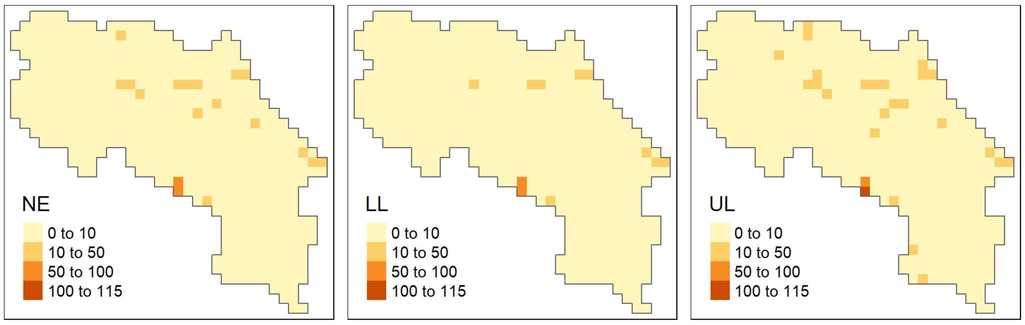


Figure 5: Maps with the predicted mean number of sloths (left), and lower (center) and upper (right) limits of 95% credible intervals. Maps show low intensity of sloth occurrence overall, and some specific locations with high intensity.

Maps created are shown in Figure 5. We observe that overall, the intensity of sloth occurrence is low, with less than 10 sloths in each of the cells. We also see there are some locations of high sloth intensity in the west and east coasts and the north of Costa Rica. The maps with the lower and upper limits of 95% credible intervals denote the uncertainty of these predictions. The maps created inform about the spatial patterns in the period where the data were collected. In addition, maps of the sloth numbers over time can also be produced using spatio-temporal point process models and this would help understand spatio-temporal patterns. The modeling results can be useful for decision-makers to identify areas of interest for conservation management strategies.

Summary

Species distribution models are widely used in ecology for conservation management of species and their environments. In this paper, we have described how to develop and fit a log-Gaussian Cox process model using the **R-INLA** package to predict the intensity of species occurrence, and assess the effect of spatial explanatory variables. We have illustrated the modeling approach using sloth occurrence data in Costa Rica retrieved from the Global Biodiversity Information Facility database (GBIF) using **spocc**, and a spatial climatic variable obtained with **raster**. We have also shown how to examine and interpret the results including the estimates of the parameters and the intensity of the process, and how to create maps of variables of interest using **tmap**.

Statistical packages such as Stan (Carpenter et al., 2017) or JAGS (Plummer, 2003) could have been used instead of **R-INLA** to fit our data. However, these packages use Markov chain Monte Carlo (MCMC) algorithms and may be high computationally demanding and become infeasable in large spatial data problems. In contrast, INLA produces faster inferences which allows us to fit large spatial datasets and explore alternative models.

The objective of this paper is to illustrate how to analyze species occurrence data using spatial point process models and cutting-edge statistical techniques in R. Therefore, we have ignored the data collection methods and have assumed that the spatial pattern analyzed is a realization of the true underlying process that generates the data. In real investigations, however, it is important to understand the sampling mechanisms, and assess potential biases in the data such as overrepresentation of certain areas that can invalidate inferences. Ideally, we would analyze data that have been obtained using well-defined sampling schemes. Alternatively, we would need to develop models that adjust for biases in the data to produce meaningful results (Giraud et al., 2015; Dorazio, 2014; Fithian et al., 2015). Moreover, expert knowledge is crucial to be able to develop appropriate models that include important predictive covariates and random effects that account for different types of variability.

To conclude, this paper provides an accessible illustration of spatial point process models and computational approaches that can help non-statisticians analyze spatial point patterns using R. We have shown how to use these approaches in the context of species distribution modeling, but they are also useful to analyze spatial data that arise in many other fields such as epidemiology and the environment.

Bibliography

T. Appelhans, F. Detsch, C. Reudenbach, and S. Woellauer. *mapview: Interactive Viewing of Spatial Data in R*, 2019. URL https://CRAN.R-project.org/package=mapview. R package version 2.7.0. [p312]

R. Bivand and C. Rundel. *rgeos: Interface to Geometry Engine - Open Source ('GEOS')*, 2019. URL https://CRAN.R-project.org/package=rgeos. R package version 0.4-3. [p315]

B. Carpenter, A. Gelman, M. D. Hoffman, D. Lee, B. Goodrich, M. Betancourt, M. Brubaker, J. Guo, P. Li, and A. Riddell. Stan: A probabilistic programming language. *Journal of Statistical Software*, 76 (1), 2017. doi: https://doi.org/10.18637/jss.v076.i01. [p319]

S. Chamberlain. *spocc: Interface to Species Occurrence Data Sources*, 2018. URL https://CRAN.R-project.org/package=spocc. R package version 0.9.0. [p311, 312]

J. Cheng, B. Karambelkar, and Y. Xie. *leaflet: Create Interactive Web Maps with the JavaScript 'Leaflet' Library*, 2018. URL https://CRAN.R-project.org/package=leaflet. R package version 2.0.2. [p312]

P. J. Diggle. *Statistical Analysis of Spatial and Spatio-Temporal Point Patterns*. Chapman & Hall/CRC, 2013. [p311]

P. J. Diggle, P. Moraga, B. Rowlingson, and B. M. Taylor. Spatial and Spatio-Temporal Log-Gaussian Cox Processes: Extending the Geostatistical Paradigm. *Statistical Science*, 28(4):542–563, 2013. URL https://doi.org/10.1214/13-STS441. [p311]

R. M. Dorazio. Accounting for imperfect detection and survey bias in statistical analysis of presence-only data. *Global Ecology and Biogeography*, 23, 2014. doi: https://doi.org/10.1111/geb.12216. [p319]

W. Fithian, J. Elith, T. Hastie, and D. A. Keith. Bias correction in species distribution models: pooling survey and collection data for multiple species. *Methods in Ecology and Evolution*, 6(4):428–438, 2015. doi: https://doi.org/10.1111/2041-210X.12242. [p319]

GBIF: The Global Biodiversity Information Facility. What is GBIF?, 2020. URL https://www.gbif.org/what-is-gbif. Accessed on 2 October 2020. [p312]

GBIF.org. GBIF Home Page, 2020. URL https://www.gbif.org. Accessed on 2 October 2020. [p312]

C. Giraud, C. Calenge, C. Coron, and R. Julliard. Capitalizing on opportunistic data for monitoring relative abundances of species. *Biometrics*, 72, 2015. doi: https://doi.org/10.1111/biom.12431. [p319]

R. J. Hijmans. *raster: Geographic Data Analysis and Modeling*, 2019. URL https://CRAN.R-project.org/package=raster. R package version 2.9-5. [p311, 313]

J. B. Illian, S. H. Sorbye, H. Rue, and D. Hendrichsen. Using INLA To Fit A Complex Point Process Model With Temporally Varying Effects - A Case Study. *Journal of Environmental Statistics*, 3, 2012. [p314]

P. Moraga. *Geospatial Health Data: Modeling and Visualization with R-INLA and Shiny*. Chapman & Hall/CRC Biostatistics Series, 2019. [p315]

P. Moraga and F. Montes. Detection of spatial disease clusters with LISA functions. *Statistics in Medicine*, 30:1057–1071, 2011. URL https://doi.org/10.1002/sim.4160. [p311]

E. J. Pebesma and R. S. Bivand. Classes and methods for spatial data in R. *R News*, 5, 2005. URL https://cran.r-project.org/doc/Rnews/. [p312]

M. Plummer. JAGS: A Program for Analysis of Bayesian Graphical Models Using Gibbs Sampling. In *Proceedings of the 3rd International Workshop on Distributed Statistical Computing (DSC 2003)*, 2003. [p319]

H. Rue, S. Martino, and N. Chopin. Approximate Bayesian Inference for Latent Gaussian Models Using Integrated Nested Laplace Approximations (with discussion). *Journal of the Royal Statistical Society B*, 71:319–392, 2009. URL https://doi.org/10.1111/j.1467-9868.2008.00700.x. [p311, 315]

A. South. *rnaturalearth: World Map Data from Natural Earth*, 2017. URL https://CRAN.R-project.org/package=rnaturalearth. R package version 0.1.0. [p314]

M. Tennekes. tmap: Thematic Maps in R. *Journal of Statistical Software*, 84(6):1–39, 2018. doi: 10.18637/jss.v084.i06. [p312]

H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. ISBN 978-3-319-24277-4. URL https://ggplot2.tidyverse.org. [p312]

Paula Moraga
Computer, Electrical and Mathematical Sciences and Engineering Division
King Abdullah University of Science and Technology (KAUST)
Thuwal, 23955-6900
Saudi Arabia
ORCiD: 0000-0001-5266-0201
Webpage: http://www.paulamoraga.com/
paula.moraga@kaust.edu.sa

A Graphical EDA Tool with ggplot2: brinton

by Pere Millán-Martínez, Ramon Oller

Abstract We present **brinton** package, which we developed for graphical exploratory data analysis in R. Based on **ggplot2**, **gridExtra** and **rmarkdown**, **brinton** package introduces `wideplot()` graphics for exploring the structure of a dataset through a grid of variables and graphic types. It also introduces `longplot()` graphics, which present the entire catalog of available graphics for representing a particular variable using a grid of graphic types and variations on these types. Finally, it introduces the `plotup()` function, which complements the previous two functions in that it presents a particular graphic for a specific variable of a dataset. This set of functions is useful for understanding the structure of a data set, discovering unexpected properties in the data, evaluating different graphic representations of these properties, and selecting a particular graphic for display on the screen.

Introduction

In 1977, J.W. Tukey noted that "The greatest value of a picture is when it *forces* us to notice what we never expected to see" (Tukey, 1977, p.iv). This statement aligns with expectation disconfirmation theory (Oliver, 1977), which links consumers' satisfaction to their expectations. The field of exploratory data analysis (EDA) is characterized precisely by not requiring an expectation, since in this approach hypotheses may not be pre-established. Rather, they are allowed to emerge through the observation of the data. Additionally, because we cannot automate the processes of defining a problem or the corresponding hypotheses —as signaled by J. Bertin that same year (Bertin, 1977, p.2)—, we face the challenge of automating graphical representations so that users can examine the data, develop hypotheses and then select the appropriate statistical graphic that will enable them to satisfy their recently created expectations.

The tools for generating graphics and statistics for a dataset automatically are called automated exploratory data analysis or autoEDA (Staniak and Biecek, 2019). These tools facilitate some of the characteristic tasks of EDA, such as describing variables and validating observations or the relationships established between the values of one or more variables. **brinton**, a new package we have developed for use within R, shows only graphics, leading us to classify it as a tool for automated graphical exploratory data analysis or autoGEDA. We can include in this category tools such as **GGobi** (Cook et al., 2007) and **Mondrian** (Theus and Urbanek, 2008). These tools differ from **brinton** in that they use interactive techniques extensively, and therefore are usually classified as visual analytics.

Multiple strategies exist for automating statistical diagramatic representations. Millán-Martínez and Valero-Mora (2018) differentiate strategies according to whether they are based on the characteristics of the data (functional design, Kamps (1999)), on the habits of a group of users (collaborative filtering), on those of a single user (content-based filtering), on the tasks that the user is meant to perform (task design), on the characteristics of human perception (perceptual design), on the limitations of the communication channel or the screen on which the graphics are projected (responsive design), or, finally, on the selection of characteristics of the desired graphics or models of representation (representation model design or deterministic design).

The statistical programming environment R has two graphics systems (Friendly, 2018). One is the standard graphics system of the package **graphics** with low-level functions, such as `lines()`, `points()`, `legend()` (which define concrete elements of a graphic) and high-level functions, such as `plot()`, `pie()`, and `barplot()` (which present a complete graphic). The other graphics system is based on the **grid** package, with low-level functions such as those of the **gridExtra** package (Auguie, 2017) and high-level ones such as those of the packages **lattice** (Sarkar, 2008) and **ggplot2** (Wickham, 2016), which produce complete graphics. Both in **graphics** and **grid** we find examples of the strategies mentioned above. We find functional design, for example, in the `plot()` function. If we apply it to the dataset `cars`, it produces a scatterplot, because it contains two numerical variables and is of the `data.frame` class. If we apply it to the dataset `airmiles`, it produces a line graph because this dataset has a single numerical variable and is of the `ts` class. We find task design in multiple packages, for example **survminer** (Therneau, 2015), which includes the function `ggsurvplot()` to generate graphics specifically for survival analysis. We find representation model design in basic functions such as `barplot()`, which produces a bar graph; `hist()`, which produces a histogram; and `pie()`, which produces a pie chart. We also see lower-level functions, such as the `geom_point()` function of **ggplot2**, which reduces the graphic to a kind of point plot. We can also find in **ggplot2** examples of perceptual design in decisions such as the default size, shape and color of the points, the grid lines and the panel background.

Despite all of the solutions already implemented in R, we are lacking an approach based on functional design that uses higher-level functions to show systematically not only a complete graphic but also a wide range of available graphics using the same data. Examining multiple graphics could lead the user to raise questions, which he or she could then answer using the presented graphics, new more specific graphics, or a particular graphic that could be adapted as needed (deterministic design). **brinton** package is our proposal for filling this gap in the R programming environment. We have named it after Willard Cope Brinton, whose Graphic Presentation (Brinton, 1939) solved a similar problem for physical libraries.

The article is organized as follows: Section 2 briefly reviews the autoGEDA packages within R and also the variants of multipanel graphics. Section 3 presents the three functions of **brinton** package and the available graphic types in the specimen. Section 4 details the graphical degrees of freedom that this package enjoys in the moment of expanding the specimen. Section 5 describes the situations in which the functions are useful and Section 6 offers our conclusions and outline future work.

AutoGEDA and multipanel graphics

We classify **brinton** package within the autoGEDA tools we have described above. Another essential feature of this package is that it extensively combines different graphic types referring to the same records and variables in the form of multipanel graphics. A range of autoGEDA tools exist both outside and inside R. For the purposes of contextualizing **brinton**, we will concentrate on the solutions based in R.

The landscape of autoGEDA in R

Among the R packages dedicated to autoEDA (Staniak and Biecek, 2019) only a few have a graphic orientation. We classify these packages according to their graphic solutions (although packages can have functions that offer different solutions).

Packages such as **tabplot** (Tennekes et al., 2013), **visdat** (Tierney, 2017) and **inspectdf** (Rushworth, 2019) use the structure plot, a graphic type that compacts all of the values of a dataset into a single panel. More specifically, **tabplot** and **visdat** essentially offer variants of tableplots, which are static versions of the table lens (Rao and Card, 1994), while **inspectdf** presents spine plots or bar charts, according to the type of summary to which the function `show_plot()` is applied. Another set of packages groups the variables of a dataset by type and represents the distribution of each variable in the cell of a multipanel graphic. This is the basic orientation of the packages **xray** (Seibelt, 2017), **DataExplorer** (Cui, 2019) and **SmartEDA** (Dayanand Ubrangala et al., 2019).

The packages **dataMaid** (Petersen and Ekstrøm, 2019) and **summarytools** (Comtois, 2019) offer another way to observe all variables. These packages have functions that produce a descriptive summary of the variables along with a histogram or bar graph, depending on the type of variable. We also find packages with miscellaneous functions, each of which is aimed at facilitating the generation of an adhoc graphic type. This is the case, for example, of **ExPanDaR**, **dlookr**, **summarytools** and **explore**.

AutoEDA packages tend to have a double presentation of results: tabulated and graphical. Some of them, such as **dataMaid**, **summarytools** and **SmartEDA**, make it possible to generate automatic reports and even adapt these reports to the needs of a particular user. Despite the utility of the packages described here, they tend to offer few options for graphic presentation beyond the most widely used graphics. The relationships between the values of the variables can be revealed much more easily if multiple graphic types are presented. These packages lack a wider range of graphic alternatives.

Multipanel graphics

There are different types of multipanel graphics depending on the diversity of graphic types and the origin of the data. On one hand, we have dashboards, which generally combine different graphic types in a limited space. Dashboards can draw from different data sources and are particularly useful for monitoring complex processes. Graphics of this type are implemented in R through packages such as **shinydashboard** (Chang and Borges Ribeiro, 2018) and **flexdashboard** (Iannone et al., 2018). The `plot_grid()` function of the package **cowplot** (Wilke, 2019) offers the possibility of combining graphics of the same or different type without space restrictions by creating multipanel graphics.

This can also be achieved with the **patchwork** package (Pedersen, 2019) that adds versatility to the composition of multipanel graphics by introducing operators that partition the canvas.

A second type of multipanel graphic is the conditioning plot[1]. In these, the same graphic type is repeated in different panels at the same scale, representing subsets of data according to the level of one or more variables. A third type of multipanel graphic is the matrix of plots, which links pairs of variables of the same type and from the same dataset. A classic example is the scatterplot matrix (Hartigan, 1975), or, more recently, the HE plot (Friendly, 2007). The diagonal of these grids can be populated with a different graphic type, since a single variable is involved. A variant of the matrix of plots uses source variables of different types that, when paired, result in a grid with multiple graphic types, depending on how the variables are combined. This graphic type is known as a generalized pairs plot (Emerson et al., 2013).

The brinton package

We created **brinton** package to facilitate exploratory data analysis following the visual information-seeking mantra (Shneiderman, 1996): "Overview first, zoom and filter, then details on demand." The main idea is to assist the user during these three phases through three functions: `wideplot()`, `longplot()` and `plotup()`. A distinctive feature is the following: the `wideplot()` function provides a limited selection of available graphics for all the variables in a data frame, the `longplot()` function provides all the range of available graphics for a limited selection of variables and, finally, the `plotup()` function provides one single graphic for a limited selection of variables. While each of these functions has its own arguments and purpose, all three serve to facilitate exploratory data analysis and the selection of a suitable graphic.

The `wideplot()` function allows the user to explore a dataset as a whole using a grid of graphics in which each variable is represented through multiple graphics. Once we have explored the dataset as a whole, the `longplot()` allows us to explore other graphics for a given variable. This function also presents a grid of graphics, but instead of showing a selection of graphics for each variable, it presents the full range of graphics available in the package to represent a single variable. Once we have narrowed in on a certain graphic, we can use the `plotup()` function, which presents the values of a variable on a single graphic. We can access the code of the resulting graphic and adapt it as needed. These three functions expand the graphic types that are presented automatically by the autoGEDA packages in the R environment.

brinton package is based primarily in the grammar of graphics (Wilkinson, 2005) implemented in R by the package **ggplot2**. Additionally, it draws on the package **gridExtra** (Auguie, 2017) for creating multipanel graphics and on **rmarkdown** (Allaire et al., 2019) for dynamically composing the results.

In the context of graphics packages in R based on the **grid** system, the package **lattice** allows the user to create a range of some 13 graphic types, which can be adapted to a very fine level of detail. **ggplot2** makes it possible to control even the finest detail of a graphic, but this comes at the price of learning its grammar and its layer system. In contrast, **brinton** package makes it possible for the user to select statistical graphics by name from a wide range of available graphics and, if he or she knows the grammar of **ggplot2**, adapt them as needed. To create a statistical graphic in R, if the desired graphic is already implemented in **brinton** package, the user must simply specify the data source and the graphic type to be produced.

The package can be installed easily from the Comprehensive R Archive Network (CRAN) using the R console. When the package is loaded into memory, it provides a startup message that pays homage to Henry D. Hubbard's enthusiastic introduction to the book Graphic Presentation (Brinton, 1939):

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

```
M a G i C i N G R a P H S
```

The wideplot function

When a dataset is loaded into R, the next function to be used tends to be `str()`. This occurs because if we don't determine the nature of the values explicitly, the functions for loading datasets make

[1]The terminology for conditioning plots is not unanimous. These plots were first described by J. Bertin as *séries homogènes* (Bertin, 1967, p.26). Later, E. Tufte introduced them as small multiples (Tufte, 1983). W.S. Cleveland called them juxtaposed panels (Cleveland, 1985, p.200) and also trellis graphics (Becker et al., 1996). In the R environment they are generally known as lattice graphics (Sarkar, 2008) or facet plots, based on the description of this technique by L. Wilkinson (2005) and later implemented in **ggplot2**

assumptions about it. The function `str()` shows in the console the type of object to which the function is being applied, the number of rows, the number and names of columns, their class (number, factor, etc.) and the initial observations for each variable. The `wideplot()` function takes inspiration from this function, but instead of describing the dataset in textual or tabular form, it does it graphically. We can easily compare the results of these two functions, for example, with the dataset `esoph` from a case-control study of esophageal cancer in Ille-et-Vilaine, France. The dataset has three ordered factor-type variables and two numerical variables:

```
  str(esoph)

#> 'data.frame':    88 obs. of  5 variables:
#>  $ agegp    : Ord.factor w/ 6 levels "25-34"<"35-44"<..: 1 1 1 1 1 1 1 1 1 1 ...
#>  $ alcgp    : Ord.factor w/ 4 levels "0-39g/day"<"40-79"<..: 1 1 1 1 2 2 2 2 3 3 ...
#>  $ tobgp    : Ord.factor w/ 4 levels "0-9g/day"<"10-19"<..: 1 2 3 4 1 2 3 4 1 2 ...
#>  $ ncases   : num  0 0 0 0 0 0 0 0 0 0 ...
#>  $ ncontrols: num  40 10 6 5 27 7 4 7 2 1 ...
```

```
  wideplot(data = esoph)
```

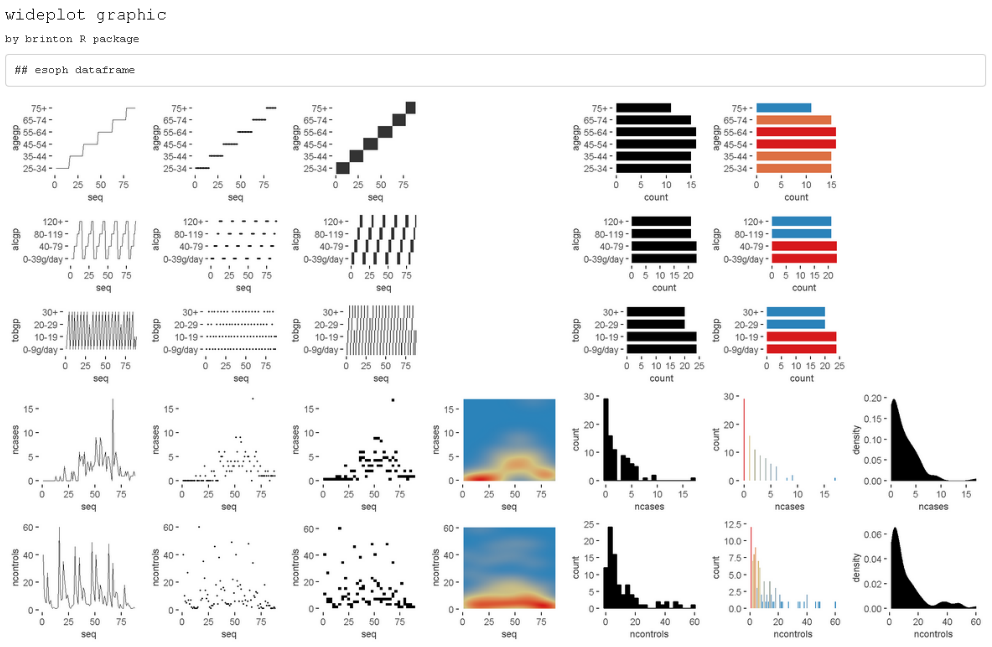


Figure 1: Output of `wideplot(esoph)`. A grid of graphics in which each row corresponds to a variable in the dataset `esoph` and each column displays different available graphics.

The `wideplot()` function creates html files, as side-effects, with a graphical summary (See Figure 1) of the variables included in the dataset to which it has been applied. First it groups the variables according to the following sequence: logical, ordered, factor, character, datetime and numeric. Next, it creates a multipanel graphic in html format, in which each variable of the dataset is represented in a row of the grid, while each column displays the different available graphics for each variable. We called the resulting graphic type *wideplot* because it shows a range of graphics for all of the columns of the dataset. The structure of the function, the arguments it permits and its default values are as follows:

```
#> wideplot(data, dataclass = NULL, logical = NULL, ordered = NULL,
#>   factor = NULL, character = NULL, datetime = NULL, numeric = NULL,
#>   group = NULL, ncol = 7, label = 'FALSE')
```

The only argument necessary to obtain a result is `data` that expects a `data.frame` class object; `dataclass` selects and sorts the types of variables to be shown; `ncol` filters the first *n* columns of the grid, between 3 and 7, which will be shown. The fewer columns displayed, the larger the size of the resulting graphics, a feature that is especially useful if the scale labels dwarf the graphics area; `label` adds to the grid a vector below each group of rows according to the variable type, with the names and order of the graphics; `logical`, `ordered`, `factor`, `character`, `datetime` and `numeric` make it possible to choose which graphics, from among the ones included by the specimen (Sec. 2.3.4), appear in the grid and in what order, for each variable type. Finally, `group` changes the selection of graphics that are shown by default according to the criteria of Table 1.

If the order and graphic types to be shown for each variable type are not specified and if the graphic types aren't filtered using the argument `group`, then the default graphic will contain an opinion-based selection of graphics for each variable type, organized especially to facilitate comparison between graphics of the same row and between graphics of the same column. The user can overwrite this selection of graphics as needed, using the arguments `logical`, `ordered`, `factor`, `character`, `datetime` and `numeric`.

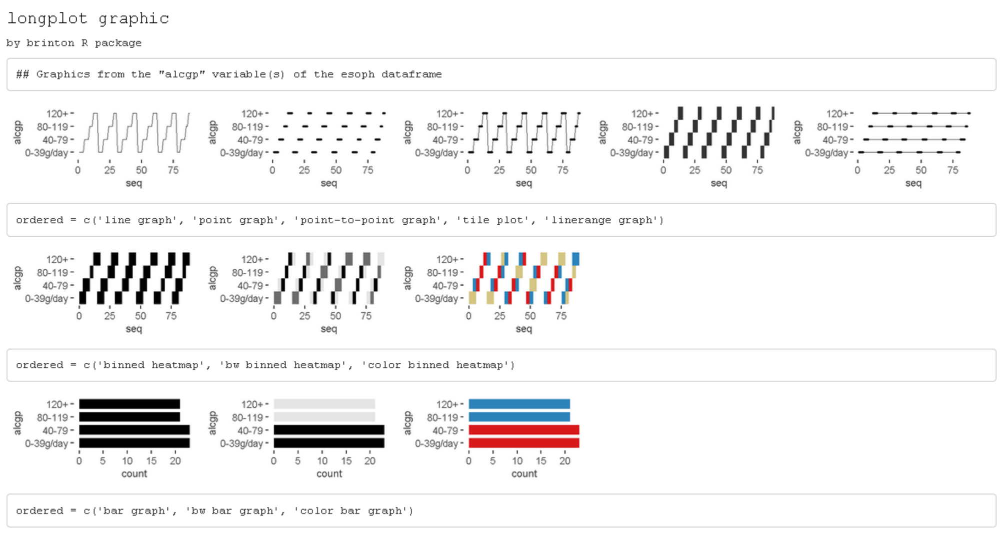
group	**graphic type**
`sequence`	includes the sequence in which the values are observed so that an axis develops this sequence. e.g. line graph, point-to-point graph
`scatter`	marks represent individual observations. e.g. point graph, stripe graph
`bin`	marks represent aggregated observations based on class intervals. e.g. histogram, bar graph
`model`	represents models based on observations. e.g. density plot, violin plot
`symbol`	represents models based on observations. and not only points, lines or areas e.g. box plot
`GOF`	represents the goodness of fit of some values with respect to a model e.g. qq plot
`random`	chosen at random

Table 1: Possible values for the `group` argument of the `wideplot()` function.

The longplot function

To facilitate economy of calculation, the `wideplot()` function presents a limited number of graphics in each row. If the user wants to expand the range of suggested graphics for a given variable, he or she should use the `longplot` function, which returns a grid with all of the graphics considered by the package (See Figure 2) for that variable. The structure of the function is very simple `longplot(data,vars,label = TRUE)` and we can easily check the outcome of applying this function to the variable `alcgp` of the dataset `esoph`:

```
longplot(data = esoph, vars = "alcgp")
```



```
longplot graphic
by brinton R package

## Graphics from the "alcgp" variable(s) of the esoph dataframe

ordered = c('line graph', 'point graph', 'point-to-point graph', 'tile plot', 'linerange graph')

ordered = c('binned heatmap', 'bw binned heatmap', 'color binned heatmap')

ordered = c('bar graph', 'bw bar graph', 'color bar graph')
```

Figure 2: Output of `longplot(esoph, 'alcgp')`. A grid of graphics where the variable `alcgp` in the dataset `esoph` is displayed for the full range of graphics considered by the package.

We named the resulting graphic type *longplot* because it shows the full range of available graphics to represent the relationships among the values of a limited selection of variables (although for now, in this package we have only included graphics for a single variable).

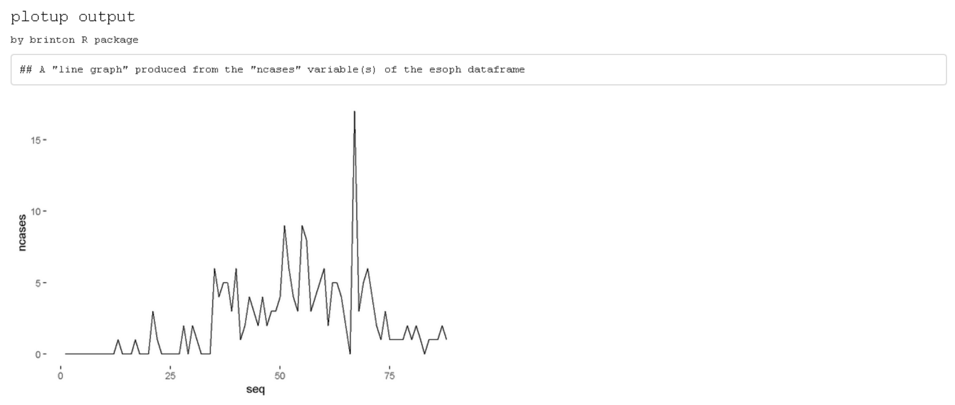
The arguments of the function are `data`, which must be a `data.frame` class object; `vars`, which requires the name of a specific variable of the dataset; and `label`, which does not have to be defined and which adds a vector below each row of the grid indicating the name of each graphic. Unlike the grid of the `wideplot` function, the grid of the `longplot` function does not include parameters to limit the range of graphics to be presented. We made this decision because the main advantage of this function is precisely that it presents all of the graphic representations available for a given variable. However, we do not rule out adding filters that limit the number of graphics to be shown if this feature seems useful as the catalog fills with graphics. Each graphic presented can be called explicitly by name using the functions `wideplot()` and `plotup()`, which is why the argument `label` has been set to `TRUE` by default in this case.

The range of graphics that the `longplot()` function returns is sorted so that in the rows we find different graphic types and in the columns different variations of the same graphic type. This organization, however, is not absolute and in some cases in order to compress the results, we find different graphic types in the same row.

The plotup function

The `plotup()` function has the following structure: `plotup(data,vars,diagram,output = 'plots pane')`. By default, this function returns an object belonging to class gg and `ggplot` whose graphic can be rendered in the plots pane of RStudio. This graphic is based on a variable from a given dataset and the name of the desired graphic, from among the names included by the specimen that we present in the next subsection. We can easily check the outcome of applying this function to produce a line graph from the variable `ncases` of the dataset `esoph` (See Figure 3) :

```
plotup(data = esoph, vars = 'ncases', diagram = 'line graph', output = 'html')
```



plotup output

by brinton R package

```
## A "line graph" produced from the "ncases" variable(s) of the esoph dataframe
```

Figure 3: Output of `plotup(esoph, 'ncases', 'line graph')`. A line graph from the variable `ncases` in the dataset `esoph`.

This function requires three arguments: `data`, `vars` and `diagram`. The fourth argument, `output`, is optional and has the default value of `plots pane`. However, if is set to it `html` or `console`, instead of returning a `c("gg","ggplot")` object, the function cause a side-effect: either creating and displaying a temporary html file, or printing the ggplot2 code to the console. This feature is especially useful to adapt the default graphic to the specific needs and preferences of the user.

The `diagram` argument accepts any of the values admitted by the `logical`, `ordered`, `factor`, `character`, `datetime` and `numeric` arguments of the `wideplot()` function. These values coincide with the names of the graphics considered by the package and included in the specimen. The naming convention of these graphs is implicitly addressed in Section 4 "Graphical degrees of freedom".

```
  plotup(data = esoph, vars = 'ncases', diagram = 'line graph',
         output = 'console')

#> ggplot(esoph, aes(x=seq_along(ncases), y=ncases)) +
#>   geom_line() +
#>   labs(x='seq') +
#>   theme_minimal() +
#>   theme(panel.grid = element_line(colour = NA),
#>     axis.ticks = element_line(color = 'black'))
```

The specimen

The documentation of the package includes the vignette "1v specimen", which contains a specimen with images of all the graphic types for a single variable, incorporated into the package according to the variable type. These graphics serve as an example so that the user can rapidly check whether a graphic has been incorporated, the type or types of variable for which it has been incorporated, and the label with which it has been identified. The suitability of a particular graphic will depend on the datasets of interest and the variables of each particular user. We have incorporated this specimen in its current version as supplementary material.

Graphical degrees of freedom

The utility of this package is based on the fact that different graphical representations of the same data make it possible not only to observe different characteristics of the data, but also to show a certain characteristic more effectively. For this reason, the graphics considered by this package enjoy a large number of graphical degrees of freedom. This makes it possible for the catalog to include both commonly used graphics and graphics that have not yet been developed. The concept of graphical degrees of freedom has been used by Benger and Hege

(2006) to refer to Bertin's visual variables (1967, p.43) but with some modifications. Here we use the concept in a broader sense, as detailed below.

- **Type of graphic**. The main degree of freedom of the graphics catalog is the graphic type. The different graphic types are not necessarily ones that differ greatly from each other. To the contrary, very similar graphics coexist because a high number of users prefer each of them. This is the case, for example, of the density plot and the violin plot shown in Figure 4.

```
wideplot(data = esoph[5],
         numeric = c('filled violin plot', 'filled density plot'))
```

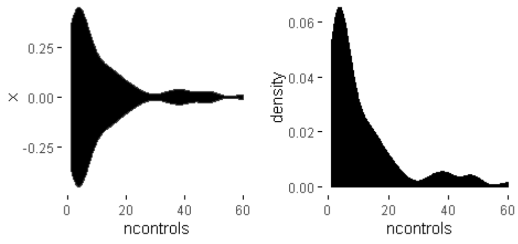


Figure 4: 1st degree of freedom (type of graphic). Density and violin plots of variable ncontrols (in the dataset esoph).

- **Chromatic scales**. The same graphic can have different versions depending on the chromatic scale associated with a variable in the data or computed from it. We can see an example of this in the following figure 5. Despite the fact that color can be broken down into the three visual variables of hue, saturation and value, for the purposes of this package we have only taken into account hue in the case of the color scale and value in the case of the grayscale, following Bertin's classification of visual variables (1967, p.43).

```
wideplot(data = esoph[5],
         numeric = c('histogram', 'bw histogram', 'color histogram'))
```

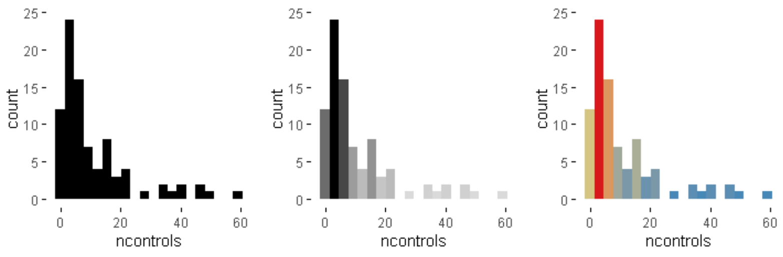


Figure 5: 2nd degree of freedom (chromatic scale). Plots of variable ncontrols (in the dataset esoph) with different chromatic scales.

- **Agreggation method: scattered or binned**. The same values can be represented such that each mark represents either a single value or an aggregate value. An example of this feature can be observed in Figure 6.

```
wideplot(data = esoph[5],
         numeric = c('stripe graph', 'binned stripe graph', 'bar graph',
                     'histogram'))
```

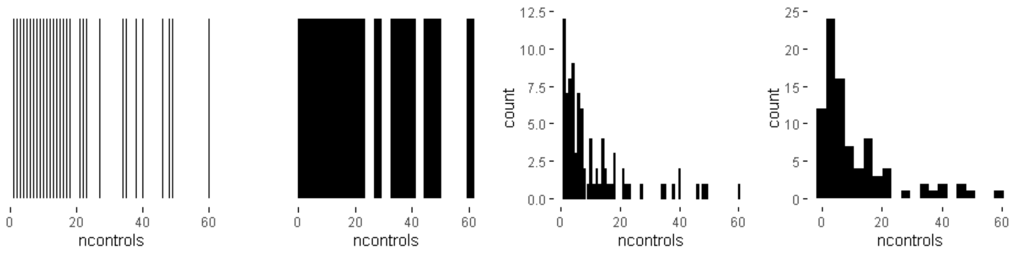


Figure 6: 3rd degree of freedom (aggregation method). Plots of variable `ncontrols` (in the dataset esoph) with single or aggregate values.

- **Nested panels**. One possibility (which has been little explored) is that of subdividing into different panels the cells of the multipanel graphic, to create systems of coordinates inside systems of coordinates. This solution is similar to the treemap. In the example in Figure 7, the graphic on the right has three panels that can substitute the first three graphics.

```
wideplot(data = esoph[5],
         numeric = c('violin plot', 'stripe graph', 'box plot', '3 uniaxial'))
```

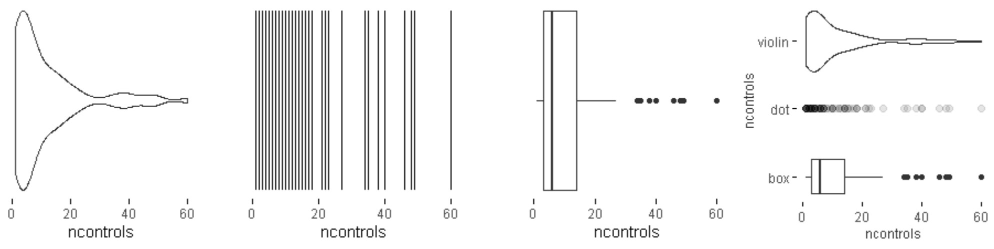


Figure 7: 4th degree of freedom (nested panels). Plots of variable `ncontrols` (in the dataset esoph) with single and multiple panels (the first three ones and the last one respectively).

- **Shape**. The same information can be represented with marks of different shapes. This possibility is exemplified in Figure 8, which compares two graphics with similar composition but different marks: circular or square.

```
wideplot(data = esoph[5],
         numeric = c('color binned point graph', 'color binned heatmap'))
```

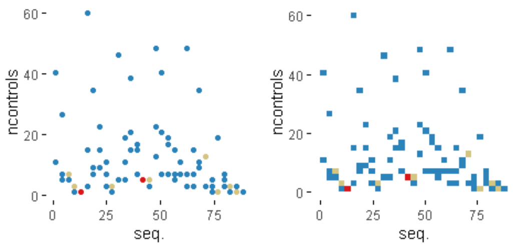


Figure 8: 5th degree of freedom (shape). Plots of variable `ncontrols` (in the dataset esoph) with marks of different shapes.

- **Implantation**. The same values can be represented with marks of a different type of implantation, such as a point, a line, an area or a combination of these. For example, Figure 9 compares a point graph, a line graph, and a point-to-point graph.

```
wideplot(data = esoph[5],
         numeric = c('point graph', 'line graph', 'point-to-point graph'))
```

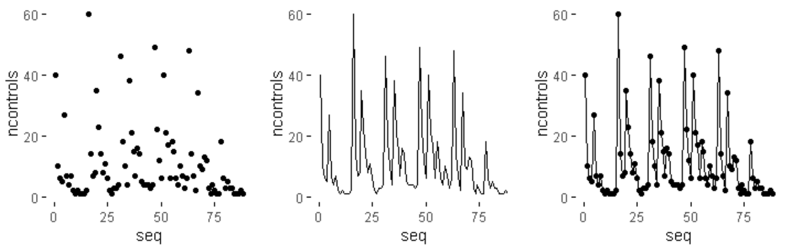


Figure 9: 6th degree of freedom (implantation). Plots of variable `ncontrols` (in the dataset `esoph`) with marks of a different type of implantation.

- **Transition**. The transition or itinerary between two points can help reflect the discrete nature of the changes in the values observed. Figure 10 compares two line graphs with different transitions between points.

```
wideplot(data = esoph[5],
         numeric = c('line graph', 'stepped line graph'))
```

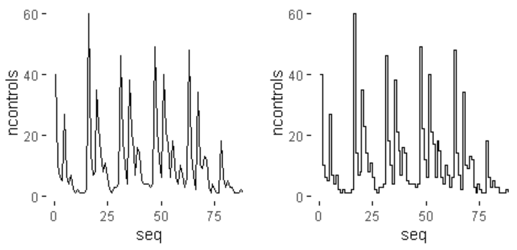


Figure 10: 7th degree of freedom (transition). Plots of variable `ncontrols` (in the dataset `esoph`) with different transitions between points.

- **Collation**. The values of variables, especially those that aren't related to order, can be sorted according to different criteria. This package, as shown in Figure 11 uses three: the order of appearance in the sequence of observations, the frequency with which the values are observed and alphabetical order.

```
wideplot(data = data.frame('Region' = state.region),
         factor = c('tile plot',
                    'freq. reordered tile plot',
                    'alphab. reordered tile plot'))
```

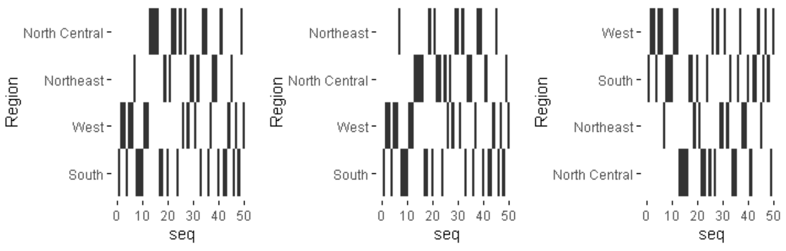


Figure 11: 8th degree of freedom (collation). Plots of variable `Region` with the values sorted according to different criteria.

- **Superposition**. The final degree of freedom that we consider is the possibility of including graphics that superpose marks whose data source is the same but that have different degrees of transformation (see Figure 12).

```
wideplot(data = esoph[5],
         numeric = c('color point graph',
                     'color point graph with trend line'))
```

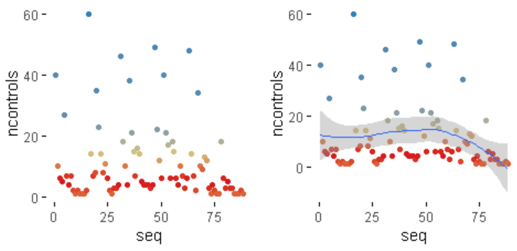


Figure 12: 9th degree of freedom (superposition). Plots of variable `ncontrols` (in the dataset `esoph`) with and without the superposition of a trend line.

To construct the specimen we have ruled out some degrees of freedom, for example, the group of imposition (Bertin, 1967, p.52) and the permutation of spacial variables (Bertin, 1967, p.43). In other words, **brinton** package exclusively presents diagrams and not networks or maps, nor does it show alternatives whose only difference is that the x and y axes are switched.

Application to real datasets

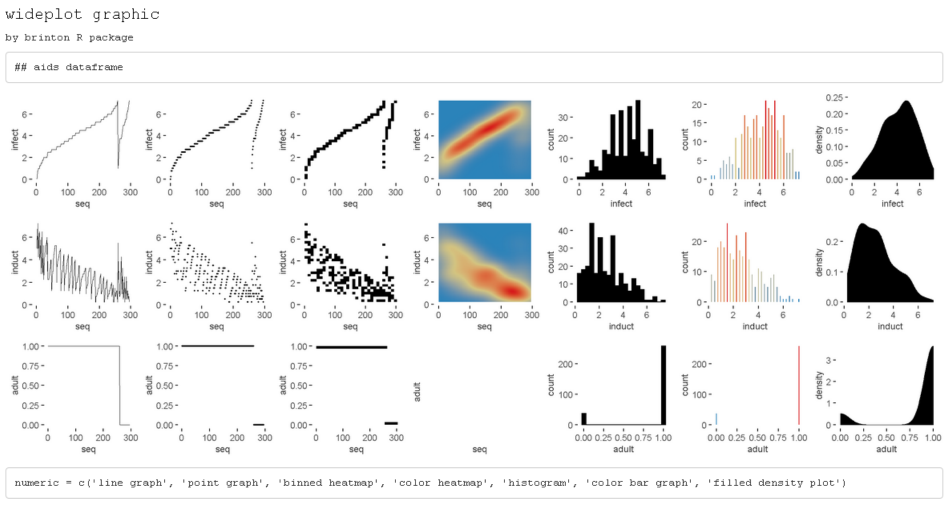
The main application of a package for exploratory data analysis is to help the user make sense of the data. This includes describing the number and nature of the variables, the number of observations and examples of the variables–this is precisely what the `str()` function does. It also includes evaluating the validity and quality of the data and the properties of the values found.

We can deduce the number of variables from the number of rows in the grid of the wideplot graphic. The names of the variables are found in each of the graphics that the catalog now contains. We can determine the variables' nature–in terms of the measurement scale—-by observing the range of graphics selected and specifying the value `label = TRUE` for the grids of wideplot and longplot graphics. We can discern the number of observations by examining the graphics that include the sequence of observations or, in the case of categorical variables, by counting the categories and the number of observations for each one. Wideplot graphics, in contrast to the textual summary of the `str()` function, show examples not only of the first observations but of all observations. To evaluate the validity of the data, we can observe specific graphics that allow us to identify outliers, missing values or discontinuity in the observations. The same goes for the properties of the values found. There is a huge range of graphics, each of which makes it possible to highlight different properties. Below we list a series of tasks for which the functions included in **brinton** are useful, and describe the process for carrying them out.

Identify multi-column sorting

Here we describe how to use the `wideplot()` function to determine whether the observations of the dataset `aids` of the package **KMsurv** are sorted according to one of the variables. This dataset has three variables, `infect` (infection time for AIDS in years), `induct` (induction time for AIDS in years), and `adult` (indicator of adult: 1=adult, 0=child). To accomplish the task, we first install the package, then load it into memory and run the wideplot function with its default output.

```
install.packages('KMsurv')
data(aids, package = 'KMsurv')
wideplot(data = aids, label = TRUE)
```

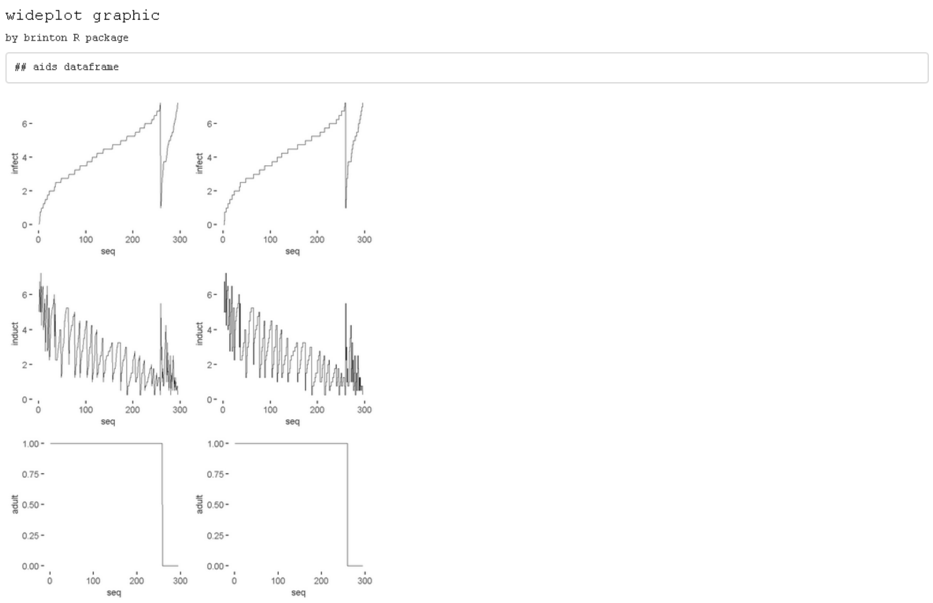


```
wideplot graphic
by brinton R package
## aids dataframe
numeric = c('line graph', 'point graph', 'binned heatmap', 'color heatmap', 'histogram', 'color bar graph', 'filled density plot')
```

Figure 13: A grid of graphics generated by `wideplot(aids, label = T)`. Each row corresponds to a variable in the dataset `aids`. The line graph shows that the dataset is sorted first by the variable `adult` and then by the variable `infect`.

From the result in Figure 13, we observe that the line graph is the one that best shows that the dataset is sorted first by the variable `adult` and then by the variable `infect`. To finish selecting the most suitable graphic we can then execute the same function but limit the graphic types such that only two variations of line graph are shown. We can moreover limit the function so that it displays, for example, only five columns, so that the graphics will be larger.

```
wideplot(data = aids,
         numeric = c('line graph', 'stepped line graph'), ncol = 5)
```



```
wideplot graphic
by brinton R package
## aids dataframe
```

Figure 14: A grid of graphics generated by `wideplot(aids, numeric = c('line graph', 'stepped line graph'), ncol=5)`. Each row corresponds to a variable in the dataset `aids`. Graphic types are limited to line and stepped line types.

The result is two variations of the line graph for each variable, in which we can clearly see that the data set is sorted first by the variable `adult` and then by the variable `infect`. In this case, there may be equally valid arguments for using the graphics of the first column as the graphics of the second column.

This same example also works for datasets with categorical variables, such as the dataset MentalHealth of the package **Stat2Data**. This dataset consists of three variables: `Month` (month of the year); `Moon` (relationship to full moon: `After`, `Before`, or `During`); and `Admission` (number of emergency room admissions). The first two variables are categorical and the third is numerical. If we examine the line graph and also the tile plot for the factor-type variables and the binned heatmap graphic for the numerical variables, we can easily see that the dataset is sorted by the variable `Moon` and then by the variable `Month` (see Figure 15).

```
install.packages('Stat2Data')
data(MentalHealth, package = 'Stat2Data')
wideplot(data = MentalHealth, label = TRUE)
```

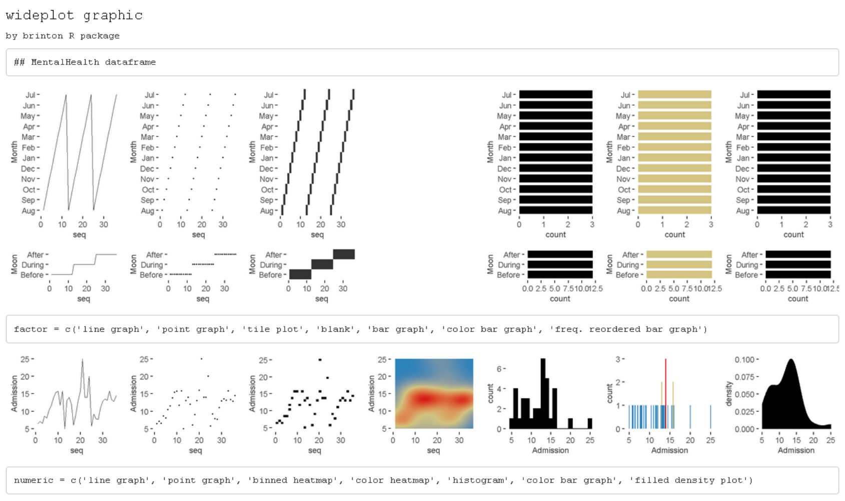

```
wideplot graphic
by brinton R package
## MentalHealth dataframe
factor = c('line graph', 'point graph', 'tile plot', 'blank', 'bar graph', 'color bar graph', 'freq. reordered bar graph')
numeric = c('line graph', 'point graph', 'binned heatmap', 'color heatmap', 'histogram', 'color bar graph', 'filled density plot')
```

Figure 15: A grid of graphics generated by `wideplot(MentalHealth, label = T)`. Each row corresponds to a variable in the dataset `MentalHealth`. It is observed that the dataset is sorted by the variable `Moon` and then by the variable `Month`.

Identify variables that can be reclassified

When loading a dataset it is important to check which assumptions the function has made and which variables can be reclassified. We can see an example of this in Figure 14, which shows that the variable `adult` of the dataset `aids` is better treated as a logical-type variable than an integer. If we recode the variable type more appropriately, when we apply the `wideplot()` function again, the graphics also tend to be more appropriate. In Figure 16 we see the result after the variable `adult` is reclassified.

```
aids$adult <- as.logical(aids$adult)
wideplot(data = aids)
```

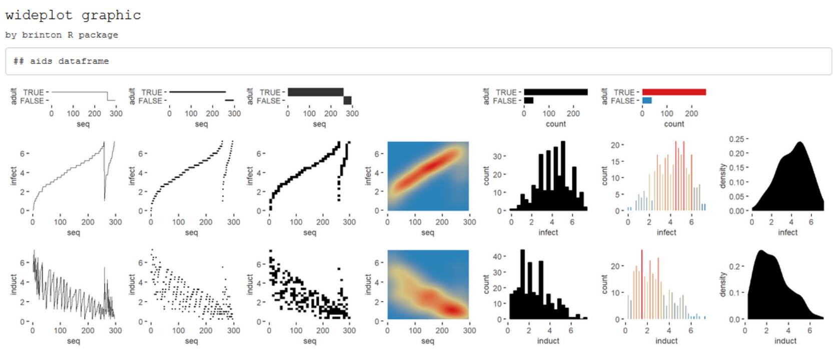
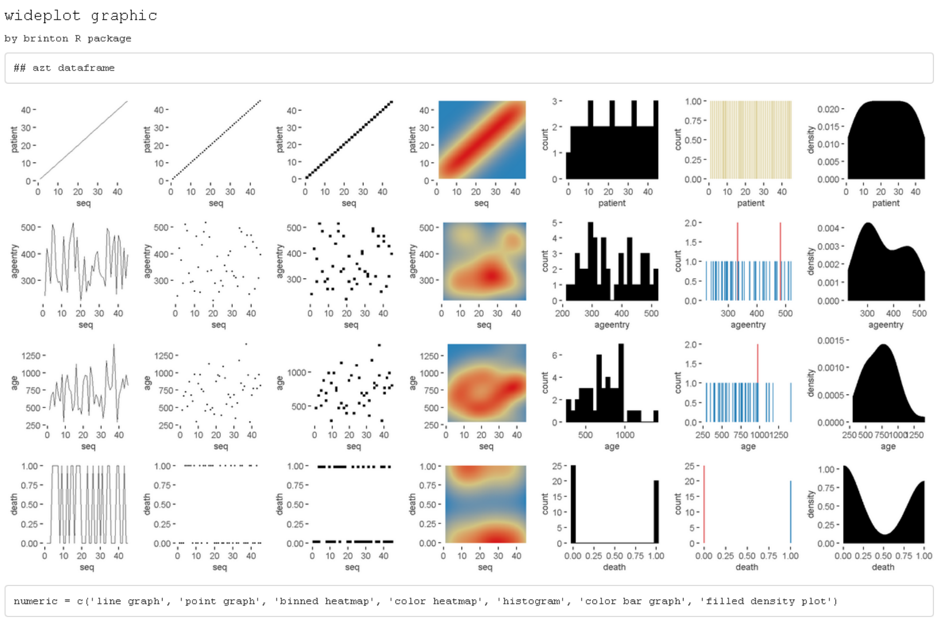


Figure 16: A grid of graphics generated by `wideplot(aids)`. Each row corresponds to a variable in the dataset `aids`. The variable `adult` has been reclassified from integer to logical in order to obtain more appropriate graphics.

Identify key variables

The best way to identify key variables is by using complementary graphics. Figure 17 makes it possible, for example, to identify rapidly the variable `patient` of the dataset `azt` in the package **KMsurv**, as a key variable, given that it assigns a sequential number to each record, each of which is observed a single time. We can draw these two conclusions from the line graph and the color bar graph.

```
data(azt, package = 'KMsurv')
wideplot(data = azt, label = TRUE)
```



```
numeric = c('line graph', 'point graph', 'binned heatmap', 'color heatmap', 'histogram', 'color bar graph', 'filled density plot')
```

Figure 17: A grid of graphics generated by `wideplot(azt, label=TRUE)`. Each row corresponds to a variable in the dataset `azt`. The line graph and the color bar graph recognize the variable `patient` as a key variable that assigns a sequential number to each record.

In the case of categorical key variables, the same line graph and color bar graph would also help us to identify the key variable. Figure 18 shows these two graphs for the factor-type variable of the dataset `SpeciesArea` in the package **Stat2Data**, which allow us to identify rapidly the variable `Name` as a key variable.

```
data(SpeciesArea, package = 'Stat2Data')
```

```
wideplot(data = SpeciesArea, dataclas = c('factor'),
         factor = c('line graph', 'color bar graph'), ncol = 5)
```

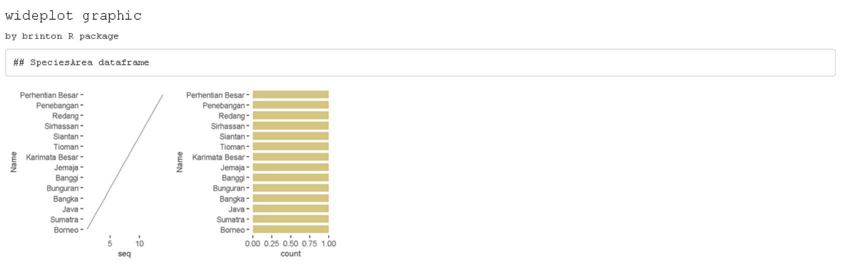


Figure 18: Line and color bar graphs of the factor-type variable `Name` (in the dataset `SpeciesArea`) produced by the function `wideplot()`. It is observed that the variable `Name` is a key variable since the values are not repeated and observed once.

Be surprised by serendipity

Next we describe isolated cases in which we are surprised by the values that the data depict. We use the following procedure to locate unexpected aspects of the data: first we obtain a general view of the dataset using the function `wideplot()`; next we focus our attention on one variable in particular and explore all of the compatible graphics using the function `longplot()`; finally, we use the function `plotup()` to obtain the graphic that best enables us to identify, narrow down and communicate the aspect of the data that we have found.

- The first example of an unexpected funding appears in the variable `experience` (years of potential work experience) of the dataset `HI` in the package **Ecdat**. This dataset contains 22,272 records of 13 variables that link health insurance policies to the weekly hours worked by the wives of the policyholders, while the variable `experience` refers to the years of potential work experience of the wives. If we look at the bar graph applied to this numerical variable (see Figure 19), we see that the frequency of the whole values is systematically greater than the frequency of the real non-whole values. This behavior could indicate that the variable can be informed with high precision and whoever informed the variable `experience` tended to round to the unit. Another possibility is that the dataset was constructed by joining two data sources with different degrees of precision[2]

```
data(HI, package = 'Ecdat')
HI_sam <- HI[sample(nrow(HI), 5000), ]
wideplot(data = HI_sam)                      # Output not reproduced here
longplot(data = HI_sam, vars = 'experience') # Output not reproduced here
plotup(data = HI_sam, vars = 'experience', diagram = 'bar graph')
```

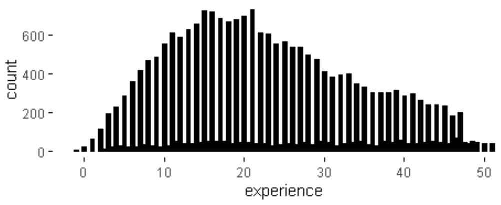


Figure 19: Bar plot of the variable `experience` (in the dataset `HI`) produced by the function `plotup()`. It is observed that the frequency of the whole values is systematically greater than the frequency of the real non-whole values.

[2]In the following example we have decided to limit the number of records to 5,000 to reduce the calculation time and facilitate the reproduction of these same examples.

- In the same dataset we can see that we could reach mistaken conclusions about the distribution of the variable husby (husband's income in thousands of dollars) if we only looked at a histogram. As we can see in Figure 20, the distribution, and in particular the value zero, acquires a different value if we compare the histogram (left) with another graphic that isn't as common for numerical variables: the bar graph (right), which shows the count of unique values. The bar graph makes it possible to clearly differentiate two groups: the informants whose husbands have no income and the informants whose husbands do have income (and to whom, therefore, it makes more sense to ask approximate income).

```
library(patchwork)
plotup(HI_sam, 'husby', 'histogram') + plotup(HI_sam, 'husby', 'bar graph')
```

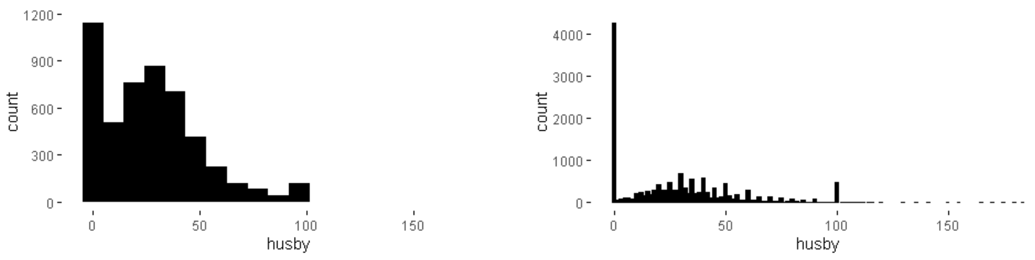


Figure 20: Histogram (left) and bar plot (right) of the variable husby (in the dataset HI_sam) produced by the function plotup(). The bar plot makes it possible to identify the zero as a value with a special meaning.

Combine graphics that best explains a specific data characteristic

Just as multipanel graphics make it possible to reveal different aspects of the data, it can also be helpful to use a selection of graphics to present a certain characteristic of the data. Next, we show an example of how **brinton** package can help us improve the default graphics in order to combine them later to show a particular feature.

- A recurring problem when we deal with datasets with many records is that when marks overlap, we cannot correctly interpret the set of observations. The presentation of multiple graphics to represent the same values enables us to identify these overlaps and improve the representation that the package shows by default. For example, in Figure 21 we can see how the point graph for the same variable husby is unclear because the marks overlap.

```
plotup(data = HI_sam, vars = 'husby', diagram = 'point graph')
```

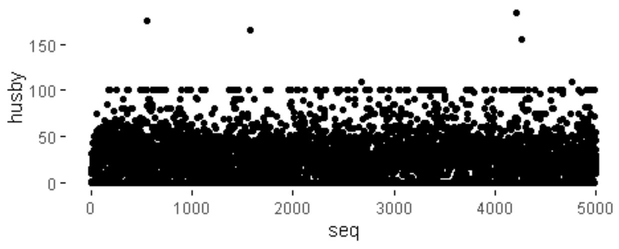


Figure 21: Point plot of the variable husby (in the dataset HI_sam) produced by the function plotup(). The point plot does not identify the zero as a value with a special meaning because the marks overlap.

- We do not have to accept the default result. Rather we can retrieve the package's **ggplot2** function using the argument output = 'console' and then improve it:

```
  plotup(data = HI_sam, vars = 'husby', diagram = 'point graph',
  output = 'console')

#> ggplot(HI_sam, aes(x=seq_along(husby), y=husby)) +
#>   geom_point() +
#>   labs(x='seq') +
#>   theme_minimal() +
#>   theme(panel.grid = element_line(colour = NA),
#>     axis.ticks = element_line(color = 'black'))
```

- In this case we can, for example, improve the graphic by reducing the size of the points and adding an alpha channel (see Figure 22).

```
  newpointgraph <- ggplot(HI_sam, aes(x=seq_along(husby), y=husby)) +
  geom_point(size = 0.3, alpha = 0.15) +
  labs(x='seq') +
  theme_minimal() +
  theme(panel.grid = element_line(colour = NA),
  axis.ticks = element_line(color = 'black'))

  newpointgraph
```

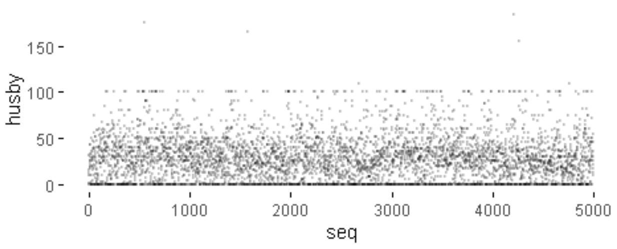


Figure 22: New point plot of the variable `husby` (in the dataset `HI_sam`) produced by the functions `plotup()` and `ggplot()`. Now the point plot makes it possible to identify the zero as a value with a special meaning.

- One option for this graphic that isn't affected by the overlapping of marks is the heatmap, and yet another is the bar graph that represents the frequency with which the unique values are observed. Combining the three graphics helps to highlight different aspects in order to make it easier to understand the data. Figure 23 shows how to combine the three graphics. Note that the bar graph has been rotated 90 degrees to become a marginal plot, following the grammar implemented in **ggplot2**, to facilitate the correspondence between individual observations, the density that can be deduced from them, and the frequency of unique values. Also, the axis labels have been adjusted to avoid unnecessary repetition.

```
  newpointgraph + labs(y = "husband's income * 1000$") +
  plotup(data = HI_sam, vars = 'husby', diagram = 'color heatmap') +
  labs(y = '') +
  plotup(data = HI_sam, vars = 'husby', diagram = 'bar graph') +
  labs(x = '') + coord_flip()
```

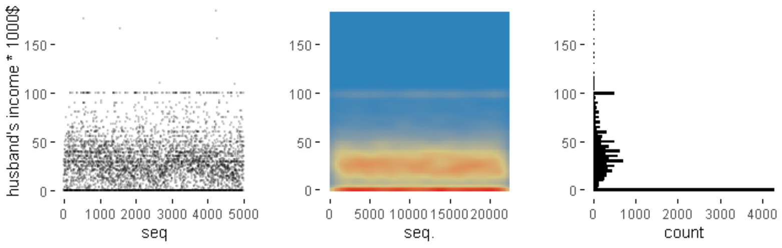


Figure 23: Multipanel graphic as a composition of three plots of the variable husby (in the dataset HI_sam). The source of each plot is the function plotup(). Combining the three plots helps to highlight different aspects of the distribution of the variable husby.

The resulting multipanel graphic shows that throughout the dataset, the revenue distribution remains essentially constant, highlighting the number of husbands without income and rounding the reported values to nice numbers such as 25, 30, 40, 50 and 100–although in reality, the value that draws a horizontal line around 100 is, surprisingly, 99,999. And here we have another mystery to solve.

Conclusions

We have introduced **brinton** package, a graphical EDA tool designed to facilitate the presentation, selection and editing of statistical graphics built on **ggplot2**. This package maximizes the deterministic strategy of graphic selection by presenting a range of graphics that a user can choose by name, automating the construction of graphics and even allowing the user to recover the underlying **ggplot2** function in order to adapt the graphics as necessary. This package makes it easier for a user to become familiar with a dataset and generate hypotheses based on it.

This is a project in progress and new software implementations are being updated and released. We plan to create a fuller catalog that will include graphics that can combine up to three variables, improve the aesthetics of the default graphics and add new functions for autoGEDA.

Acknowledgements

We thank Michael Friendly and Pedro Valero-Mora for corresponding with AUTHOR 1 about the package **cowplot**, which inspired the wideplot() function that forms the core of this package. We acknowledge Susan Frekko for translating so accurately the manuscript from Catalan.

Bibliography

J. Allaire, Y. Xie, J. McPherson, J. Luraschi, K. Ushey, A. Atkins, H. Wickham, J. Cheng, W. Chang, and R. Iannone. *rmarkdown: Dynamic Documents for R*, 2019. URL https://rmarkdown.rstudio.com. R package version 1.12. [p324]

B. Auguie. *gridExtra: Miscellaneous Functions for "Grid" Graphics*, 2017. URL https://CRAN.R-project.org/package=gridExtra. R package version 2.3. [p322, 324]

R. A. Becker, W. S. Cleveland, and M.-J. Shyu. The visual design and control of trellis display. *Journal of computational and Graphical Statistics*, 5(2):123–155, 1996. [p324]

W. Benger and H.-C. Hege. Strategies for direct visualization of second-rank tensor fields. In J. Weickert and H. Hagen, editors, *Visualization and Processing of Tensor Fields*, pages 191–214. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006. ISBN 978-3-540-31272-7. doi: 10.1007/3-540-31272-2_11. URL https://doi.org/10.1007/3-540-31272-2_11. [p329]

J. Bertin. *Sémiologie graphique. Les diagrammes, les réseaux, les cartes*. Mouton, Paris, 1967. [p324, 329, 332]

J. Bertin. *La graphique et le traitement graphique de l'information*. Flammarion, Paris, 1977. [p322]

W. Brinton. *Graphic Presentation*. McGraw-Hill Book Company Inc., New York City, 1939. URL https://archive.org/details/graphicpresentat00brinrich. [p323, 324]

W. Chang and B. Borges Ribeiro. *shinydashboard: Create Dashboards with 'Shiny'*, 2018. URL https://CRAN.R-project.org/package=shinydashboard. R package version 0.7.1. [p323]

W. Cleveland. *The Elements of Graphing Data*. Hobart Press, Summit, New Jersey, 1985. [p324]

D. Comtois. *summarytools: Tools to Quickly and Neatly Summarize Data*, 2019. URL https://CRAN.R-project.org/package=summarytools. R package version 0.9.3. [p323]

D. Cook, D. F. Swayne, and A. Buja. *Interactive and dynamic graphics for data analysis: with R and GGobi*. Springer Science & Business Media, 2007. [p322]

B. Cui. *DataExplorer: Automate Data Exploration and Treatment*, 2019. URL https://CRAN.R-project.org/package=DataExplorer. R package version 0.8.0. [p323]

Dayanand Ubrangala, K. R, R. Prasad Kondapalli, and S. Putatunda. *SmartEDA: Summarize and Explore the Data*, 2019. URL https://CRAN.R-project.org/package=SmartEDA. R package version 0.3.2. [p323]

J. W. Emerson, W. A. Green, B. Schloerke, J. Crowley, D. Cook, H. Hofmann, and H. Wickham. The generalized pairs plot. *Journal of Computational and Graphical Statistics*, 22(1):79–91, 2013. doi: 10.1080/10618600.2012.694762. URL https://doi.org/10.1080/10618600.2012.694762. [p324]

M. Friendly. He plots for multivariate general linear models. *Journal of Computational and Graphical Statistics*, 16(4):421–444, 2007. [p324]

M. Friendly. Lecture 2: Standard graphics in r, 2018. URL http://www.datavis.ca/courses/RGraphics/. OpenCourseWare. [p322]

J. A. Hartigan. Printer graphics for clustering. *Journal of Statistical Computation and Simulation*, 4(3):187–213, 1975. [p324]

R. Iannone, J. Allaire, and B. Borges. *flexdashboard: R Markdown Format for Flexible Dashboards*, 2018. URL https://CRAN.R-project.org/package=flexdashboard. R package version 0.5.1.1. [p323]

T. Kamps. *Diagram Design: A Constructive Theory*. Springer Berlin Heidelberg, 1999. [p322]

P. Millán-Martínez and P. Valero-Mora. Automating statistical diagrammatic representations with data characterization. *Information Visualization*, 17(4):316–334, 2018. [p322]

R. L. Oliver. Effect of expectation and disconfirmation on postexposure product evaluations: An alternative interpretation. *Journal of applied psychology*, 62(4):480, 1977. [p322]

T. L. Pedersen. *patchwork: The Composer of Plots*, 2019. URL https://CRAN.R-project.org/package=patchwork. R package version 1.0.0. [p324]

A. H. Petersen and C. T. Ekstrøm. dataMaid: Your assistant for documenting supervised data quality screening in R. *Journal of Statistical Software*, 90(6):1–38, 2019. doi: 10.18637/jss.v090.i06. [p323]

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2018. URL https://www.R-project.org/. [p]

R. Rao and S. K. Card. The table lens: Merging graphical and symbolic representations in an interactive focus + context visualization for tabular information. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '94, pages 318–322, New York, NY, USA, 1994. ACM. ISBN 0-89791-650-6. doi: 10.1145/191666.191776. URL `http://doi.acm.org/10.1145/191666.191776`. [p323]

A. Rushworth. *inspectdf: Inspection, Comparison and Visualisation of Data Frames*, 2019. URL `https://CRAN.R-project.org/package=inspectdf`. R package version 0.0.4. [p323]

D. Sarkar. *Lattice: Multivariate Data Visualization with R*. Springer, New York, 2008. URL `http://lmdvr.r-forge.r-project.org`. ISBN 978-0-387-75968-5. [p322, 324]

P. Seibelt. *xray: X Ray Vision on your Datasets*, 2017. URL `https://CRAN.R-project.org/package=xray`. R package version 0.2. [p323]

B. Shneiderman. The eyes have it: a task by data type taxonomy for information visualizations. In *Proceedings 1996 IEEE Symposium on Visual Languages*, pages 336–343, Sep. 1996. doi: 10.1109/VL.1996.545307. [p324]

M. Staniak and P. Biecek. The landscape of r packages for automated exploratory data analysis. *arXiv preprint arXiv:1904.02101*, 2019. [p322, 323]

M. Tennekes, E. de Jonge, P. J. Daas, et al. Visualizing and inspecting large datasets with tableplots. *Journal of Data Science*, 11(1):43–58, 2013. [p323]

T. M. Therneau. *A Package for Survival Analysis in S*, 2015. URL `https://CRAN.R-project.org/package=survival`. version 2.38. [p322]

M. Theus and S. Urbanek. *Interactive Graphics for Data Analysis: Principles and Examples (Computer Science and Data Analysis)*. Chapman & Hall/CRC, 2008. ISBN 1584885947, 9781584885948. [p322]

N. Tierney. visdat: Visualising whole data frames. *JOSS*, 2(16):355, 2017. doi: 10.21105/joss.00355. URL `http://dx.doi.org/10.21105/joss.00355`. [p323]

E. R. Tufte. *The Visual Display of Quantitative Information*. Graphics Press, Cheshire, 1983. [p324]

J. Tukey. *Exploratory Data Analysis*. Addison-Wesley series in behavioral science. Addison-Wesley Publishing Company, 1977. ISBN 9780201076165. URL `https://books.google.es/books?id=UT9dAAAAIAAJ`. [p322]

H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. ISBN 978-3-319-24277-4. URL `http://ggplot2.org`. [p322]

C. O. Wilke. *cowplot: Streamlined Plot Theme and Plot Annotations for 'ggplot2'*, 2019. URL `https://CRAN.R-project.org/package=cowplot`. R package version 1.0.0. [p323]

L. Wilkinson. *The Grammar of Graphics*. Statistics and Computing. Springer, 2nd edition, 2005. [p324]

Pere Millán-Martínez
Servei Català de Trànsit
Carrer Diputació, 355 08009 Barcelona, Spain
Research Group on Methodology, Methods, Models and Outcomes of Health and Social Sciences (M3O)
Faculty of Health and Welfare Sciences
Universitat de Vic - UCC
Sagrada Família, 7 08500 Vic, Spain
ORCID: 0000-0003-0879-9358
`info@sciencegraph.org`

Ramon Oller
Data Analysis and Modeling Research Group
Departament d'Economia i Empresa
Universitat de Vic - UCC
Sagrada Família 7, 08500 Vic, Spain
ORCID: 0000-0002-4333-0021
ramon.oller@uvic.cat

MoTBFs: An R Package for Learning Hybrid Bayesian Networks Using Mixtures of Truncated Basis Functions

by Inmaculada Pérez-Bernabé, Ana D. Maldonado, Antonio Salmerón and Thomas D. Nielsen

Abstract This paper introduces **MoTBFs**, an R package for manipulating mixtures of truncated basis functions. This class of functions allows the representation of joint probability distributions involving discrete and continuous variables simultaneously, and includes mixtures of truncated exponentials and mixtures of polynomials as special cases. The package implements functions for learning the parameters of univariate, multivariate, and conditional distributions, and provides support for parameter learning in Bayesian networks with both discrete and continuous variables. Probabilistic inference using forward sampling is also implemented. Part of the functionality of the **MoTBFs** package relies on the **bnlearn** package, which includes functions for learning the structure of a Bayesian network from a data set. Leveraging this functionality, the **MoTBFs** package supports learning of MoTBF-based Bayesian networks over hybrid domains. We give a brief introduction to the methodological context and algorithms implemented in the package. An extensive illustrative example is used to describe the package, its functionality, and its usage.

Introduction

Mixtures of truncated basis functions (MoTBFs) (Langseth et al., 2012a) have been proposed as a general framework for handling hybrid Bayesian networks, i.e., Bayesian networks where discrete and continuous variables coexist. As special cases, the framework includes the so-called mixtures of truncated exponentials (MTEs) (Moral et al., 2001) and mixtures of polynomials (MoPs) (Shenoy and West, 2011; López-Cruz et al., 2012).

One of the advantages of MoTBFs is that they allow for hybrid Bayesian networks with no structural restrictions on the relations between the continuous and discrete variables; this is in contrast to conditional Gaussian (CG) models (Lauritzen, 1992), where discrete variables are not allowed to have continuous parents. Restricting arc directions is problematic, for instance in situations where the network structure is given a causal interpretation; the fact that MoTBFs have no such restriction make them suitable for this kind of analysis. Furthermore, MoTBFs are closed under addition, multiplication, and integration, which facilitates the use of exact probabilistic inference methods like the Shenoy-Shafer architecture (Shenoy and Shafer, 1990) or the variable elimination algorithm (Zhang and Poole, 1996).

Methods for learning MoTBFs from data have previously been studied and cover algorithms for learning both marginal (Langseth et al., 2012b,a) and conditional MoTBF densities from data (Langseth et al., 2009, 2014; Pérez-Bernabé et al., 2015). To address situations where data availability is limited, Pérez-Bernabé et al. (2016) proposed a methodology for integrating prior knowledge when learning univariate and conditional MoTBFs from data. The underlying idea of the integration is to represent the prior knowledge as an MoTBF density that is later combined with the density learned from data, thus resulting in a new MoTBF density. Together, the learning algorithms for MoTBF-based Bayesian networks facilitate learning in data-rich domains as well as domains where limited quantitative data is counterbalanced by qualitative domain knowledge.

The aim of the **MoTBFs** package is to provide a free and accessible implementation of algorithms for learning MoTBFs from data. The package implements state-of-the-art learning algorithms for univariate, conditional, and joint MoTBF densities. By extension, functionality is also provided for learning MoTBF-based Bayesian networks by leveraging functionality from the **bnlearn** package (Scutari, 2010). Hybrid Bayesian networks supported by **bnlearn** are restricted to CG models, which implies that conditional distributions of discrete variables given continuous ones are not permitted. Other packages like **deal** (Bøttcher and Dethlefsen, 2003) and **pcalg** (Kalisch et al., 2012) are also restricted to CG models. By adopting the MoTBF framework, the **MoTBFs** package sidesteps this restriction on the possible network structures. Furthermore, the **MoTBFs** package also provides methods for integrating prior domain knowledge in the learning process, thus also supporting data sparse domains.

Another package that includes hybrid Bayesian networks functionality is **HydeNet** (Dalton and Nutter, 2019). This package is able to handle conditional distributions beyond the Gaussian model class, but discrete variables conditional on continuous ones are modeled using generalized linear models, and inference is only possible using Markov Chain Monte Carlo through an interface to

JAGS (Plummer, 2003). Unlike **MoTBFs** and **bnlearn**, **HydeNet** does not provide functionality for learning the network structure. **HydeNet** is especially appropriate for modeling decision problems, since it implements influence diagrams, which are extensions of Bayesian networks that include decision and utility nodes, similarly to decision trees.

The package **abn** (Kratzer et al., 2019) deals with so-called *additive Bayesian networks*, which are Bayesian networks where each node holds a generalized linear model, and the effect of the parents of each node is additive in terms of the exponential family expression of the conditional distribution. Unlike **HydeNet**, **abn** is able to learn the network structure from data, and adopts a fully Bayesian approach, thus allowing the specification of prior distributions on the parameters in a natural way. The main difference with respect to package **MoTBFs** is that MoTBF distributions do not belong to the exponential family, and in that sense both packages are complementary.

Mixtures of truncated basis functions

The MoTBF framework (Langseth et al., 2012a) is based on the abstract notion of real-valued *basis functions*, which includes both polynomial and exponential functions as special cases.

More formally, let $\mathbf{X}$ be a mixed n-dimensional random vector. Let $\mathbf{Y} = (Y_1, \ldots, Y_d)$ and $\mathbf{Z} = (Z_1, \ldots, Z_c)$ be the discrete and continuous parts of $\mathbf{X}$, respectively, with $c + d = n$. Let $\Psi = \{\psi_i(\cdot)\}_{i=0}^{\infty}$ with $\psi_i : \mathbb{R} \to \mathbb{R}$ define a collection of real basis functions. We say that a function $\hat{f} : \Omega_{\mathbf{X}} \mapsto \mathbb{R}_0^+$ is a *mixture of truncated basis functions* potential to level k wrt. Ψ if one of the following two conditions holds:

- f can be written as

$$f(\mathbf{x}) = f(\mathbf{y}, \mathbf{z}) = \sum_{i=0}^{k} \prod_{j=1}^{c} a_{i,\mathbf{y}}^{(j)} \, \psi_i \left(z_j \right), \tag{1}$$

  where $a_{i,\mathbf{y}}^{(j)}$ are real numbers.

- There is a partition $\Omega_{\mathbf{X}}^1, \ldots, \Omega_{\mathbf{X}}^m$ of $\Omega_{\mathbf{X}}$ for which the domain of the continuous variables, $\Omega_{\mathbf{Z}}$, is divided into hyper-cubes and such that f is defined as

$$f(\mathbf{x}) = f_\ell(\mathbf{x}) \quad \text{if } \mathbf{x} \in \Omega_{\mathbf{X}}^\ell,$$

  where each f_ℓ, $\ell = 1, \ldots, m$ can be written in the form of Equation 1.

Typically, a univariate MoTBF for a variable X does not rely on a partitioning of Ω_X.

To see the relationship between MoTBFs and MoPs (Shenoy and West, 2011), we can instantiate the basis functions as polynomials (i.e., $\psi_i(x) = x^i$, for $i = 0, \ldots, k$) in which case the MoTBF model reduces to an MoP model. For example, with polynomial basis functions, a univariate MoTBF of level k for a variable X is given by

$$f(x) = \sum_{i=0}^{k} \theta_i \, \psi_i \left(x \right).$$

Similarly, by having exponential basis functions the MoTBF model implements an MTE model (Moral et al., 2001). For ease of exposition, we shall in the remainder of this paper assume polynomial basis functions unless explicitly stated otherwise.

An MoTBF potential is a *density* if $\sum_{\mathbf{y} \in \Omega_{\mathbf{Y}}} \int_{\Omega_{\mathbf{z}}} f(\mathbf{y}, \mathbf{z}) d\mathbf{z} = 1$. Similarly, we say that an MoTBF $f(\mathbf{y}, \mathbf{z})$ is a *conditional density* for $\mathbf{Z}' \subseteq \mathbf{Z}$ and $\mathbf{Y}' \subseteq \mathbf{Y}$ given $(\mathbf{Z} \setminus \mathbf{Z}')$ and $(\mathbf{Y} \setminus \mathbf{Y}')$ if

$$\sum_{\mathbf{y}' \in \Omega_{\mathbf{Y}'}} \int_{\Omega_{\mathbf{z}'}} f(\mathbf{y}', \mathbf{y}'', \mathbf{z}', \mathbf{z}') d\mathbf{z}' = 1,$$

for all $\mathbf{z}'' \in \Omega_{\mathbf{Z} \setminus \mathbf{Z}'}$ and $\mathbf{y}'' \in \Omega_{\mathbf{Y} \setminus \mathbf{Y}'}$. Following Langseth et al. (2012a) we furthermore assume that the influence that a set of continuous parent variables $\mathbf{Z}$ have on their child variable X is encoded only through the partitioning of $\Omega_{\mathbf{Z}}$ into hyper-cubes, and not directly in the functional form of $f(x|\mathbf{z})$ inside the hyper-cube $\Omega_{\mathbf{Z}}^j$. That is, for a partitioning $\mathcal{P} = \{\Omega_{\mathbf{Z}}^1, \ldots, \Omega_{\mathbf{Z}}^m\}$ of $\Omega_{\mathbf{Z}}$, the conditional MoTBF is defined for $\mathbf{z} \in \Omega_{\mathbf{Z}}^j$, $1 \le j \le m$, as

$$f_k^{(j)}(x | \mathbf{z} \in \Omega_{\mathbf{Z}}^j) = \sum_{i=0}^{k} \theta_i^{(j)} \, \psi_i^{(j)}(x). \tag{2}$$

In the remainder of this paper we shall assume that a conditional MoTBF density includes only a single 'head' variable, i.e., $|\mathbf{Z}' \cup \mathbf{Y}'| = 1$.

Learning univariate MoTBFs from data

Langseth et al. (2014) present a method for learning univariate MoTBF distributions from data. This method is also implemented in the **MoTBFs** package and is briefly described here together with its extension to both conditional and joint distributions. The estimation procedure relies on the empirical cumulative distribution function (CDF) as a representation of the data, which, for a sample $D = \{x_1, \dots, x_N\}$, is defined as

$$G_N(x) = \frac{1}{N} \sum_{\ell=1}^{N} \mathbf{1}\{x_\ell \leq x\}, \quad x \in \mathbb{R}, \tag{3}$$

where $\mathbf{1}\{\cdot\}$ is the indicator function.

The algorithm developed by Langseth et al. (2014) fits a potential, whose derivative is an MoTBF, to the empirical CDF using least squares. As an example, if we use polynomials as basis functions, $\Psi = \{1, x, x^2, x^3, \dots\}$, the parameters of the CDF, denoted as $c_0, \dots, c_k$, are estimated by solving the optimization problem

$$\begin{aligned} &\underset{c_0,\dots,c_k}{\textbf{minimize}} \sum_{\ell=1}^{N} \left(G_N(x_\ell) - \sum_{i=0}^{k} c_i\, x_\ell^i \right)^2 \\ &\textbf{subject to} \sum_{i=1}^{k} i\, c_i\, x^{i-1} \geq 0 \quad \forall x \in \Omega, \\ &\sum_{i=0}^{k} c_i\, \alpha^i = 0 \text{ and } \sum_{i=0}^{k} c_i\, \beta^i = 1, \end{aligned} \tag{4}$$

where the constraints ensure that the obtained parameter estimates define a valid CDF, with α and β being the minimum and maximum values of the data sample, respectively. More precisely, the first constraint guarantees that the corresponding density is non-negative and the last two restrictions ensure that it integrates to 1. The latter is equivalent to stating that the CDF should be equal to 0 at the minimum and equal to 1 at the maximum. Note that the estimated function is not actually a density, but a CDF instead. An MoTBF density can be obtained by simply taking the derivative of the CDF.

Note that the optimization program above is convex, and can be efficiently solved in theory. However, the infinite number of constraints introduced by imposing that $\frac{\mathrm{d}F(x)}{\mathrm{d}x} \geq 0$ for *all* $x \in \Omega_X$ complicates the implementation on a computer. In practice, we only check that the constraint is fulfilled for a limited set of points spread across Ω_X.

In learning scenarios involving a large amount of data (i.e., when N is large), solving the program can be time consuming. In such cases we define a *grid* on Ω_X, that is selected so that the number of observations is the same between each pair of consecutive grid-points. The grid-points are used to evaluate the objective function instead of the sample points.

The level k of the estimated MoP can be decided using different model selection techniques. For the results presented in this paper we have performed a greedy search, choosing the value for k maximizing the Bayesian information criterion (BIC) (Schwarz, 1978):

$$\mathrm{BIC}(f, D) = \sum_{\ell=1}^{N} \log f(x_\ell) - \frac{k+2}{2} \log N. \tag{5}$$

This choice is motivated by Langseth et al. (2014), who showed that the estimators based on Equation 4 are consistent in terms of the mean squared error for all $x \in \Omega_X$.

Learning conditional MoTBFs from data

In a conditional MoTBF density, the continuous parent variables $\mathbf{Z}$ only influence the child variable X through the partitioning of the domain of $\mathbf{Z}$ into hyper-cubes, and not directly in the functional form of $f(x|\mathbf{z})$ inside each hyper-cube (Langseth et al., 2012a). Thus, learning a conditional MoTBF basically consists in finding a partitioning of the domain of the parent variables and using the procedure above for estimating the density of the child variable for each of these partitions.

This procedure is formally described by Langseth et al. (2014), where the domain of the parent variables, $\Omega_{\mathbf{Z}}$, is incrementally split as long as the BIC score improves. Equal frequency binning is used to determine candidate split points. After splitting a variable Z, the algorithm fits a univariate MoTBF density for each induced sub-partition $\Omega^1_{\mathbf{Z}}$ and $\Omega^2_{\mathbf{Z}}$. A candidate partition $\Omega_{\mathbf{Z}'}$ is only accepted if the BIC score is improved, i.e., if

$$\text{BIC-Gain}(\Omega'_{\mathbf{Z}}, Z) = \text{BIC}(f', D) - \text{BIC}(f, D) > 0,$$

where f' is the conditional MoTBF potential defined over the candidate partition.

Learning joint MoTBFs from data

The procedure for learning joint densities is an extension of the program in Equation 4 to random vectors of arbitrary dimension (Pérez-Bernabé et al., 2015). In the multivariate case, the sample is a set of d-dimensional observations, $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$, $\mathbf{x} \in \Omega_{\mathbf{X}} \subset \mathbb{R}^d$. We say that the event $\mathbf{x}_\ell \leq \mathbf{x}$ is true if and only if $\mathbf{x}_{\ell,i} \leq \mathbf{x}_i$ for each dimension $i = 1, \ldots, d$. For notational convenience we use $\Omega^-_{\mathbf{X}} \in \mathbb{R}^d$ to denote the minimal point of $\Omega_{\mathbf{X}}$ (obtained by choosing the minimum of $\Omega_{\mathbf{X}}$ in each dimension), and let $\Omega^+_{\mathbf{X}} \in \mathbb{R}^d$ be the corresponding maximal point. Then, the empirical CDF is defined as

$$G_N(\mathbf{x}) = \frac{1}{N} \sum_{\ell=1}^{N} \mathbf{1}\{\mathbf{x}_\ell \leq \mathbf{x}\}, \quad \mathbf{x} \in \Omega_{\mathbf{X}} \subset \mathbb{R}^d.$$

The goal is to find a representation of the empirical CDF of the form

$$F(\mathbf{x}) = \sum_{\ell_1=0}^{k} \cdots \sum_{\ell_d=0}^{k} c_{\ell_1,\ell_2,\ldots,\ell_d} \prod_{i=1}^{d} x_i^{\ell_i},$$

obtained by solving the optimization problem

$$\begin{aligned} &\textbf{minimize} \sum_{\ell=1}^{N} \left(G_N(\mathbf{x}_\ell) - F(\mathbf{x}_\ell)\right)^2 \\ &\textbf{subject to } \frac{\partial^d F(\mathbf{x})}{\partial x_1, \ldots, \partial x_d} \geq 0 \quad \forall \mathbf{x} \in \Omega_{\mathbf{X}}, \\ &\qquad F\left(\Omega^-_{\mathbf{X}}\right) = 0 \text{ and } F\left(\Omega^+_{\mathbf{X}}\right) = 1. \end{aligned} \tag{6}$$

The solution to this problem is the parameter-set that defines the joint CDF, and the density can be obtained by differentiation of the joint CDF. As in the univariate case, it is a quadratic optimization problem, that can be solved efficiently if the objective function is only evaluated on a set of grid-points.

Incorporating prior knowledge

There are real world situations, where the amount of available data is insufficient for accurate density estimation. The **MoTBFs** package includes implementations oriented to face these kinds of situations by allowing prior knowledge to be taken into account during density estimation. In Bayesian statistics (Bernardo and Smith, 2009), prior information is encoded as prior probability distributions over the parameters. As an example, consider the case of a random variable representing the body temperature of a patient in a hospital, and assume that the variable is normally distributed with mean μ and standard deviation σ. Prior knowledge could be provided in the form of a prior distribution on μ by establishing that $\mu \sim \mathcal{N}(37, 0.1)$. However, in the case of MoTBF distributions, the parameters do not have a meaning in general. Therefore, there is no clear way in which a practitioner could provide prior information on any of the parameters, although some information could still be specified. For instance, in the body temperature example, the practitioner could choose not to give prior information on any single parameter, but instead provide a full distribution of the variable, reflecting his or her prior knowledge when no data is available. Such prior information could, e.g., include that the body temperature follows a normal distribution with mean 37 and standard deviation 0.5. This is the approach followed by Pérez-Bernabé et al. (2016), where prior knowledge is encoded as an MoTBF distribution over the random variable, which is then later combined with the MoTBF density learned from data.

In order to represent such a prior distribution as an MoTBF, a sample is drawn from the prior – e.g., $\mathcal{N}(37, 0.1)$ in the example above – and the sample is then used to learn an MoTBF using the methods previously described in this paper. Alternatively, one may also rely on direct translation schemes as described in, e.g., (Langseth et al., 2010; Cobb et al., 2006).

Given a prior MoTBF density f_{prior} over a variable X, Pérez-Bernabé et al. (2016) obtain a posterior density f_{post} by making a linear combination of the prior density and the density f_{data} learned from data, expressed as

$$f_{post} = w_p f_{prior} + w_d f_{data} \ ,$$

where w_p is the weight of f_{prior} and w_d is the weight of f_{data}, with $0 \leq w_p \leq 1$, $0 \leq w_d \leq 1$ and $w_p + w_d = 1$. The weights w_p and w_d reflect the way in which they explain the observed data. They are computed by measuring the difference between the log-likelihood produced by each one of the densities (f_{prior} and f_{data}) and the expected log-likelihood that would be produced by a randomly generated MoTBF. See (Pérez-Bernabé et al., 2016) for a detailed explanation of the procedure.

Probabilistic inference

In a Bayesian network, probabilistic inference is the task of computing the posterior distribution of any variable X given that some other variables $\mathbf{Y}$ have been observed to take some value $\mathbf{y}$. Therefore, the goal of probabilistic inference is to compute the density $f(x|\mathbf{y})$, where the value $\mathbf{y}$ is fixed.

An easy approach to estimate this density is by *forward sampling* (Henrion, 1988). The idea is to draw a sample of configurations of the variables in the Bayesian network by simulating each variable using its conditional distribution following a top-down order. Prior to sampling, all the densities in the network are restricted to the value $\mathbf{Y} = \mathbf{y}$ and when a variable is to be sampled, its conditional distribution is restricted to the values already obtained for its parents. Sampling is always possible since the variables are sampled following the topological ordering of the network. Once the sample has been obtained, $f(x|\mathbf{y})$ is estimated as a univariate MoTBF as we described before.

Package description and illustrative example

The **MoTBFs** package is designed using `S3` objects. The functions provided by the package implement the methods explained in the previous sections. The package implements functions for learning univariate, multidimensional, and conditional distributions, and provides support for parameter learning in hybrid Bayesian networks. In addition, it includes functions for incorporating prior knowledge when there is lack of data and for carrying out probabilistic inference. Moreover, two classes are incorporated in the package, `"motbf"` for defining univariate mixtures of truncated basis functions and `"jointmotbf"` for specifying multidimensional MoTBFs.

The functionality of the **MoTBFs** package is illustrated through an analysis carried out on a real world dataset. More precisely, we use the `ecoli` dataset (Lichman, 2013), which is provided along with the package. The dataset contains information about *Escherichia coli* and consists of $n = 336$ records, 8 input variables, and 1 output variable (the class). It is a bacterium of the genus *Escherichia* that is commonly found in the lower intestine of warm-blooded organisms. This dataset can be downloaded from `http://archive.ics.uci.edu/ml/datasets/Ecoli`.

To begin the analysis, the package and the data are loaded by

```
> install.packages("MoTBFs")
> library("MoTBFs")
> data("ecoli", package = "MoTBFs")
> str(ecoli)

'data.frame':        336 obs. of  9 variables:
$ Sequence.Name: chr  "AAT_ECOLI" "ACEA_ECOLI" "ACEK_ECOLI" "ACKA_ECOLI" ...
$ mcg          : num  0.49 0.07 0.56 0.59 0.23 0.67 0.29 0.21 0.2 0.42 ...
$ gvh          : num  0.29 0.4 0.4 0.49 0.32 0.39 0.28 0.34 0.44 0.4 ...
$ lip          : chr  "0.48" "0.48" "0.48" "0.48" ...
$ chg          : chr  "0.5" "0.5" "0.5" "0.5" ...
$ aac          : num  0.56 0.54 0.49 0.52 0.55 0.36 0.44 0.51 0.46 0.56 ...
$ alm1         : num  0.24 0.35 0.37 0.45 0.25 0.38 0.23 0.28 0.51 0.18 ...
$ alm2         : num  0.35 0.44 0.46 0.36 0.35 0.46 0.34 0.39 0.57 0.3 ...
$ class        : chr  "cp" "cp" "cp" "cp" ...
```

The `ecoli` dataset is a data frame with 336 rows corresponding to proteins and 9 columns corresponding to variables. The dataset contains 4 discrete variables, stored as characters, and 5 continuous variables. The variables provide measurements of the cells used for predicting the localization site of proteins. The first variable, `Sequence.Name`, which is the accession number for the SWISS-PROT database, and the output variable `class` will not be used in this running example, and we will therefore remove them from the data frame. The discrete variables `lip` and `chg` are binary attributes, where character numbers are used as states; "0.48" and "1", and "0.5" and "1", respectively. For validation purposes, the dataset is split into a training and a test set.

```
>   data <- ecoli[,-c(1,9)]
>   set.seed(2)
>   dataTT <- TrainingandTestData(data, percentage_test = 0.2)
>   trainingData <- dataTT$Training
>   testData <- dataTT$Test
```

The seed value determines the partitioning of the data into training and test, and is therefore key to reproducing the experiments. From now on, we will carry out all the analyses on the training data, leaving the test dataset for estimating the predictive capabilities of the learned models.

Our illustrative example basically consists of fitting MoTBF densities to a previously learned Bayesian network structure over the variables in the dataset. The structure can, for instance, be obtained, using the function `hc()` from the **bnlearn** package. This function returns a directed acyclic graph obtained from the dataset using a local search method. For the sake of simplicity, we have included the function `LearningHC()` in our package, which automatically converts into factors those columns that are non-numeric, before calling the function `hc()` in **bnlearn**. `LearningHC()` can also be used to discretize the dataset before calling `hc()`, but we are not using this functionality in the running example.

```
>   dag <- LearningHC(trainingData)
>   dag

Bayesian network learned via Score-based methods

model:
[lip][alm1][mcg|lip:alm1][chg|lip][aac|alm1][gvh|mcg][alm2|gvh:lip:alm1]
nodes:                                 7
arcs:                                  8
undirected arcs:                       0
directed arcs:                         8
average markov blanket size:           3.14
average neighbourhood size:            2.29
average branching factor:              1.14

learning algorithm:                    Hill-Climbing
score:                                 BIC (cond. Gauss.)
penalization coefficient:              2.797356
tests used in the learning procedure:  102
optimized:                             TRUE

> plot(dag)
```

The network structure obtained is shown in Figure 1.

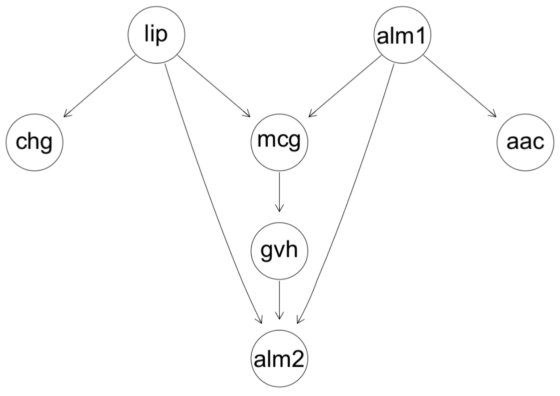


Figure 1: Directed acyclic graph learned from 80 percent of the `ecoli` dataset used as training data.

Before describing how to learn the MoTBF distributions associated with the network structure, we first present the basic functionality for learning different types of MoTBF representations, i.e., univariate, conditional, and joint MoTBF densities.

We illustrate the learning of a univariate MoTBF density by considering the continuous variable mcg.

```
> f1 <- univMoTBF(trainingData[,1], POTENTIAL_TYPE = "MTE", nparam = 13)
> f2 <- univMoTBF(trainingData[,1], POTENTIAL_TYPE = "MOP", nparam = 11)
```

The univMoTBF() function is used for learning univariate densities. The function is at the core of a collection of functions included in the package to learn densities of class "motbf" from data. Least squares optimization is used to minimize the mean squared error between the empirical cumulative distribution and the estimated MoTBF.

The function takes two mandatory arguments, data and POTENTIAL_TYPE, where the latter can either be "MOP" or "MTE" if polynomial or exponential basis functions should be used, respectively. univMoTBF() also accepts optional arguments: it is possible to specify the domain over which the model will be fitted, evalRange, the exact number of basis functions to be used, nparam, and the maximum number of parameters in the function, maxParam, which selects the best fit using the log-likelihood score. If nparam or maxParam are not given, then the Bayesian information criterion (BIC) (Schwarz, 1978) is used for scoring and function selection: it evaluates the two next functions and if the BIC value does not improve then the function with the best BIC score so far is returned.

An overview of the obtained results is shown via print() and summary().[1]

```
R> print(f1)


[1] 31692.5765-19886.9389*exp(2*x)-3430.0930*exp(-2*x)+6520.8968*exp(4*x)
-91374.9603*exp(-4*x)-1269.7572*exp(6*x)+189285.9358*exp(-6*x)
+145.8260*exp(8*x)-186348.2886*exp(-8*x)-8.9962*exp(10*x)+94460.6774*exp(-10*x)
+0.2244*exp(12*x)-19787.1017*exp(-12*x)


> summary(f2)

MoTBFs FOR UNIVARIATE DISTRIBUTIONS

Model:
0.0009+31.8820*x-1161.8247*x^2+16513.1506*x^3-121362.4662*x^4+529132.7157*x^5
-1434074.5145*x^6+2426253.6805*x^7-2482246.1917*x^8+1401057.0127*x^9-334304.5309*x^10

Class: motbf
Subclass: mop

Coefficients:
0.001 31.8821 -1161.825 16513.15 -121362.5 529132.7 -1434075 2426254
-2482246 1401057 -334304.5

Domain:
(0, 0.89)

Number of Iterations: 7

Processing Time: 0.002254009 secs
```

The object returned by univMoTBF() is a list containing several elements, including its mathematical expression and other hidden elements related to the learning task. The processing time is one of the values returned by this function and it can be extracted by $Time. Although the learning process is always the same for a particular data sample, the processing time can vary inasmuch as it depends on the CPU.

```
> hist(trainingData[,1], prob = TRUE , main = "", xlab = "X")
> plot(f1, xlim = range(trainingData[,1]), col = "red", add = TRUE)
> plot(f2, xlim = range(trainingData[,1]), col = "blue", add = TRUE)
```

Figure 2 shows the model fits, provided by univMoTBF(), displayed using the generic method plot().

[1]In order to save space and increase readability, we are only printing the 4 most significant digits in the examples in this paper.

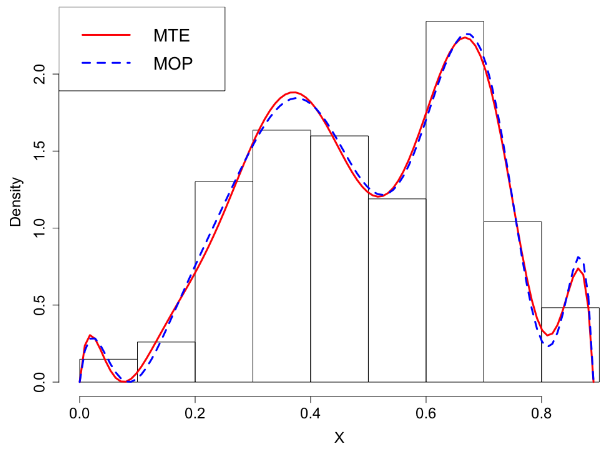


Figure 2: Univariate learning with blue dashed line for MOPs and red solid line for MTEs overlaying the histogram of the training data of the mcg variable.

To evaluate the predictive ability of the models we use the generic method `as.function()` developed for the `"motbf"` class to get the log-likelihood as well as `BICMoTBF()` to obtain the BIC score.

```
> sum(log(as.function(f1)(testData[,1])))
[1] 9.1945

> sum(log(as.function(f2)(testData[,1])))
[1] 8.7249

> BICMoTBF(f1,testData[,1])
[1] -20.2383

> BICMoTBF(f2,testData[,1])
[1] -16.5032
```

An alternative way to visually check the goodness of fit of the estimated models is to simulate a data sample from the learned functions and compare it with the training data. For doing this, we use the inverse transform method, a technique for generating random samples from a specific probability distribution based on evaluating the inverse of the CDF on a uniform random number, yielding a value for the random variable being sampled. This is done by function `rMoTBF()`. For the sake of reproducibility, we fix the seed for the random numbers to be used by the `rMoTBF()` function, which is set to 5 in this example. In the next code snippet, the previous function fitted with a polynomial basis, `f2`, will be used.

```
> set.seed(5)
> X <- rMoTBF(size = 400, fx = f2)
> ks.test(trainingData[,1], X)

Two-sample Kolmogorov-Smirnov test

data:  trainingData[, 1] and X
D = 0.065167, p-value = 0.5018
alternative hypothesis: two-sided
```

In this example the two-sample Kolmogorov-Smirnov test is used. The p-value is notably above 0.05, so there is no evidence to reject the null hypothesis that both samples are drawn from the same population.

```
> hist(X, prob = TRUE, col = "deepskyblue3", main = "", ylim = c(0,2.2))
> hist(trainingData[,1], prob = TRUE, col = adjustcolor("gold",
+       alpha.f = 0.5), add = TRUE)
> plot(ecdf(trainingData[,1]), cex = 0, main = "")
> plot(integralMoTBF(f2), xlim = range(trainingData[,1]), col = "red", add = TRUE)
```

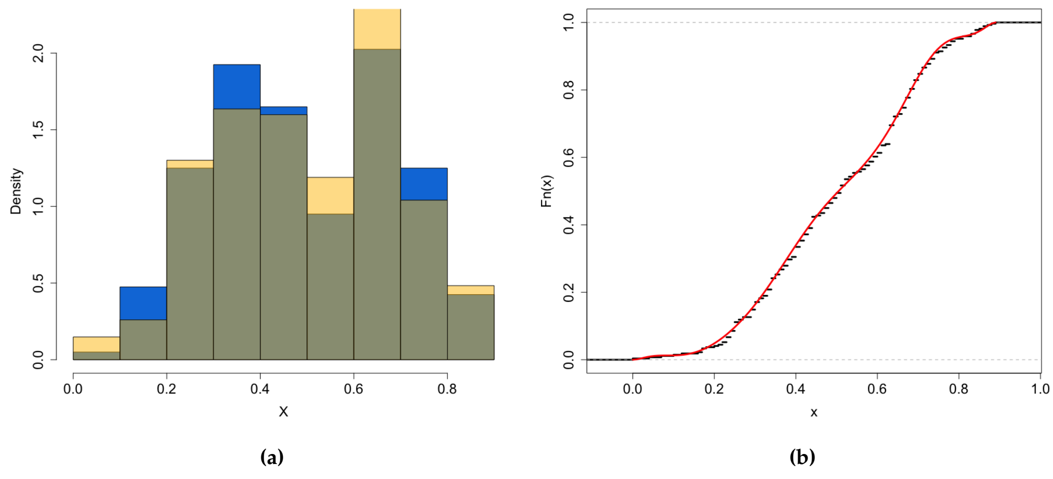


Figure 3: Histogram of variable mcg plotted over the histogram of the generated sample (a). Illustration of the inverse transform method (b), red solid line is the CDF of the generated sample, and black dashed line is the empirical CDF of the training data for variable mcg.

Figure 3 shows two plots comparing the training data of variable mcg and the sample simulated from the distribution learned using the same training data.

We can also manipulate the distributions with a collection of methods for class "motbf". Here is an example of the use of three of them, `coef()`, `integralMoTBF()`, and `derivMoTBF()`.

```
> coef(f1)
[1]  3.1692e+04 -1.9886e+04 -3.4300e+03  6.5208e+03 -9.1374e+04
[6] -1.2697e+03  1.8928e+05  1.4582e+02 -1.8634e+05 -8.9962e+00
[11]  9.4460e+04  2.2449e-01 -1.9787e+04

> integralMoTBF(f2)
[1] 0.0009*x+15.9410*x^2-387.2749*x^3+4128.2876*x^4-24272.4932*x^5+88188.7859*x^6
-204867.7877*x^7+303281.7100*x^8-275805.1324*x^9+140105.7012*x^10-30391.3209*x^11

> integralMoTBF(f2, min = min(trainingData[,1]), max = max(trainingData[,1]))
[1] 1

> derivMoTBF(f2)
[1] 31.8820-2323.6495*x+49539.4519*x^2-485449.8648*x^3+2645663.5790*x^4
-8604447.0881*x^5+16983775.7655*x^6-19857969.5358*x^7+12609513.1160*x^8
-3343045.3101*x^9
```

The learning process for multidimensional variables is similar to the previous one. The function `parametersJointMoTBF()` is used to solve the quadratic optimization problem. It returns `Parameters`, `Range` and `Time`, among other values. The function `jointMoTBF()` is used for obtaining the analytical expression, where the returned object is of class "jointmotbf". The expression is the only visible element, while the others can be retrieved using `attributes()`. In this example only two variables are used, mcg and alm1, in order to be able to plot the results.

```
> parameters <- parametersJointMoTBF(X = trainingData[,c("mcg", "alm1")],
+                  dimensions = c(5,5))
> P <- jointMoTBF(parameters)
> attributes(P)

$names
[1] "Function"   "Domain"     "Iterations" "Time"
$class
[1] "jointmotbf"

> plot(P, data = trainingData[,c(1,6)])
> plot(P, data = trainingData[,c(1,6)], filled = FALSE)
> plot(P, type = "perspective", data = trainingData[,c(1,6)], orientation=c(60,20))
```

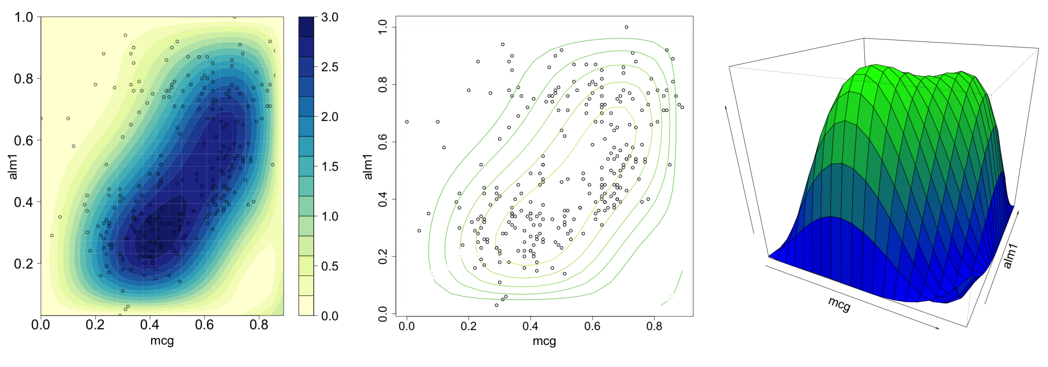


Figure 4: Filled contour (a), simple contour (b) and perspective (c) plots of the joint MoTBF $f(Y, X)$ of variables mcg and alm1.

The plots in Figure 4 are generated using `plot()`, developed for objects of class `"jointmotbf"`. This function accepts optional arguments such as `type`, where one can choose between `"perspective"` and `"contour"`, `ranges`, used to specify the plotting range, `orientation`, which indicates the orientation of the perspective graph, and `filled` for getting a filled contour plot.

The function `print()` can be used to obtain an expression of the learned joint density, while `summary()` yields a more thorough excerpt of the `"jointmotbf"` object.

```
> summary(P)

MoTBFs FOR MULTIVARIATE DISTRIBUTIONS

Model:
0.0355-0.2915*y+0.8351*y^2-0.9711*y^3+0.3939*y^4-2.9706*x+88.6144*x*y
-339.2388*x*y^2+451.7559*x*y^3-198.1810*x*y^4-9.2802*x^2+494.5320*x^2*y
-1392.3658*x^2*y^2+1175.9828*x^2*y^3-268.7925*x^2*y^4+35.3166*x^3
-1708.9509*x^3*y+5662.3268*x^3*y^2-5878.6313*x^3*y^3+1889.8292*x^3*y^4
-22.2566*x^4+1164.5531*x^4*y-4117.2185*x^4*y^2+4479.3097*x^4*y^3-1504.3348*x^4*y^4

Class: jointmotbf

Coefficients:
0.0355 -0.2915 0.8351 -0.9711 0.3939 -2.9706 88.6144 -339.2388 451.7559
-198.1811 -9.2802 494.5321 -1392.366 1175.983 -268.7926 35.3166 -1708.951
5662.327 -5878.631 1889.829 -22.2566 1164.553 -4117.219 4479.31 -1504.335

Domain x:
(0, 0.89)
Domain y:
(0.03, 1)

Number of Iterations: 96

Processing Time: 1.144651 secs
```

As in the univariate case, the processing time, `P$Time`, can vary depending on the CPU, but the learning outcome will always be the same for a specific data sample. The `marginalJointMoTBF()` function computes the marginals of joint densities. In this example we have two variables, so there are two marginal densities.

```
> marginalJointMoTBF(P, var = 1)
[1] 0.0031+1.6119*x+14.1692*x^2-23.7487*x^3+6.7540*x^4

> marginalJointMoTBF(P, var = 2)
[1] -0.2716+13.0463*y-32.4520*y^2+32.5558*y^3-12.8781*y^4
```

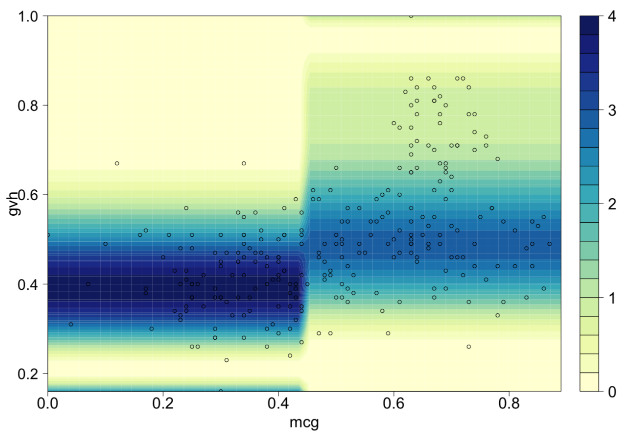


Figure 5: Conditional density of gvh given mcg.

The next step in our analysis is learning conditional densities, which is implemented by the function conditionalMethod(). Five of its arguments are compulsory: data, the dataset; nameParents, a character vector indicating the name of the parents; nameChild, a character string containing the name of the child; numIntervals, the maximum number of intervals for splitting the domain of the parent variables; POTENTIAL_TYPE, the type of basis function. Other arguments are optional, like maxParam, indicating the maximum number of parameters for each function, and s, the expert's relative confidence in any prior knowledge, and priorData if prior knowledge is incorporated in the analysis.

We will do the conditional analysis for only two variables in order to be able to make a 2-dimensional plot of the obtained results using plotConditional(). For example, taking into account the relationship found by the dag, we consider the child variable gvh with parent variable mcg.

```
> P <- conditionalMethod(trainingData, nameParents = "mcg", nameChild = "gvh",
+       numIntervals = 5, POTENTIAL_TYPE ="MOP")
> printConditional(P)

Parent: mcg             Range: 0 < mcg < 0.44
[1] 115.8704-1783.8943*x+10688.6518*x^2-32342.1750*x^3+54497.1689*x^4-52033.2599*x^5
+26393.6470*x^6-5536.0079*x^7
Parent: mcg             Range: 0.44 < mcg < 0.89
[1] -37.3009+733.0877*x-5676.3802*x^2+22283.2874*x^3-47834.5916*x^4+56908.0376*x^5
-35242.7482*x^6+8867.5576*x^7

> plotConditional(P, data = trainingData, nameChild = "gvh", points = TRUE)
```

Figure 5 shows the resulting conditional density (a MOP in this case) with the sample points overlaid. It can be noticed that the learning algorithm decides to split the domain of the parent into two intervals even though we have set the argument numIntervals to five. This is because the BIC score is not improved any further by splitting the domain into more than two intervals.

The last step is to learn the distributions tied to the Bayesian network learned previously. For doing this task, the MoTBFs_Learning() function of the **MoTBFs** package is used. The graph is a mandatory argument, that can be of class "bn", "graphNEL" or "network". Other mandatory arguments are the data, the maximum number of intervals for splitting the domain of the parents, and the type of basis function. The function also accepts additional arguments, but they are not listed here.

In the example, the DAG was obtained using the **bnlearn** package and therefore it is an object of class "bn". As an example, we will use a maximum of 4 intervals and "MTE" potentials when learning the densities (i.e. exponential basis functions).

```
> bn <- MoTBFs_Learning(dag, data = trainingData, numIntervals = 4,
+        POTENTIAL_TYPE = "MTE")
> printBN(bn)

Potential(mcg)
Parent: alm1             Range: 0.03 < alm1 < 0.33
Parent: lip             Range = "0.48"
[1] 77.3522-39.7786*exp(2*x)-4.7525*exp(-2*x)+7.4896*exp(4*x)
-110.4468*exp(-4*x)-0.48548*exp(6*x)+71.3407*exp(-6*x)
```

```
Parent: alm1             Range: 0.33 < alm1 < 1
Parent: lip             Range = "0.48"
[1] -5.4620+3.3347*exp(2*x)+3.6655*exp(-2*x)-0.4237*exp(4*x)-1.0852*exp(-4*x)
Parent: lip             Range = "1"
[1] -11.4070+6.0014*exp(2*x)+4.4488*exp(-2*x)-0.7109*exp(4*x)+2.3944*exp(-4*x)

Potential(gvh)
Parent: mcg             Range: 0 < mcg < 0.51
[1] -97.6780+34.1822*exp(2*x)-282.2167*exp(-2*x)-2.7274*exp(4*x)+2028.8986*exp(-4*x)
-0.12096*exp(6*x)-3609.5088*exp(-6*x)+0.0175*exp(8*x)+2069.4440*exp(-8*x)
Parent: mcg             Range: 0.51 < mcg < 0.89
[1] 685.6507-360.2896*exp(2*x)+252.1477*exp(-2*x)+79.5432*exp(4*x)-3074.9108*exp(-4*x)
-8.3530*exp(6*x)+4377.0877*exp(-6*x)+0.3410*exp(8*x)-2004.8518*exp(-8*x)

Potential(lip)
0.9630 0.0369


Potential(chg)
Parent: lip             Range = "0.48"
1 0
Parent: lip             Range = "1"
0.8181 0.1818


Potential(aac)
Parent: alm1             Range: 0.03 < alm1 < 1
[1] -3742.3665+2528.6429*exp(2*x)+1860.6477*exp(-2*x)-894.0268*exp(4*x)
+2121.1450*exp(-4*x)+175.4771*exp(6*x)-3473.1126*exp(-6*x)-18.0836*exp(8*x)
+1679.6126*exp(-8*x)+0.7634*exp(10*x)-238.3277*exp(-10*x)

Potential(alm1)
[1] 158.6127-95.2652*exp(2*x)-56.9318*exp(-2*x)+25.4631*exp(4*x)-97.0824*exp(-4*x)
-3.1624*exp(6*x)+52.8052*exp(-6*x)+0.1480*exp(8*x)+17.6287*exp(-8*x)

Potential(alm2)
Parent: alm1             Range: 0.03 < alm1 < 0.33
Parent: gvh             Range: 0.16 < gvh < 1
Parent: lip             Range = "0.48"
[1] 193.4903-112.6569*exp(2*x)-52.9058*exp(-2*x)+28.5291*exp(4*x)-185.8167*exp(-4*x)
-3.3698*exp(6*x)+155.0492*exp(-6*x)+0.1517*exp(8*x)-21.3863*exp(-8*x)
Parent: alm1             Range: 0.33 < alm1 < 0.45
Parent: gvh             Range: 0.16 < gvh < 1
Parent: lip             Range = "0.48"
[1] 395.0603-217.7061*exp(2*x)+16.0642*exp(-2*x)+51.0248*exp(4*x)-1009.4114*exp(-4*x)
-5.6246*exp(6*x)+1224.5254*exp(-6*x)+0.2387*exp(8*x)-454.1703*exp(-8*x)
Parent: alm1             Range: 0.45 < alm1 < 0.71
Parent: gvh             Range: 0.16 < gvh < 1
Parent: lip             Range = "0.48"
[1] -3.0260+3.1097*exp(2*x)+6.9062*exp(-2*x)-0.8013*exp(4*x)-12.5762*exp(-4*x)
+0.0578*exp(6*x)+6.3306*exp(-6*x)
Parent: lip             Range = "1"
[1] 1.9648-0.2660*exp(2*x)-0.2660*exp(-2*x)
Parent: alm1             Range: 0.71 < alm1 < 1
Parent: gvh             Range: 0.16 < gvh < 1
Parent: lip             Range = "0.48"
[1] -1.2697+0.6353*exp(2*x)+0.6353*exp(-2*x)
Parent: lip             Range = "1"
[1] 1.0101+0*exp(2*x)
```

The results are reported using the `printBN()` function. Notice how nodes in the DAG with only discrete parents contain as many functions as configurations of the parents, nodes that have continuous parents have at most 4 functions for each parent and nodes that have mixed parents contain as many functions as configurations of the discrete parents times the number of regions into which the domain of the continuous parents is split. The BIC criterion is used to decide the number of splitting points of the domain of the continuous parent nodes and to choose the number of basis functions used. The function `BiC.MoTBFBN()` can be used to compute the log-likelihood and the BIC score of a dataset given the Bayesian network.

```
> bic <- BiC.MoTBFBN(bn, data = testData)
> attributes(bic)
$names
[1] "LogLikelihood" "BIC"

> bic$LogLikelihood
[1] 173.5496
> bic$BIC
[1] -51.40147
```

We will now exemplify the use of prior knowledge in the learning process. In order to illustrate the approach, we first select a small subset of the `Ecoli` dataset using `TrainingandTestData()`. In the next example the percentage of the test data is 99%, which means the training data is only 1% of the full dataset.

```
> set.seed(4)
> dataTT <- TrainingandTestData(data, percentage_test = 0.99)
> trainingData <- dataTT$Training
> testData <- dataTT$Test
> nrow(trainingData)
[1] 13
```

There are 13 entries in the training dataset. We are going to fit MoTBFs with and without prior information. To generate an artificial prior dataset the `generateNormalPriorData()` function can be used.

```
> means <-  sapply(data, mean)
> set.seed(4)
> priorData <- generateNormalPriorData(dag, data = trainingData, size = 5000,
+               means = means)
```

Learning univariate and conditional distributions and Bayesian networks can be done using the functions `learnMoTBFpriorInformation()` and `MoTBFs_Learning()`. The arguments for these functions are the same as previously explained and, in addition, it is necessary to specify the expert confidence in the prior knowledge, `s`, and the prior dataset `priorData`. Argument s takes values on the interval $[0, N]$, where N is the sample size, and is used to synchronize the support of the prior knowledge and the sample. We refer the reader to (Pérez-Bernabé et al., 2016) for the details. In this example we will use the aac variable from the data set, have `s = 5` as confidence level, and set `"MOP"` as potential type.

```
> f <- learnMoTBFpriorInformation(priorData$aac, trainingData$aac,
+        s = 5, POTENTIAL_TYPE = "MOP")
> attributes(f)

$names
[1] "coeffs"            "posteriorFunction" "priorFunction"
[4] "dataFunction"      "domain"

> print(f)

$coeffs
[1] 0.5509206 0.4490794

$posteriorFunction
[1] 0.2911+2.7626*x+4.3721*x^2-30.4218*x^3-0.8389*x^4+965.8444*x^5-4183.3860*x^6
+7735.8433*x^7-7321.9450*x^8+3498.9449*x^9-671.5104*x^10

$priorFunction
[1] 0.0673+1.7639*x+11.8532*x^2-55.2199*x^3-1.5228*x^4+1753.1465*x^5-7593.4470*x^6
+14041.6676*x^7-13290.3826*x^8+6351.0879*x^9-1218.8879*x^10
```

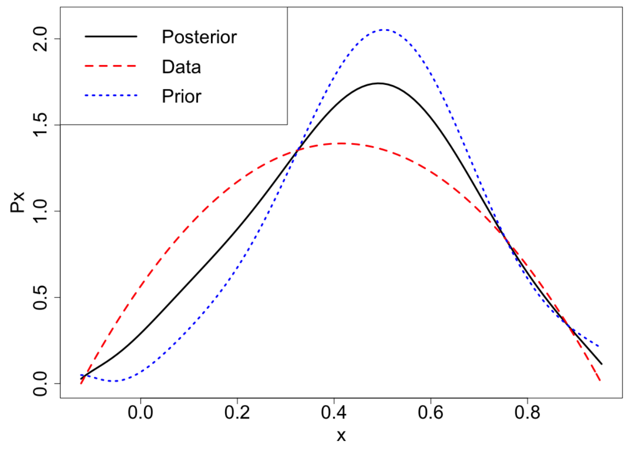


Figure 6: Univariate density estimation using prior knowledge. Solid black, dashed red and dotted blue lines represent the posterior, the data, and the prior function, respectively.

```
$dataFunction
[1] 0.5656+3.9878*x-4.8055*x^2

$domain
[1] -0.1232  0.9531


> sum(log(as.function(f$posteriorFunction)(testData$aac)))
[1] 134.566
> sum(log(as.function(f$dataFunction)(testData$aac)))
[1] 78.96405
```

The best model, taking into account the log-likelihood, is the MoTBF which uses the prior data, `f$posteriorFunction`. The generic method `plot()` for `"motbf"` object is used for displaying the functions depicted in Figure 6.

```
> plot(f$posteriorFunction, xlim = f$domain, ylim = c(0,2.1))
> plot(f$dataFunction, xlim = f$domain, add = TRUE, col = 2)
> plot(f$priorFunction, xlim = f$domain, add = TRUE, col = 4)
```

The last step is to incorporate the prior knowledge in the full Bayesian network. For this analysis we are not going to print out the results, because the structure is similar to the previous Bayesian network representations. As an example, we will use `numIntervals = 2`, `POTENTIAL_TYPE = "MOP"`, and `s = 5`.

```
> priorBN <- MoTBFs_Learning(dag, trainingData, numIntervals = 2,
+              POTENTIAL_TYPE = "MOP", s = 5, priorData = priorData)
> BN <- MoTBFs_Learning(dag, trainingData, numIntervals = 2, POTENTIAL_TYPE = "MOP")

> logLikelihood.MoTBFBN(priorBN, data = testData)
[1] 124.384
> logLikelihood.MoTBFBN(BN, data = testData)
[1] 14.64589
```

Looking at the log-likelihood corresponding to the network with and without prior data, we can see that, in this example, incorporating prior knowledge is better when data is scarce.

After a Bayesian network has been constructed, the **MoTBFs** package can be used to obtain the conditional density of any variable in the network given that some other variables have been observed. The conditional distribution is obtained by forward sampling. As an example, consider a network estimated from the `ecoli` dataset:

```
> data("ecoli", package = "MoTBFs")
> data <- ecoli[,-c(1,9)]
> dag <- LearningHC(data)
> bn <- MoTBFs_Learning(dag, data = data, numIntervals = 4, POTENTIAL_TYPE = "MTE")
```

The observed values are specified using a data frame. In the example, we are assuming that we want to compute the conditional density of alm2 given that lip="0.48", alm1 = 0.55 and gvh = 1. This is achieved by using the function forward_sampling were we have chosen a sample size equal to 10 specified by parameter size = 10.

```
> obs <- data.frame(lip = "0.48", alm1 = 0.55, gvh = 1, stringsAsFactors=FALSE)
> node <- "alm2"
> set.seed(5)
> forward_sampling(bn, dag, target = node, evi = obs, size = 10, maxParam = 15)

Processing Time: 0.209545850753784secs

$fx
[1] -4.3738+2.4552*exp(2*x)+1.5054*exp(-2*x)-0.2392*exp(4*x)+2.0045*exp(-4*x)

$sample
mcg gvh  lip chg       aac alm1       alm2
1  0.7450156   1 0.48 0.5 0.3564026 0.55 0.53709571
2  0.2493266   1 0.48 0.5 0.4873396 0.55 0.55395502
3  0.4075408   1 0.48 0.5 0.5052861 0.55 0.74832025
4  0.3205169   1 0.48 0.5 0.4844040 0.55 0.82546475
5  0.4448844   1 0.48 0.5 0.4248584 0.55 0.07672959
6  0.5717975   1 0.48 0.5 0.4611889 0.55 0.35412707
7  0.7548104   1 0.48 0.5 0.5695499 0.55 0.68322911
8  0.5475842   1 0.48 0.5 0.7797320 0.55 0.45343460
9  0.3048408   1 0.48 0.5 0.5251502 0.55 0.73064241
10 0.7281756   1 0.48 0.5 0.3819332 0.55 0.70396470
```

The output consists of the posterior density and the sample from which the density parameters were estimated.

Conclusions

This paper has presented the R package **MoTBFs** for learning Mixtures of Truncated Basis Functions in hybrid Bayesian networks. It provides a free and accessible implementation of algorithms for learning the parameters of MoTBFs densities as well as MoTBF-based Bayesian networks relying on state-of-the-art learning algorithms.

The **MoTBFs** package is designed to provide the required implementation to tackle experimental data analysis with both discrete and continuous data. Not only does the package provide methods for learning distributions from data, it also includes a set of auxiliary functions to perform descriptive statistics as well as other basic operations like inference using forward sampling.

The **MoTBFs** package expands the functionality for handling hybrid Bayesian networks already provided by packages **bnlearn** and **HydeNet**, by implementing MoTBF distributions, resulting in unrestricted network structures, regardless of the discrete or continuous nature of the variables involved, and by providing methods for building models from data that are compatible with exact inference methods. The package **MoTBFs** is complementary to **abn** in the sense that the former is based on MoTBF densities, which do not belong to the exponential family.

Acknowledgments

This research has been partly funded by the Spanish Ministry of Science and Innovation, through projects TIN2016-77902-C3-3-P, PID2019-106758GB-C32 and by ERDF-FEDER funds.

Bibliography

J. M. Bernardo and A. F. Smith. *Bayesian theory*, volume 405. Wiley. com, 2009. [p346]

S. G. Bøttcher and C. Dethlefsen. deal: A package for learning Bayesian networks. *Journal of Statistical Software*, 8:1–40, 2003. URL https://doi.org/10.18637/jss.v008.i20. [p343]

B. Cobb, P. Shenoy, and R. Rumí. Approximating probability density functions with mixtures of truncated exponentials. *Statistics and Computing*, 16:293–308, 2006. URL `https://doi.org/10.1007/s11222-006-8175-8`. [p346]

J. E. Dalton and B. Nutter. *HydeNet: Hybrid Bayesian Networks Using R and JAGS*, 2019. URL `https://CRAN.R-project.org/package=HydeNet`. R package version 0.10.9. [p343]

M. Henrion. Propagating uncertainty by logic sampling in Bayes' networks. In J. Lemmer and L. Kanal, editors, *Uncertainty in Artificial Intelligence*, volume 2, pages 317–324. North-Holland (Amsterdam), 1988. [p347]

M. Kalisch, M. Maechler, D. Colombo, M. H. Maathuis, and P. Buehlmann. Causal inference using graphical models with the R package pcalg. *Journal of Statistical Software*, 47:1–26, 2012. URL `https://doi.org/10.18637/jss.v047.i11`. [p343]

G. Kratzer, F. I. Lewis, A. Comin, M. Pittavino, and R. Furrer. Additive bayesian network modelling with the r package abn, 2019. [p344]

H. Langseth, T. Nielsen, R. Rumí, and A. Salmerón. Maximum likelihood learning of conditional MTE distributions. *ECSQARU 2009. Lecture Notes in Computer Science*, 5590:240–251, 2009. [p343]

H. Langseth, T. Nielsen, R. Rumí, and A. Salmerón. Parameter estimation and model selection for mixtures of truncated exponentials. *International Journal of Approximate Reasoning*, 51:485–498, 2010. URL `https://doi.org/10.1016/j.ijar.2010.01.008`. [p346]

H. Langseth, T. Nielsen, R. Rumí, and A. Salmerón. Mixtures of truncated basis functions. *International Journal of Approximate Reasoning*, 53:212–227, 2012a. URL `https://doi.org/10.1016/j.ijar.2011.10.004`. [p343, 344, 345]

H. Langseth, T. Nielsen, and A. Salmerón. Learning mixtures of truncated basis functions from data. In *Proceedings of the Sixth European Workshop on Probabilistic Graphical Models (PGM'2012)*, pages 163–170, 2012b. [p343]

H. Langseth, T. Nielsen, I. Pérez-Bernabé, and A. Salmerón. Learning mixtures of truncated basis functions from data. *International Journal of Approximate Reasoning*, 55:940–956, 2014. URL `https://doi.org/10.1016/j.ijar.2013.09.012`. [p343, 345]

S. Lauritzen. Propagation of probabilities, means and variances in mixed graphical association models. *Journal of the American Statistical Association*, 87:1098–1108, 1992. URL `https://doi.org/10.2307/2290647`. [p343]

M. Lichman. UCI machine learning repository, 2013. URL `http://archive.ics.uci.edu/ml`. [p347]

P. L. López-Cruz, C. Bielza, and P. Larrañaga. Learning mixtures of polynomials from data using B-spline interpolation. In A. Cano, M. Gómez-Olmedo, and T. D. Nielsen, editors, *Proceedings of the 6th European Workshop on Probabilistic Graphical Models (PGM'12)*, pages 211–218, 2012. [p343]

S. Moral, R. Rumí, and A. Salmerón. Mixtures of truncated exponentials in hybrid Bayesian networks. In *ECSQARU'01. Lecture Notes in Artificial Intelligence*, volume 2143, pages 135–143, 2001. [p343, 344]

I. Pérez-Bernabé, A. Salmerón, and H. Langseth. Learning conditional distributions using mixtures of truncated basis functions. *ECSQARU'2015. Lecture Notes in Artificial Intelligence*, 9161:397–406, 2015. [p343, 346]

I. Pérez-Bernabé, A. Fernández, R. Rumí, and A. Salmerón. Parameter learning in hybrid Bayesian networks using prior knowledge. *Data Mining and Knowledge Discovery*, 30:576–604, 2016. URL `https://doi.org/10.1007/s10618-015-0429-7`. [p343, 346, 347, 355]

M. Plummer. JAGS: A program for analysis of Bayesian graphical models using Gibbs sampling, 2003. [p344]

G. Schwarz. Estimating the dimension of a model. *Annals of Statistics*, 6:461–464, 1978. [p345, 349]

M. Scutari. Learning bayesian networks with the bnlearn R package. *Journal of Statistical Software*, 35 (3):1–22, 2010. URL `https://doi.org/10.18637/jss.v035.i03`. [p343]

P. Shenoy and G. Shafer. Axioms for probability and belief function propagation. In R. Shachter, T. Levitt, J. Lemmer, and L. Kanal, editors, *Uncertainty in Artificial Intelligence 4*, pages 169–198. North Holland, Amsterdam, 1990. [p343]

P. Shenoy and J. West. Inference in hybrid Bayesian networks using mixtures of polynomials. *International Journal of Approximate Reasoning*, 52:641–657, 2011. URL https://doi.org/10.1016/j.ijar.2010.09.003. [p343, 344]

N. Zhang and D. Poole. Exploiting causal independence in Bayesian network inference. *Journal of Artificial Intelligence Research*, 5:301–328, 1996. URL https://doi.org/10.1613/jair.305. [p343]

Inmaculada Pérez-Bernabé
Department of Mathematics
University of Almería
Almería, 04120, Spain
iperez@ual.es

Ana D. Maldonado
Department of Mathematics
University of Almería
Almería, 04120, Spain
(ORCiD: 0000-0001-8253-2526)
ana.d.maldonado@ual.es

Thomas D. Nielsen
Department of Computer Science
Aalborg University
Aalborg, 9220, Denmark
tdn@cs.aau.dk

Antonio Salmerón
Department of Mathematics and
Center for the Development and Transfer of Mathematical Research to Industry (CDTIME)
University of Almería
Almería, 04120, Spain
(ORCiD: 0000-0003-4982-8725)
antonio.salmeron@ual.es

Analyzing Basket Trials under Multisource Exchangeability Assumptions

by Michael J. Kane, Nan Chen, Alexander M. Kaizer, Xun Jiang, H. Amy Xia, Brian P. Hobbs

Abstract Basket designs are prospective clinical trials that are devised with the hypothesis that the presence of selected molecular features determine a patient's subsequent response to a particular "targeted" treatment strategy. Basket trials are designed to enroll multiple clinical subpopulations to which it is assumed that the therapy in question offers beneficial efficacy in the presence of the targeted molecular profile. The treatment, however, may not offer acceptable efficacy to all subpopulations enrolled. Moreover, for rare disease settings, such as oncology wherein these trials have become popular, marginal measures of statistical evidence are difficult to interpret for sparsely enrolled subpopulations. Consequently, basket trials pose challenges to the traditional paradigm for trial design, which assumes inter-patient exchangeability. The package **basket** for the R programmming environment facilitates the analysis of basket trials by implementing multi-source exchangeability models. By evaluating all possible pairwise exchangeability relationships, this hierarchical modeling framework facilitates Bayesian posterior shrinkage among a collection of discrete and pre-specified subpopulations. Analysis functions are provided to implement posterior inference of the response rates and all possible exchangeability relationships between subpopulations. In addition, the package can identify "poolable" subsets of and report their response characteristics. The functionality of the package is demonstrated using data from an oncology study with subpopulations defined by tumor histology.

Keywords: Bayesian analysis, basket design, hierarchical model, master protocol, oncology, patient heterogeneity

Introduction

Basket designs are prospective clinical trials that are devised with the hypothesis that the presence of selected molecular features determine a patient's subsequent response to a particular "targeted" treatment strategy. Central to the design are assumptions 1) that a patient's expectation of treatment benefit can be ascertained from accurate characterization of their molecular profile and 2) that biomarker-guided treatment selection supersedes traditional clinical indicators for the studied populations, such as primary site of origin or histopathology. Thus, basket trials are designed to enroll multiple clinical subpopulations to which it is assumed that the therapy(s) in question offers beneficial efficacy in the presence of the targeted molecular profile(s). These designs have become popular as drug developers seek to conform therapeutic interventions to the individuals being treated with precision medicine and biomarker-guided therapies. Most basket trials have been conducted within exploratory settings to evaluate agent-specific estimates of tumor response. Cunanan et al. (Cunanan et al., 2017a) describe three studies implemented in oncology settings which extend the basic formulation of a basket trial to multiple targets and/or agent combinations. Most commonly uncontrolled trials, extensions have recently accommodated a wide variety of potential motivations beyond exploratory studies.

Molecularly targeted treatment strategies may not offer acceptable efficacy to all putatively promising clinical indications. Early basket trials were criticized for their reliance on basketwise analysis strategies that suffered from limited power in the presence of imbalanced enrollment as well as failed to convey to the clinical community evidentiary measures of heterogeneity among the studied clinical subpopulations, or "baskets". Acknowledging the potential for differential effectiveness among the enrolled patient subpopulations by design, heterogeneity exists as an intrinsic hypothesis in evaluations of treatment efficacy. Moreover, for rare disease settings, such as oncology wherein these trials have become popular, marginal measures of statistical evidence are difficult to interpret on the basis of individual basket-wise analyses for sparsely enrolled subpopulations. Consequently, basket trials pose specific challenges to the traditional paradigm for trial design, which assume that the patients enrolled represent a statistically exchangeable cohort.

Hobbs and Landin (2018) extended the Bayesian multisource exchangeability model (MEM) framework to basket trial design and subpopulations inference. Initially proposed by Kaizer et al. (2017), the MEM framework addressed the limitations associated with "single-source" Bayesian hierarchical models, which rely on a single parameter to determine the extent of influence, or shrinkage, from all sources. In the presence of subpopulations that arise as mixtures of exchangeable and non-exchangeable subpopulations, single-source hierarchical models (SEM) are characterized by limited borrowing, even in the absence of heterogeneity (Kaizer et al., 2017). Moreover, when considering the effectiveness of a particular treatment strategy targeting a common disease pathway that is observed among differing histological subtypes, SEMs fail to admit statistical measures that delineate which patient subtypes should be considered "non-exchangeable" based on the observed data. By way of contrast, MEM provides a general Bayesian hierarchical modeling strategy accommodating source-specific smoothing parameters. MEMs yield multi-resolution smoothed estimators that are asymptotically consistent and accommodate both full and non-exchangeability among discrete subpopulations. The inclusion of methods for shrinkage of multiple sources is not restricted to use in basket trial master protocols, but has also been extended in the MEM framework to a sequential combinatorial platform trial design where it demonstrated improved efficiency relative to approaches without information sharing (Kaizer et al., 2018).

This paper introduces the **basket** (Chen et al., 2019) package for the R-programming environment to analyze basket trials under MEM assumptions. The main analyses conduct full posterior inference with respect to a set of response rates corresponding to the studied subpopulations. The posterior exchangeability probability (PEP) matrix is calculated, which describes the probability that any pair of baskets are exchangeable. Based on the resultant PEP, subpopulations are clustered into meta-baskets. Additionally, posterior effective sample sizes are calculated for each basket, describing the extent of posterior shrinkage achieved. Posterior summaries are reported for both "basketwise" and "clusterwise" analyses.

The package used in the examples below is available on CRAN at `https://cran.r-project.org/package=basket` and it fits into the general category of the "Design and Analysis of Clinical Trials" (Zhang and Zhang, 2018) focusing on uncontrolled, early-phase trial analysis. The interface is designed to be simple and will readily fit into clinical trial frameworks. It has been tested using R version 3.5 and the **basket** package version 0.9.9.

Exchangeability for Trials with Subpopulations

The Single-Source Exchangeability Model

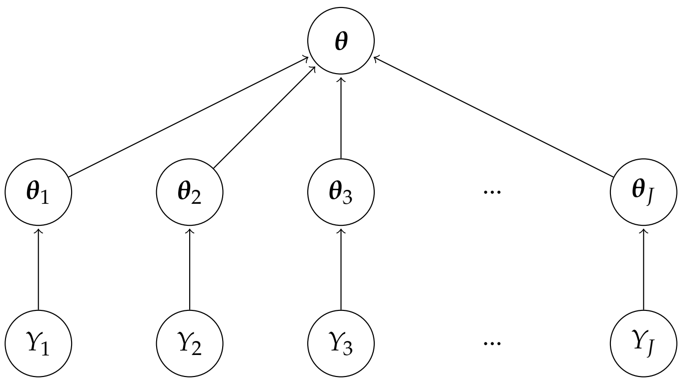

$$\begin{tikzcd}
 & & \boldsymbol{\theta} & & \\
\boldsymbol{\theta}_1 \arrow[urr] & \boldsymbol{\theta}_2 \arrow[ur] & \boldsymbol{\theta}_3 \arrow[u] & \cdots & \boldsymbol{\theta}_J \arrow[ull] \\
Y_1 \arrow[u] & Y_2 \arrow[u] & Y_3 \arrow[u] & \cdots & Y_J \arrow[u]
\end{tikzcd}$$

Figure 1: A conventional single-source Bayesian hierarchical model with J subtypes.

Basket trials intrinsically include subpopulations, which require *a priori* consideration for inference. When ignored the trial simply pools patients, conducting inference with the implicit assumption of inter-patient statistical exchangeability, which can induce bias and preclude the identification of unfavorable/favorable subtypes in the presence of heterogeneity. At the other extreme, subpopulation-specific analyses assume independence. While attenuating bias, this approach suffers from low power, especially in rare subpopulations enrolling limited sample size. Bayesian hierarchical models address this polarity, facilitating information sharing by "borrowing strength" across subtypes with the intent of boosting the effective sample size of inference for individual subtypes.

Single-source exchangeability models (SEM), represent one class of Bayesian hierarchical models. In the context of a basket trial design, statistical approaches using the SEM framework rely on a single parametric distributional family to characterize heterogeneity across all subpopulations, which is computationally tractable but intrinsically reductive in characterization of heterogeneity. In the presence of both exchangeable and non-exchangeable arms, the SEM framework tends to favor the extremes of no borrowing or borrowing equally from all sources, effectively ignoring disjointed singleton subpopulations and meta-subtypes.

Consider a basket trial which enrolls patients from J subpopulations (or subtypes) ($j = 1, ..., J$), where Y_j represents the responses observed among patients in the jth subtypes. Using i to index each patient, the SEM generally relies on model specifications that assume that patient-level responses, $Y_{i,j}$, are exchangeable Bernoulli random variables conditional on subtype-specific model parameters, e.g. $\boldsymbol{\theta}_j$. The second-level of the model hierarchy assumes that the collection of subtype-specific model parameters, $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_J$, are statistically exchangeable through the specification of a common parent distribution. Figure 1 illustrates this structure, wherein each Y_j has its own subtype-specific $\boldsymbol{\theta}_j$ which are further assumed exchangeable to estimate the overall $\boldsymbol{\theta}$.

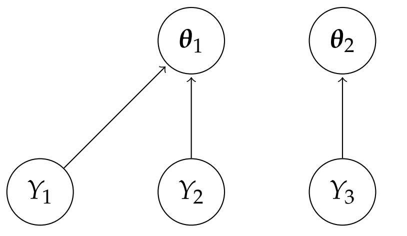
Examples of SEM approaches are introduced and discussed by (Berry et al., 2010, chapter 2), Thall et al. (2003), and Berry et al. (2013), with Hobbs and Landin (2018) providing additional background on these specific SEM implementations. SEM approaches are also implemented in packages by Nia and Davison (2012) and Savage et al. (2018) and have been extended to more specialized applications in fMRI studies (Stocco, 2014), modeling clearance rates of parasites in biological organisms (Sharifi-Malvajerdi et al., 2019), modeling genomic bifurcations (Campbell and Yau, 2017), modeling ChIP-seq data through hidden Ising models (Mo, 2018), modeling genome-wide nucleosome positioning with high-throughput short-read data (Samb et al., 2015), and modeling cross-study analysis of differential gene expression (Scharpf et al., 2009).

While integrating inter-cohort information, SEMs are limited by assumptions of exchangeability among all cohorts. That is, the joint distribution $\mathbb{P}(Y_1, Y_2, ..., Y_k)$ is invariant under a permutation describing subpopulation subsets. $\mathbb{P}(Y_1, Y_2, ..., Y_k) = (Y_k, ..., Y_2, Y_1)$. SEMs are "single-source" in the sense that the model uses a single set of parameters to characterize heterogeneity such that the statistical exchangeability of model parameters is always assumed. Violations of these assumptions with analyses of response rates in clinical trials yields bias, potentially inflating the estimated evidence of an effective response rate for poorly responding cohort or minimizing the effect in effective subsets. These assumptions have resulted in poor results for frequentist power when controlling for strong type I error, leading some cancer trialists to question the utility of Bayesian hierarchical models for phase II trials enrolling discrete subtypes (Freidlin and Korn, 2013; Cunanan et al., 2017b).

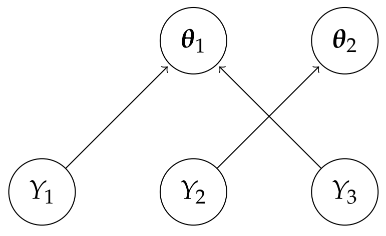
The Multi-source Exchangeability Model

Limitations of SEM can be overcome through model specification devised to explicitly characterize the evidence for exchangeability among collections of subpopulations enrolling in a clinical trial. Multi-source exchangeability models (MEM) produce cohort-specific smoothing parameters that can be estimated in the presence of the data to facilitate dynamic multi-resolution smoothed estimators that reflect the extent to which subsets of subpopulations should be consider exchangeable. Shown to be asymptotically consistent, MEMs were initially proposed by Kaizer et al. (2017) for "asymmetric" cases wherein a primary data source is designated for inference in the presence of potentially non-exchangeable supplemental data sources. The framework was extended by Hobbs and Landin (2018) to the "symmetric" case wherein no single source or subtype is designated as primary (e.g., a basket trial). The symmetric MEM approach considers all possible pairwise exchangeability relationships among J subpopulations and estimates the probability that any subset of subpopulations should be considered statistically exchangeable (or poolable).

The symmetric MEM is the motivation and focus of the **basket** package. While SEMs are parameterized by a single set of parameters $\boldsymbol{\theta}$, the MEM may have up to J (the number of subtypes) sources of exchangeability with each set of data Y_j contributing to only one set of parameters. All possible combinations of exchangeability can be enumerated, denoted as K possible configurations (Ω_k, $k = 1, ..., K$).



(a) Model where Y_1 and Y_2 are exchangeable.



(b) Model where Y_1 and Y_3 are exchangeable.

Figure 2: Two example exchangeability configurations of the MEM.

Model Description

Figure 2 depicts two possible MEMs among three subpopulations wherein at least two subpopulations are statistically exchangeable. Both examples comprise two "sources" of exchangeability for inference, with Y_1 and Y_2 combined to represent one "source" to estimate $\boldsymbol{\theta}_1$ and Y_3 to estimate $\boldsymbol{\theta}_2$ in (a) and Y_1 and Y_3 combined in (b). Implementation of **basket** considers the number of "sources" ranging from one (as in the single-source case), wherein all subtypes are pooled together, to J, the total number of subtypes. The MEM Bayesian model specification facilitates posterior inference with respect to all possible pairwise exchangeability relationships among J subpopulations. The framework facilitates estimation of disjointed subpopulations comprised of meta-subtypes or singelton subtypes and thereby offers additional flexibility when compared to SEM specifications.

The set space of all possible pairwise exchangeability relationships among a collection of J discrete cohorts can be represented by a symmetric $J \times J$ matrix $\boldsymbol{\Omega}$ with element $\Omega_{ij} = \Omega_{ji} \in [0,1]$ with value 1 (0) indicating that patients of subtype i are statistically exchangeable with (independent of) patients of subtype j. Without additional patient-level characteristics, it is assumed patients within an identical subtype are assumed to be statistically exchangeable. That is $\Omega_{ii} = 1$ for $\{i : 1, ..., J\}$. There are $K = \prod_{j=1}^{J-1} 2^j$ possible configurations of $\boldsymbol{\Omega}$, each representing one possible pairwise exchangeability relationship among the J subtypes. The framework differs fundamentally from SEM in that it allows for the existence of multiple closed subpopulations (or cliques) comprised of fully exchangeable subtypes. Therefore, following the terminology of Kaizer et al. (2017) we refer to each possible configuration of $\boldsymbol{\Omega}$ as a MEM.

For a basket trial designed to enroll a total of N patients in J baskets, let $y_{ij} = 1$ indicate the occurrence of a successful response for the ith patient enrolled in basket j, and 0 indicate treatment failure. Let n_j denote the number of patients observed in basket j and denote the total number of responses in basket j by $S_j = \sum_{i=1}^{n_j} y_{ij}$. The set $\{S_1, S_2, ..., S_J\}$ is denoted $\boldsymbol{S}$. Let $\boldsymbol{\pi} = \{\pi_1, \pi_2, ...\pi_J\}$ vectorize the set of response rates such that π_j denotes the probability of response for jth basket and $S_j \sim$Bin(n_j, π_j) with prior distribution $\pi_j \sim$ Beta(a_j, b_j). Let $B()$ denote the beta function. Given an exchangeability configuration $\boldsymbol{\Omega}_j$, the marginal density of S_j follows as (see Hobbs and Landin, 2018, for details)

$$m(\boldsymbol{S}_j \mid \boldsymbol{\Omega}_j,\, \boldsymbol{S}_{(-j)}) \propto \frac{B\left(a + \sum_{h=1}^{J} \Omega_{j,h} S_h,\ b + \sum_{k=1}^{J} \Omega_{j,k}(n_k - S_k)\right)}{B(a,b)} \times \prod_{i=1}^{J} \left(\frac{B(a + S_i,\, b + n_i - S_i)}{B(a,b)}\right)^{1-\Omega_{j,i}}. \tag{1}$$

Marginal posterior inference with respect to $\pi_j \mid \boldsymbol{S}$ averages the conditional posterior of $\pi_j \mid \boldsymbol{\Omega}_j, \boldsymbol{S}$ with respect to the marginal posterior probability of $G = 2^{J-1}$ possible exchangeability configurations of $\boldsymbol{\Omega}_j$. Let $\boldsymbol{\omega} = \{\boldsymbol{\omega}_1, ..., \boldsymbol{\omega}_G\}$ denote the collection of vectors each of length J that collectively span the sample space of $\boldsymbol{\Omega}_j$. The marginal posterior distribution can be represented by a finite mixture density

$$q(\pi_j|\boldsymbol{S}) \propto \sum_{g=1}^{G} q(\pi_j \mid \boldsymbol{S}, \boldsymbol{\Omega}_j = \boldsymbol{\omega}_g) Pr(\boldsymbol{\Omega}_j = \boldsymbol{\omega}_g \mid \boldsymbol{S}), \tag{2}$$

where the posterior probability of exchangeability configuration $\boldsymbol{\omega}_g$ given the observed data follows from Bayes' Theorem in proportion to the marginal density of the data given $\boldsymbol{\omega}_g$ and its unconditional prior probability

$$Pr(\boldsymbol{\Omega}_j = \boldsymbol{\omega}_g \mid \boldsymbol{S}) \propto \frac{m(\boldsymbol{S}_j \mid \boldsymbol{\Omega}_j = \boldsymbol{\omega}_g,\, \boldsymbol{S}_{(-j)}) Pr(\boldsymbol{\Omega}_j = \boldsymbol{\omega}_g)}{\sum_{u=1}^{G} m(\boldsymbol{S}_j \mid \boldsymbol{\Omega}_j = \boldsymbol{\omega}_u,\, \boldsymbol{S}_{(-j)}) Pr(\boldsymbol{\Omega}_j = \boldsymbol{\omega}_u)}. \tag{3}$$

Model specification for the symmetric MEM method is described in detail by Hobbs and Landin (2018).

Estimating Basketwise Exchangeability

The **basket** package computes the posterior probability that subpopulations i and j should be considered statistically exchangeable. The collection of all pairwise posterior exchangeability probabilities (PEP) is denoted in the output as the PEP matrix. Additionally, **basket** identifies the maximum *a posteriori* (MAP) multisource exchangeability model.

Let $\mathcal{O}$ denote the entire sample domain of $\boldsymbol{\Omega}$ comprised of $K = \prod_{j=1}^{J-1} 2^j$ strictly symmetric MEMs.

The PEP matrix is obtained by evaluating the union of MEMs for which $\Omega_{ij} = 1$ over the sample domain of $\mathcal{O}$,

$$\mathbb{P}(\Omega_{ij} = 1|S) = \sum_{\boldsymbol{\Omega}\in\mathcal{O}} \mathbb{1}_{\{\Omega_{ij}=1\}}\, \mathbb{P}(\boldsymbol{\Omega}|S),$$

where $\mathbb{P}(\boldsymbol{\Omega}|S)$ is the product of row-wise calculations specified in Equation 3. Note that there are $K/2$https://www.overleaf.com/project/5c982c6d19014f441ddd8c2d MEM configurations in the space of $\mathcal{O}$ where $\Omega_{ij} = 1$. The MAP follows as the MEM configuration that attains maximum $Pr(\boldsymbol{\Omega} \mid S)$ over $\mathcal{O}$.

Effective Sample Size

Measurement of the extent to which information has been shared across sources in the context of a Bayesian analysis is best characterized by the effective sample size (ESS) of the resultant posterior distribution Hobbs et al. (2013); Murray et al. (2015). ESS quantifies the extent of information sharing, or Bayesian "shrinkage," as the number of samples that would be required to obtain the extent of posterior precision achieved by the candidate posterior distribution when analyzed using a vague "reference" or maximum entropy prior. Calculation of the ESS in **basket** deviates from the approach suggested in Hobbs and Landin (2018), which is sensitive to heavy-tailed posteriors. Robustness is introduced with **basket** through beta distributional approximation, which yields more conservative estimates of ESS. Specifically, the simulated annealing algorithm (implemented with `GenSA` package Yang Xiang et al. (2013)) is used to identify the parametric beta distribution with minimal Euclidean distance between the interval boundaries obtained from the posterior estimated HPD interval and the corresponding beta $1-$`hpd_alpha` Bayesian credible interval. Shape parameters attained from the "nearest" parametric beta distribution are summed to yield estimates of posterior ESS for each basket and cluster.

Posterior Probability

Basket trials are devised for the purpose of testing the hypothesis that a targeted treatment strategy achieves sufficiently promising activity among a partition of the targeted patient population. The MEM framework acknowledges the potential for heterogeneity with respect to the effectiveness of the enrolled patient subpopulations or baskets. Within the MEM framework, this testing procedure follows from the cumulative density function (cdf) of the marginal posterior distribution (2). Specifically, the posterior probability that π_j exceeds a null value π_0 is computed by the weighted average of cdfs for all possible exchangeability configurations. **basket** implements this computation and allows for subpopulation-specific values of the null hypothesis, π_0, which quantify differing benchmarks for effectiveness among the studied baskets. Note that this feature accommodates basket formulation on the basis of varying levels of clinical prognosis.

Package Overview

The **basket** package facilitates implementation of the binary, symmetric multi-source exchangeability model with posterior inference arising with both exact computation and Markov chain Monte Carlo sampling. The user is required to input vectors that describe the number of samples (`size`) and observed successes (`responses`) corresponding to each subpopulation (or basket). Analysis output includes full posterior samples, highest posterior density (HPD) interval boundaries, effective sample sizes (ESS), mean and median posterior estimates, posterior exchangeability probability matrices, and the maximum *a posteriori* MEM. Subgroups can be combined into meta-baskets, or clusters, by setting logical argument `cluster_analysis` to `TRUE`. Cluster analyses use graphical clustering algorithms implemented with the `igraph` package.

A specific clustering algorithm needs to be specified via argument `cluster_function`. The `cluster_function` is a user defined function that first creates a graph using the MAP, then assigns the baskets to discrete clusters using one of the community detection algorithms implemented in the **igraph** package. The default value of `cluster_function` is `cluster_membership`, a function defined in the **basket** package that implements cluster analysis based on the `"cluster_louvain"` method. Users can define their own `cluster_function` using different clustering methods in the **igraph** package. `cluster_analysis` is set to `FALSE` by default. The package includes similar calculations, summaries, and visualization for "clusterwise" and "basketwise" results. Additionally, plotting tools are provided to visualize basket and cluster densities as well as their pairwise exchangeability.

Analysis requires the specification of beta shape parameters (`shape1` and `shape2`) for the prior distributions of the basketwise response probabilities π_j. Shape parameter arguments may be specified as single positive real values, by which identical prior distributions are assumed for all π_j, or as vectors of length J with each pair of `shape1` and `shape2` values corresponding to each basket. Arguments `shape1` and `shape2` assume values 0.5 by default characterizing prior distributions with the effective sample size of 1 patient for each π_j.

The user must additionally specify the symmetric matrix of prior exchangeability probabilities (`prior`). The model assumes that exchangeable information is contributed among patients enrolling into a common basket. Thus, all diagonal entries of `prior` must assume value 1. Off-diagonal entries, however, quantify the *a priori* belief that each pair of subpopulations represents an exchangeable unit. Thus, off-diagonal cells of `prior` may assume any values on the unit interval. The **basket** package assumes the "reference" prior proposed by Hobbs and Landin (2018) as the default setting for which all off-diagonal cells assume prior probability 0.5, and thus are unbiased with respect to exchangeability in the absence of the data.

Evidence for sufficient activity is reported by basket and cluster as posterior probabilities. Posterior probability calculations require the further specification of either a null response rate or vector of null response rates corresponding to each basket (`p0` set to 0.15 by default) as well as the direction of evaluation (`alternative` set to "greater" by default). Additionally, summary functions report the posterior estimates by basket and cluster. The highest posterior density (HPD) is calculated for a given a level of probabilistic significance (`hpd_alpha` set to 0.05 by default).

Bayesian computation is implemented by two methods: the exact method (`mem_exact()` function) and the Markov chain Monte Carlo (MCMC) sampling method (`mem_mcmc()` function). `mem_mcmc()` is the preferred method. `mem_exact()` provides slightly more precise estimates than the former but scales poorly in number of baskets. The discrepancy in precision between exact and sampling-based implementations is easily controlled by specifying a larger number of MCMC iterations (`num_iter` set to `2e+05` by default) in `mem_mcmc()`.

The Exact Method and the MCMC Method

Implementation of `mem_exact()` conducts posterior inference through enumeration of the entire sample domain of MEMs, denoted $\mathcal{O}$ above. Facilitating precise calculation of the posterior estimators, `mem_exact()` is computationally feasible only in the presence of a small number of subpopulations. Increasing the size of J increases the number of configurations in $\mathcal{O}$ by order of $\mathcal{O}(2^{J^2})$. Thus, the exact computation is impractical for large values of J. We recommend its use for $J < 7$.

Our MCMC sampling method, formulated from the Metropolis algorithm (see e.g. Gelman et al., 2013), extends the model's implementation to larger collections of subpopulations, which currently accommodates more than $J = 20$ baskets. Specifically, MCMC sampling is used to approximate the posterior distribution $\mathbb{P}(\boldsymbol{\Omega}_j = \boldsymbol{\omega}_g | \boldsymbol{S})$. Implementation of `mem_mcmc()` requires the specification of an initial MEM matrix (`initial_mem`) used as the starting point for $\boldsymbol{\Omega}$ from which to initiate the Metropolis algorithm. Argument `initial_mem` is set to `round(prior -0.001)` by default, which for the default setting of `prior` yields the identity matrix.

The MCMC algorithm proceeds in iterative fashion with each step selecting a random number of cells of $\boldsymbol{\Omega}$ to flip from 0 to 1 or from 1 to 0 to produce a new candidate MEM which we denote $\boldsymbol{\Omega}^*$. Acceptance criteria for the candidate $\boldsymbol{\Omega}^*$ compares the marginal posterior density of $\boldsymbol{\Omega}^*$ and its unconditional prior distribution with respect to the last accepted MEM matrix configuration. Denote the sum of log marginal posterior density and prior distribution with new candidate MEM configuration by D^* and previously accepted configuration by D_0, respectively. If $D^* - D_0 \geq 0$, the candidate configuration is accepted. Otherwise, the new configuration is accepted randomly with probability $\exp(D^* - D_0)$. For each sampled $\boldsymbol{\Omega}$ configuration, π_j, is sample from its conditional posterior distribution for all $j = 1, ..., J$.

The algorithm initiates with a burn-in period (`mcmc_burnin` set to 50,000 by default). Discarding the burn-in samples, PEP calculation with `mem_mcmc()` evaluates the distribution of sampled MEMs, reporting for all basket pair combinations the proportion of samples that identify basket i as exchangeable with basket j. The MAP calculation reports the posterior mode or most frequently sampled MEM. Bayesian computation facilitated by `mem_mcmc()` scales MEM analyses to more than 20 baskets. Specification of the size of the MCMC iterations (`num_iter`) is pivotal to attaining precise estimates of the resultant posterior quantities. Our investigations support the default value of `2e+05` as a practical lower bound. In practice, one may gradually increase the number of the MCMC iterations until the resultant PEP matrix converges to stable values.

Method	Return Description
`basket_pep`	Basketwise PEP matrix
`basket_map`	Basketwise maximum *a posteriori* probability (MAP) matrix
`cluster_baskets`	Basket assignments for each cluster
`cluster_pep`	Clusterwise PEP matrix
`cluster_map`	Clusterwise MAP matrix

Table 1: MEM model accessor functions.

MEM Data Structure and Associated Methods

Analysis functions `mem_mcmc()` and `mem_exact()` are parameterized almost identically, with the former requiring extra arguments that control the MCMC algorithm: the current seed (for reproducibility), the length of burn-in and number of MCMC iterations for computation of posterior quantities, and an initial MEM matrix from which to start the algorithm. Function arguments are specified with reasonable default values for implementation of either analysis type. Both functions return a common list data structure. Both are derived from an abstract S3 `"exchangeability_model"` class with concrete type `"mem_mcmc"` or `"mem_exact"` depending on which function generated the analysis. The two data structures differ only by extra elements included with `"mem_mcmc"` objects to control implementation of the MCMC algorithm. For convenience, and to promote using `"mem_mcmc"` by default, a wrapper function `basket()` was created. The `method` argument allows the user to specify the analysis function as either MCMC (via `"mcmc"`) or exact (via `"exact"`). By default the argument is set to `"mcmc"`.

MEM or "exchangeability" objects are composed of named elements. The first, `"call"` is the expression used to generate the analysis. Second is the `"basket"` element, which is a list with concrete class `mem_basket`, derived from the `mem` abstract class. Basket reports posterior estimates of trial subpopulations including the PEP, HPD interval, posterior probability, ESS, and other distribution characteristics. The `"cluster"` element comprises a list with concrete class `mem_cluster` and abstract class `mem` which contains posterior estimates for clusters rather than baskets. In addition to these three elements, an `mem_mcmc` object will also contain the seed used to generate the results. This value can be used to reproduce subsequent analyses.

Because they are relatively complex, a `summary` function is implemented to summarize the components relevant to exchangeability models for trial analysis. The `summary.exchangeability_model()` method returns an object of type `"mem_summary"`. A `print.mem_summary()` method is provided for a user-readable summary of the trial. Because there is little distinction between an `exchangeability_model` object and its summary, `print.exchageability_model()` method prints the summary object.

The `mem_summary` object provides access to the overall study characteristic. Accessor methods are also provided to extract other key information from the analysis objects at both the basket and cluster levels. These functions and their descriptions are given in Table 1. In addition, a complete MEM analysis is computationally intensive; altering the null response rate need not imply rerunning the entire analysis. To facilitate partial analysis updates under a new null (argument `p0`), the `update_p0()` function is provided. Likewise samples can be drawn from the posterior distribution of the basket and cluster models using the `sample_posterior()` function.

Visualizations

Two types of functions are provided for visualizing the results of an MEM analysis, both of which are supported at basket and cluster levels of inference. Density plotting is available with the `plot_density()` functions, which produce graphs depicting the posterior distributions of response probabilities at the basket and cluster level. Additionally, functions for visualizing exchangeability relationships are provided in a manner similar to correlograms. Since the values visualized are exchangeability, rather than correlation, we have termed these plots *exchangeograms*. These can be plotted for PEP and MAP matrices using the `plot_pep()` and `plot_map()` functions, respectively. A network graphical visualization integrating the resultant PEP and posterior probability is provided via function `plot_PEP_graph()`.

Case Study: The Vemurafenib Basket Trial

The "Vemurafenib in multiple nonmelanoma cancers with BRAF V600 mutations" study (Hyman et al., 2015), enrolled patients into predetermined baskets that were determined by organ site with

Basket	Enrolled	Evaluable	Responses	Response Rate
NSCLC	20	19	8	0.421
CRC (vemu)	10	10	0	0.000
CRC (vemu+cetu)	27	26	1	0.038
Bile Duct	8	8	1	0.125
ECD or LCH	18	14	6	0.429
ATC	7	7	2	0.286

Table 2: Vemurafenib trial enrollment and responses.

primary end point defined by Response Evaluation Criteria in Solid Tumors (RECIST), version 1.1 (Eisenhauer et al., 2009) or the criteria of the International Myeloma Working Group (IMWG) (Durie et al., 2006). Statistical evidence for preliminary clinical efficacy was obtained through estimation of the organ-specific objective response rates at 8 weeks following the initiation of treatment. This section demonstrates the implementation of **basket** through analysis of six organs comprising non–small-cell lung cancer (NSCLC), cholangiocarcinoma (Bile Duct), Erdheim–Chester disease or Langerhans'-cell histiocytosis (ECD or LCH), anaplastic thyroid cancer (ATC), and colorectal cancer (CRC) which formed two cohorts. Patients with CRC were initially administered vemurafenib. The study was later amended to evaluate vemurafenib in combination with cetuximab for CRC which comprised a new basket. Observed outcomes are summarized in Table 2 by basket. Included in the **basket** package, the dataset is accessible in short `vemu_wide` as well as long formats `vemu`.

Inspection of Table 2 reveals heterogeneity among the studied baskets. CRC (vemu), CRC (vemu+cetu), and Bile Duct had relatively low response rates when compared to other baskets, suggesting that patients presenting the BRAF V600 mutation may not yield exchangeable information for statistical characterization of the effectiveness of the targeted therapy. Therefore, the MEM framework is implemented to measure the extent of basketwise heterogeneity and evaluate the effectiveness of the targeted therapy on the basis of its resultant multi-resolution smoothed posterior distributions. Hobbs et al. (2018) present a permutation study which extends the evaluation of heterogeneity to evaluate summaries of patient attributes reported in Table 1 of the aforementioned trial report. This case study reports posterior probabilities evaluating the evidence that the response probability for each organ-site exceeds the null rate of $\texttt{p0} = 0.25$.

The analysis can be reproduced by loading the `vemu_wide` data, which is included with the package. The data set includes the number of evaluable patients (column `evaluable`), the number of responding patients (column `responders`), and the associated baskets for the respective results (column `baskets`). The model is fit by passing these values to the `basket()` function along with an argument specifying the null response rate of 0.25 for evaluation of each basket. The results are shown by passing the fitted model object to the `summary()` function. Code to perform the analysis as well as produce the output is shown below.

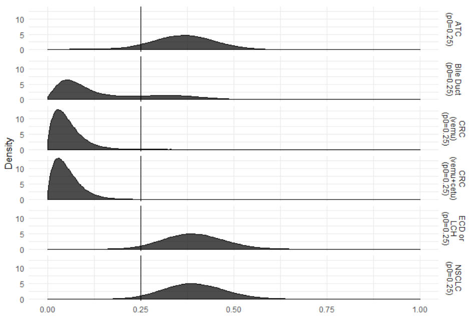
```
library(basket)
      data(vemu_wide)
      vm <- basket(vemu_wide$responders, vemu_wide$evaluable,
      vemu_wide$baskets, p0 = 0.25, cluster_analysis = TRUE)
      summary(vm)


  -- The MEM Model Call ----------------------------------------------------------

  mem_mcmc(responses = responses, size = size, name = name, p0 = p0,
      shape1 = shape1, shape2 = shape2, prior = prior, hpd_alpha = hpd_alpha,
      alternative = alternative, mcmc_iter = mcmc_iter, mcmc_burnin = mcmc_burnin,
      initial_mem = initial_mem, seed = seed, cluster_analysis = cluster_analysis,
      call = call, cluster_function = cluster_function)

  -- The Basket Summary ----------------------------------------------------------

  The Null Response Rates (alternative is greater):
                 NSCLC CRC (vemu) CRC (vemu+cetu) Bile Duct ECD or LCH   ATC
  Null           0.250      0.250            0.25     0.250       0.25 0.250
  Posterior Prob 0.972      0.003            0.00     0.225       0.97 0.891

  Posterior Mean and Median Response Rates:
         NSCLC CRC (vemu) CRC (vemu+cetu) Bile Duct ECD or LCH   ATC
```

```
Mean   0.394      0.055           0.053     0.148      0.394 0.358
Median 0.392      0.046           0.045     0.097      0.391 0.361

Highest Posterior Density Interval with Coverage Probability 0.95:
            NSCLC CRC (vemu) CRC (vemu+cetu) Bile Duct ECD or LCH  ATC
Lower Bound 0.242       0.00           0.001     0.005      0.238 0.17
Upper Bound 0.550       0.13           0.122     0.403      0.551 0.56

Posterior Effective Sample Size:
  NSCLC CRC (vemu) CRC (vemu+cetu) Bile Duct ECD or LCH    ATC
 37.254     49.039          54.514    10.528     36.148 21.786

-- The Cluster Summary ---------------------------------------------------------

Cluster 1
 "CRC (vemu)" "CRC (vemu+cetu)" "Bile Duct"
Cluster 2
 "NSCLC" "ECD or LCH" "ATC"

The Null Response Rates (alternative is greater):
              Cluster 1 Cluster 2
Null              0.250     0.250
Posterior Prob    0.076     0.944

Posterior Mean and Median Response Rates:
       Cluster 1 Cluster 2
Mean       0.085     0.382
Median     0.057     0.382

Highest Posterior Density Interval with Coverage Probability 0.95:
            Cluster 1 Cluster 2
Lower Bound     0.000     0.221
Upper Bound     0.313     0.559

Posterior Effective Sample Size:
 Cluster 1 Cluster 2
     9.786     30.12
```

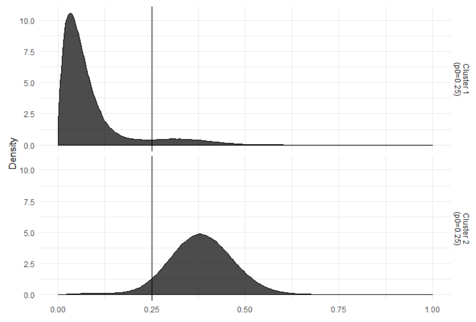
Bayesian MEM analysis using the MCMC sampler with reference prior distribution for exchangeability identifies the most likely MEM to be comprised of two closed subgraphs (or meta-baskets). Cluster 1 consists of CRC (vemu) with CRC (vemu+cetu) and BD, while cluster 2 is comprised of NSCLC, ECD or LCH, and ATC. Cluster 1 results in an estimated posterior mean response rate of 0.087. The posterior probability that baskets assigned to cluster 1 exceed the null response rate of 0.25 is only 0.082. Conversely, attaining a posterior probability of 0.944 and posterior mean of 0.382, indications identified in cluster 2 demonstrate more promising indications of activity. Figures 3a and 3b depict full posterior distributions of response probabilities for each basket and cluster produced by the `plot_density()` function.

```
plot_density(vm, type = "basket")
plot_density(vm, type = "cluster")
```

The resultant posterior probability of each pairwise exchangeability relationship (PEP) is summarized with the `basket_pep()` function and depicted in Figure 4 by application of the `plot_pep()` function. The results demonstrate that the posterior exchangeability between the high-response baskets is higher than that of the lower responding baskets. For example, the posterior probability that NSCLC and ED.LH patients are exchangeable with respect to evaluating their response to Vemurafenib is 0.938. Similarly, the analysis resulted in PEPs of 0.86 for the pairwise relationships between NSCLC with ATC and ED.LH with ATC, suggesting that these indications can be averaged. The study provided strong support to conclude that vemurafenib is identically ineffective among CRC (vemu) and CRC (vemu+cetu) subtypes with PEP = 0.92. The effectiveness of BD was identified as marginally exchangeable with CRC (vemu) and CRC (vemu+cetu) with PEP = 0.64 and 0.63, respectively. Conversely, both NSCLC and ED.LH resulted in PEPs of 0 for each CRC basket, demonstrating strong evidence of differential activity among these indications. Thus, definitive trials devised to estimate population-averaged effects should not expect these subtypes to comprise statistically exchangeable



(a) Posterior basket response density.



(b) Posterior cluster response density.

Figure 3: Posterior distributions of the MEM analysis.

patients.

Figure 5 provides a network graphical representation of the results from analysis of the Vemurafenib study. This graph is generated using the `plot_pep_graph()` function. Nodes represent individual baskets. A node's color depicts the Bayesian evaluation of the null hypothesis that the posterior probability that the objective response rate exceeds 0.25 for the corresponding basket. Edge thickness between any pair of baskets is determined by PEP. Edges with shorter length and thicker width denote basket pairs with higher magnitudes of pairwise posterior exchangeability. For example, baskets ATC, NSCLC, and ECD or LCH, depicted with yellow colored nodes, resulted in higher poster probability when compared to the other three baskets. The relatively thick edges between these baskets confer their large PEP values, suggesting that these indications can be averaged.

```
basket_pep(vm)
plot_pep(vm$basket)
plot_pep_graph(vm)

                NSCLC CRC (vemu) CRC (vemu+cetu) Bile Duct ECD or LCH   ATC
NSCLC           1.000      0.002           0.000     0.231      0.938 0.866
CRC (vemu)      0.002      1.000           0.917     0.643      0.002 0.068
CRC (vemu+cetu) 0.000      0.917           1.000     0.626      0.000 0.031
Bile Duct       0.231      0.643           0.626     1.000      0.243 0.536
ECD or LCH      0.938      0.002           0.000     0.243      1.000 0.861
ATC             0.866      0.068           0.031     0.536      0.861 1.000
```

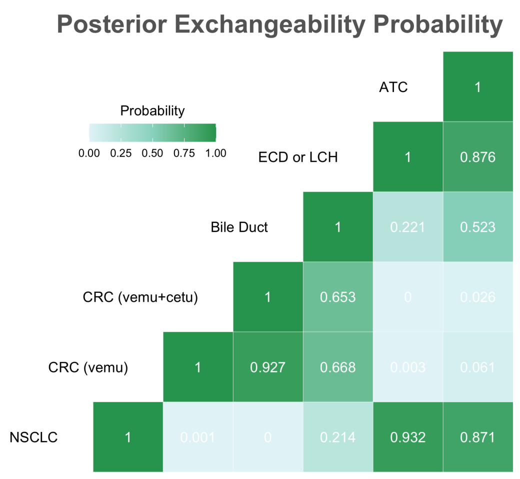


Figure 4: The exchangeogram depicting PEP resulting from analysis of the Vemurafenib study.

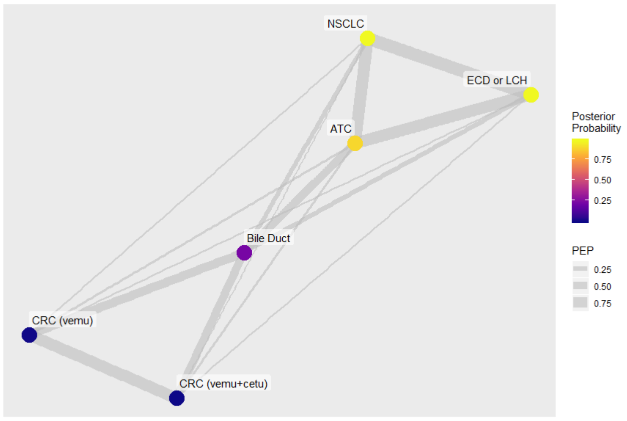


Figure 5: Network graphical representation of PEP resulting from analysis of the Vemurafenib study.

Summary

With the emergence of molecularly targeted therapies, contemporary trials are devised to enroll potentially heterogeneous patient populations defined by a common treatment target. Consequently, characterization of subpopulation heterogeneity has become central to the design and analysis of clinical trials, in oncology in particular. By partitioning the study population into subpopulations that comprise potentially non-exchangeable patient cohorts, the basket design framework can be used to study treatment heterogeneity in a prospective manner. When applied in this context, the Bayesian multisource exchangeability model (MEM) methodology refines the estimation of treatment effectiveness to specific subpopulations. Additionally, the MEM inferential strategy objectively identifies which patient subpopulations should be considered exchangeable and to what extent.

This article introduced the R package **basket** as well as demonstrated its implementation for basket trial analysis using the MEM methodology. An oncology case study using data acquired from a basket trial was presented and used to demonstrate the main functionality of the package. The **basket** package is the first available software package implementing Bayesian analysis with the MEM. The package is being actively maintained and used in ongoing trials.

Acknowledgements: This work was partially supported by Amgen, Inc. as well as The Yale Comprehensive Cancer Center (P30CA016359), and The Case Comprehensive Cancer Center (P30 CA043703).

Bibliography

S. M. Berry, B. P. Carlin, J. J. Lee, and P. Müller. *Bayesian Adaptive Methods for Clinical Trials*. Chapman and Hall/CRC Press, Boca Raton, FL, 2010. [p362]

S. M. Berry, K. R. Broglio, S. Groshen, and D. A. Berry. Bayesian hierarchical modeling of patient subpopulations: Efficient designs of phase II oncology clinical trials. *Clinical Trials*, 10(5):720–734, 2013. [p362]

K. R. Campbell and C. Yau. Probabilistic modeling of bifurcations in single-cell gene expression data using a bayesian mixture of factor analyzers. *Wellcome open research*, 2, 2017. doi: 10.12688/wellcomeopenres.11087.1. URL `http://dx.doi.org/10.12688/wellcomeopenres.11087.1`. [p362]

N. Chen, B. Hobbs, A. Kaizer, and M. J. Kane. *basket: Basket Trial Analysis*, 2019. R package version 0.9.2. [p361]

K. M. Cunanan, M. Gonen, R. Shen, D. M. Hyman, G. J. Riely, C. B. Begg, and A. Iasonos. Basket trials in oncology: A trade-off between complexity and efficiency. *Journal of Clinical Oncology*, 35(3):271–273, 2017a. doi: 10.1200/JCO.2016.69.9751. URL `https://doi.org/10.1200/JCO.2016.69.9751`. PMID: 27893325. [p360]

K. M. Cunanan, A. Iasonos, R. Shen, D. M. Hyman, G. J. Riely, M. Gönen, and C. B. Begg. Specifying the true-and false-positive rates in basket trials. *JCO Precision Oncology*, 1:1–5, 2017b. [p362]

B. G. Durie, J. Harousseau, J. Miguel, J. Blade, B. Barlogie, K. Anderson, M. Gertz, M. Dimopoulos, J. Westin, P. Sonneveld, et al. International uniform response criteria for multiple myeloma. *Leukemia*, 20(9):1467, 2006. [p367]

E. A. Eisenhauer, P. Therasse, J. Bogaerts, L. H. Schwartz, D. Sargent, R. Ford, J. Dancey, S. Arbuck, S. Gwyther, M. Mooney, et al. New response evaluation criteria in solid tumours: revised recist guideline (version 1.1). *European journal of cancer*, 45(2):228–247, 2009. [p367]

B. Freidlin and E. Korn. Borrowing information across subgroups in phase II trials: Is it useful? *Clinical Cancer Research*, 19:1326–1334, 2013. [p362]

A. Gelman, J. B. Carlin, H. S. Stern, and D. B. Rubin. *Bayesian Data Analysis*. Chapman and Hall/CRC Press, Boca Raton, FL, 3rd edition, 2013. [p365]

B. Hobbs, M. Kane, D. Hong, and R. Landin. Statistical challenges posed by uncontrolled master protocols: sensitivity analysis of the vemurafenib study. *Annals of Oncology*, 29(12):2296–2301, 2018. [p367]

B. P. Hobbs and R. Landin. Bayesian basket trial design with exchangeability monitoring. *Statistics in medicine*, 37(25):3557–3572, 2018. [p360, 362, 363, 364, 365]

B. P. Hobbs, B. P. Carlin, and D. J. Sargent. Adaptive adjustment of the randomization ratio using historical control data. *Clinical Trials*, 10:430–440, 2013. [p364]

D. M. Hyman, I. Puzanov, V. Subbiah, J. E. Faris, I. Chau, J.-Y. Blay, J. Wolf, N. S. Raje, E. L. Diamond, A. Hollebecque, et al. Vemurafenib in multiple nonmelanoma cancers with braf v600 mutations. *New England Journal of Medicine*, 373(8):726–736, 2015. [p366]

A. M. Kaizer, J. S. Koopmeiners, and B. P. Hobbs. Bayesian hierarchical modeling based on multisource exchangeability. *Biostatistics*, 19(2):169–184, 2017. [p360, 361, 362, 363]

A. M. Kaizer, B. P. Hobbs, and J. S. Koopmeiners. A multi-source adaptive platform design for testing sequential combinatorial therapeutic strategies. *Biometrics*, 74(3):1082–1094, 2018. [p361]

Q. Mo. *iSeq: Bayesian Hierarchical Modeling of ChIP-seq Data Through Hidden Ising Models*, 2018. R package version 1.34.0. [p362]

T. A. Murray, B. P. Hobbs, and B. P. Carlin. Combining nonexchangeable functional or survival data sources in oncology using generalized mixture commensurate priors. *Annals of Applied Statistics*, 9 (3):1549–1570, 2015. [p364]

V. P. Nia and A. C. Davison. High-dimensional bayesian clustering with variable selection: The R package bclust. *Journal of Statistical Software*, 47(5):1–22, 2012. URL `http://www.jstatsoft.org/v47/i05/`. [p362]

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2019. URL `https://www.R-project.org/`. [p]

R. Samb, K. Khadraoui, P. Belleau, A. Deschênes, L. Lakhal-Chaieb, and A. Droit. Using informative multinomial-dirichlet prior in a t-mixture with reversible jump estimation of nucleosome positions for genome-wide profiling. *Statistical Applications in Genetics and Molecular Biology*, 14, 2015. doi: 10.1515/sagmb-2014-0098. URL `https://doi.org/10.1515/sagmb-2014-0098`. [p362]

R. Savage, E. Cooke, R. Darkins, and Y. Xu. *BHC: Bayesian Hierarchical Clustering*, 2018. R package version 1.34.0. [p362]

R. B. Scharpf, H. Tjelmeland, G. Parmigiani, and A. Nobel. A bayesian model for cross-study differential gene expression. *JASA*, 2009. URL `10.1198/jasa.2009.ap07611`. [p362]

S. Sharifi-Malvajerdi, F. Zhu, C. B. Fogarty, M. P. Fay, R. M. Fairhurst, J. A. Flegg, K. Stepniewska, and D. S. Small. Malaria parasite clearance rate regression: an r software package for a bayesian hierarchical regression model. *Malaria Journal*, 18(1):4, Jan 2019. ISSN 1475-2875. doi: 10.1186/s12936-018-2631-8. URL `https://doi.org/10.1186/s12936-018-2631-8`. [p362]

A. Stocco. Coordinate-based meta-analysis of fmri studies with r. *R Journal*, 6(2), 2014. [p362]

P. Thall, J. Wathen, B. Bekele, R. Champlin, L. Baker, and R. Benjamin. Hierarchical bayesian approaches to phase II trials in diseases with multiple subtypes. *Statistics in Medicine*, 22:763–780, 2003. [p362]

Yang Xiang, S. Gubian, B. Suomela, and J. Hoeng. Generalized simulated annealing for efficient global optimization: the GenSA package for R. *The R Journal Volume 5/1, June 2013*, 2013. URL https://journal.r-project.org/archive/2013/RJ-2013-002/index.html. [p364]

E. Zhang and H. G. Zhang. CRAN Task View: Clinical Trial Design, Monitoring, and Analysis. https://CRAN.R-project.org/view=ClinicalTrials, 2018. Version 2018-06-18. [p361]

Michael J. Kane
School of Public Health
Biostatistics Department
Yale University
New Haven, CT, USA
E-mail: michael.kane@yale.edu

Nan Chen
MD Anderson Cancer Center
The University of Texas
Houston, TX, USA
E-mail: nchen2@mdanderson.org

Alex Kaizer
Colorado School of Public Health
Department of Biostatistics and Informatics
University of Colorado-Anschutz Medical Campus
Aurora, CO, USA
E-mail: alex.kaizer@cuanschutz.edu

Xun Jiang
Amgen Inc. Thousand Oaks, CA, USA
E-mail: xunj@amgen.com

H. Amy Xia
Biostatistics and Design & Innovation
Amgen Inc. Thousand Oaks, CA, USA
E-mail: hxia@amgen.com

Brian Hobbs
Taussig Cancer Center
The Cleveland Clinic
Cleveland, OH, USA
E-mail: HobbsB@ccf.org

OpenLand: Software for Quantitative Analysis and Visualization of Land Use and Cover Change

by Reginal Exavier and Peter Zeilhofer

Abstract There is an increasing availability of spatially explicit, freely available land use and cover (LUC) time series worldwide. Because of the enormous amount of data this represents, the continuous updates and improvements in spatial and temporal resolution and category differentiation, as well as increasingly dynamic and complex changes made, manual data extraction and analysis is highly time consuming, and making software tools available to automatize LUC data assessment is becoming imperative. This paper presents a software developed in R, which combines LUC raster time series data and their transitions, calculates state-of-the-art LUC change indicators, and creates spatio-temporal visualizations, all in a coherent workflow. The functionality of the application developed is demonstrated using an LUC dataset of the Pantanal floodplain contribution area in Central Brazil.

Introduction

Land use and land cover (LUC) monitoring provides key information on the ecological state and biophysical properties of the land surface and is widely used in climatic, hydrological, and ecological modelling (Brovkin et al., 2013; Verburg et al., 2015). Global population growth over the last decades has led to increased rates of LUC change (LUCC), which has affected all major ecosystem services, including biodiversity, climate, and water supply, and has altered carbon cycling (Ballantyne et al., 2015; Nelson et al., 2010; Song et al., 2018).

The importance of these applications is supported by the continuous expansion of remote sensing data acquisition programs, making freely available an increasing amount of spatially explicit LUC time series data worldwide (Prestele et al., 2016). The huge volume of these datasets, their timely actualization, ever-increasing spatial resolution and category differentiation, and a variety of data formats, render their manual extraction and analysis increasingly time consuming. This is especially true when trying to compare or harmonize LUC datasets of the same study area generated from different sources, with different methodological approaches across different spatial and temporal scales, in order to output the factual LUC in a region of interest (Yang et al., 2017).

Human interference on LUC is complex, and the intensity and frequency of LUC transitions are increasing. Therefore, the development of sound LUC models is highly dependent on a deep understanding of past and ongoing LUCC processes (Müller and Munroe, 2014). This means that LUC patterns have to be constantly reviewed to underpin our understanding of these processes (Lambin, 1997; Lambin and Geist, 2006), develop a baseline analysis for projections of future LUC (Hurtt et al., 2011), and to construct, calibrate and validate LUCC simulations (Prestele et al., 2016). Such efforts require tools for data extraction, pre-processing, visualization, and calculation of LUC metrics to relieve the burden from labour-intensive and time-consuming manual data processing and analysis (Yu et al., 2019). If the procedures implemented to do so follow standards in land cover characterization, and use formalized methodological approaches in LUC analysis, the resulting analytics become more transparent, robust, and auditable (Herold et al., 2006; Müller and Munroe, 2014; Yang et al., 2017).

Aldwaik and Pontius (2012) developed an approach called Intensity Analysis (IA), which examines changes in LUC categories by comparing the intensity of change between categories during a given time interval with a hypothesized uniform change intensity. Since then, several case studies have emphasized the potential of Intensity Analysis to synthetize complex LUCC under different spatial and temporal scales, such as urban environments (Akinyemi et al., 2017; Subasinghe et al., 2016), regional studies (Melo et al., 2018; Mwangi et al., 2017; de Souza et al., 2017), and country-wide comparisons (Chaudhuri and Mishra, 2016). Furthermore, Huang et al. (2018) concluded that intensity analysis metrics outperform other indicators of land use dynamics in the comparison between candidate regions, while Varga et al. (2019) showed that IA metrics are a helpful tool for assessing the quality of LUC modelling outputs.

In IA, the uniform change intensity of a time period is compared with the observed intensities of transitions between LUC categories. The calculations are carried out using transition matrices between LUC categories, assessing specific: (i) time intervals, (ii) categories or classes and (iii) types of transitions. The first assessment level examines in which time intervals global annual change rates are faster, slower or comparable to an average rate of change. The second level determines which category transitions are relatively dormant or more active within a given time interval, based on an analysis of gross change. The gross gains and losses for each category in a given time interval are compared with an averaged, uniform annual change. The third level of assessment seeks to identify which transitions are particularly intensive during the time interval considered. In order to do so each transition is compared with a uniform transition intensity. The mathematical notations used for the IA indicators measuring size and intensities of temporal changes among LUC categories (Equations 1-8) can be found in Aldwaik and Pontius (2012) and Aldwaik and Pontius (2013).

Analytical tools for the analysis of LUC time series are commonly available in compiled languages and distributed as software packages or extensions to proprietary geographic information systems such as ArcGIS or Idrisi (Moulds et al., 2015). Consequently, the source code for such tools, used for land use change analysis and modelling, is often unavailable (Rosa et al., 2014). This makes adopting the applications of new approaches and reproducing scientific results difficult (Morin et al., 2012; Peng, 2011). GIS software has made widely available spatially explicit visualization capabilities for multi-temporal LUC, which is crucial for documentation and analysis of LUC distribution. However, maps can become blurred and non-interpretable when extensive time series or complex landscapes with different category transitions need to be analysed. This is particularly true when multi-category changes have to be analysed at multiple time steps to improve the understanding of landscape processes, in which case non-spatial forms of LUCC will have to be explored.

By contrast, the availability of packages to perform analyses of LUC categories in time series data is limited. **intensity.analysis**, developed by Pontius Jr. and Khallaghi (2019), is an application of IA, with functionality limited to the calculation of the three IA assessment levels described above and their plotting; this package does not allow any personalization. Little pre-processing capability is available. If needed, input rasters cannot be checked for spatial and thematic consistency, and they have to be manually imported to R and stored. No other commonly used LUC metrics can be calculated or plotted. Tools to visualize LUC changes are limited to standard plots of the three IA analysis levels. **lulcc** (Moulds et al., 2015) is another package developed for LUC change analysis and focuses on the application of tools for LUC change modelling.

This highlights the need for flexible and comprehensive tools for multipurpose analyses of complex LUC time series data. This paper aims to demonstrate that we have developed a solution, available as freeware, which allows the agile consistency analysis, extraction, pre-processing, analysis, and visualization of multiresolution time series data in a straightforward workflow, thereby increasing productivity in information extraction for an improved understanding of LUCC processes at different spatial scales. Originating primarily from the application of the intensity analysis method, the formatted tabular representation of multiple transition steps can be a valuable tool to support the quick calculation of LUCC indicators.

Conceptual overview

The software was developed to provide comprehensive support for LUCC analysis over time and implement the intensity analysis conceptual approach proposed by Aldwaik and Pontius (2012), and further described in Aldwaik and Pontius (2013).

The source language of the **OpenLand** package (Exavier and Zeilhofer, 2020) is R, and it was conceived using the integrated development environment (IDE) RStudio (RStudio Team, 2016). For full functionality, **OpenLand** depends on selected, third-party functions of the **raster**, **dplyr** and **tidyr** R packages (Hijmans, 2019; Wickham et al., 2019; Wickham and Henry, 2019) (Fig. 1). Visualization tools make use of **ggplot2**, **circlize**, **gridExtra** and **networkD3** R packages (Allaire et al., 2017; Auguie, 2017; Gu et al., 2014; Wickham, 2016). Designed as an iterative workflow, the pre-processing of a raw LUC time series is followed by the extraction and visualization of single or multistep LUC transitions and/or a complete IA.

The main processing workflow should begin with a consistency check of the input files, which have to be a sequence of LUC maps in a tif format (Fig. 1). In the following step, a contingency table of LUC transitions for each time step can be calculated. Two complementary analysis workflows are then available. First, a set of miscellaneous non-spatial visualization tools allows for a quick screening of LUC dynamics. Second, a complete intensity analysis (Aldwaik and Pontius, 2012, 2013) can be implemented, including tools to visualize change intensities over time, category and transition levels. Ancillary functionalities include allow for extraction and spatial visualization of LUCC frequencies.

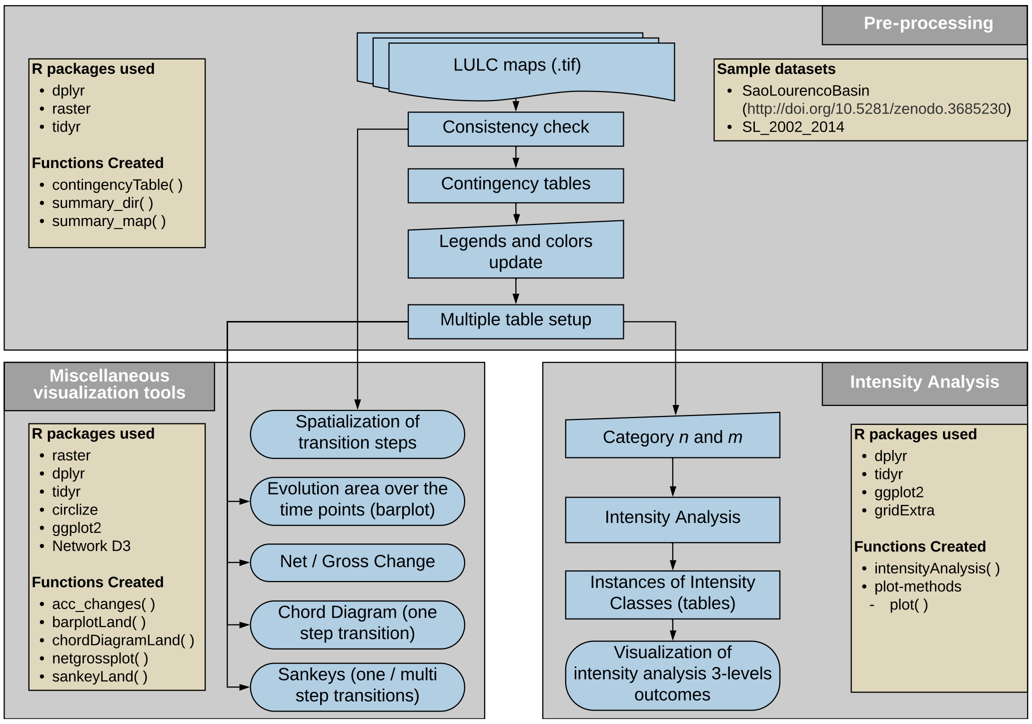


Figure 1: Conceptual overview of the OpenLand package showing reused R functionalities. The package is composed by three major blocks for *i)* pre-processing of raster time series and calculus of LUC transitions, *ii)* spatial and non-spatial visualization of LUCC and *iii)* intensity analysis including tailored visualization tools.

Functionality and implementation

The São Lourenço river basin example dataset

The **OpenLand** functionality is demonstrated using an LUC dataset of the São Lourenço river basin, of major importance to the Pantanal wetland into which it flows. The data is as provided in the 4th edition of the Monitoring of Changes in Land cover and Land Use in the Upper Paraguay River Basin - Brazilian portion - Review Period: 2012 to 2014 (Instituto SOS Pantanal and WWF-Brasil, 2015), and the time series is composed of five LUC maps (2002, 2008, 2010, 2012 and 2014). The study area is located in the Cerrado savanna biome, in the southeastern corner of the Brazilian state of Mato Grosso, and covers approximately 22,400 km^2. Some level of LUCC has occurred in about 12% of the area over the past 12 years, including some deforestation and intensification of existing agricultural uses. In order to be processed using the **OpenLand** package, the original multi-year shapefile was clipped to the extent of the São Lourenço basin, transformed into rasters and saved as a 5-layer `RasterStack`; it is available from a public repository (https://doi.org/10.5281/zenodo.3685229) as an `.RDA` file which can be loaded into R.

```
# Installing the released version of OpenLand from CRAN
install.packages("OpenLand")

# Loading the OpenLand package
library(OpenLand)

# downloading the SaoLourencoBasin multi-layer raster and make it available into R
url <- "https://zenodo.org/record/3685230/files/SaoLourencoBasin.rda?download=1"

temp <- tempfile()
download.file(url, temp, mode = "wb")
load(temp)

# looking on the metadata of the example dataset
SaoLourencoBasin
```

```
#> class      : RasterStack
#> dimensions : 6372, 6546, 41711112, 5  (nrow, ncol, ncell, nlayers)
#> resolution : 30, 30  (x, y)
#> extent     : 654007.5, 850387.5, 8099064, 8290224  (xmin, xmax, ymin, ymax)
#> crs        : +proj=utm +zone=21 +south +ellps=GRS80 +units=m +no_defs
#> names      : landscape_2002, landscape_2008, landscape_2010, ..., landscape_2014
#> min values :              2,              2,              2, ...,              2
#> max values :             13,             13,             13, ...,             13
```

To visualize the output of the LUC analysis in **OpenLand**, we simplified the legend of the original dataset and used the following 11 LUC categories for the basin: forest formation (FF), three Cerrado savanna formations (SF, SA, SG), anthropogenized vegetation (aa), i.e. mostly altered Cerrado formations used for grazing, composed of natural species, cattle farming (Ap), crop farming (Ac), mining areas (Im), urban areas (Iu), water bodies (Agua), and reforestation (R) (Table 1).

Table 1: The original legend of LUC classes and categories from Instituto SOS Pantanal and WWF-Brasil (2015) including colour coding.

Pixel Value	Legend	Class	Category	Colour
2	Ap	Anthropogenic	Cattle farming	#DDCC77
3	FF	Natural	Forest formation	#117733
4	SA	Natural	Park savanna	#44AA99
5	SG	Natural	Gramineous savanna	#88CCEE
7	aa	Anthropogenic	Anthropogenized vegetation	#CC6677
8	SF	Natural	Wooded savanna	#999933
9	Agua	Natural	Water bodies	#332288
10	Iu	Anthropogenic	Urban areas	#AA4499
11	Ac	Anthropogenic	Crop farming	#661100
12	R	Anthropogenic	Reforestation	#882255
13	Im	Anthropogenic	Mining areas	#6699CC

Consistency check and data extraction from raster time series

Two auxiliary functions allow users to check for consistency in the input gridded LUC time series, including extent, projection, cell resolution and categories. The `summary_map()` function returns the number of pixels in each category for each single raster layer, whereas the `summary_dir()` function lists the spatial extent, spatial resolution, cartographic projection and the category range of a set of LUC maps.

For the initial spatial screening of the time series, the `acc_changes()` function determines the number of LUC transitions during the entire time interval of the series. The results of percentage area by transition frequencies in the study area are stored in a table and a grid layer is generated, which can be plotted (Fig. 2) using for example the `tmap` package.

```
# the acc_changes() function, with the SaoLourencoBasin dataset
SL_changes <- acc_changes(SaoLourencoBasin)
SL_changes

#> [[1]]
#> class      : RasterLayer
#> dimensions : 6372, 6546, 41711112  (nrow, ncol, ncell)
#> resolution : 30, 30  (x, y)
#> extent     : 654007.5, 850387.5, 8099064, 8290224  (xmin, xmax, ymin, ymax)
#> crs        : +proj=utm +zone=21 +south +ellps=GRS80 +units=m +no_defs
#> names      : layer
#> values     : 0, 2  (min, max)
#>
#>
#> [[2]]
#> # A tibble: 3 x 3
#>   PxValue       Qt Percent
#>     <int>    <int>   <dbl>
#> 1       0 21819779    87.6
#> 2       1  2787995   11.2
#> 3       2   301086    1.21
```

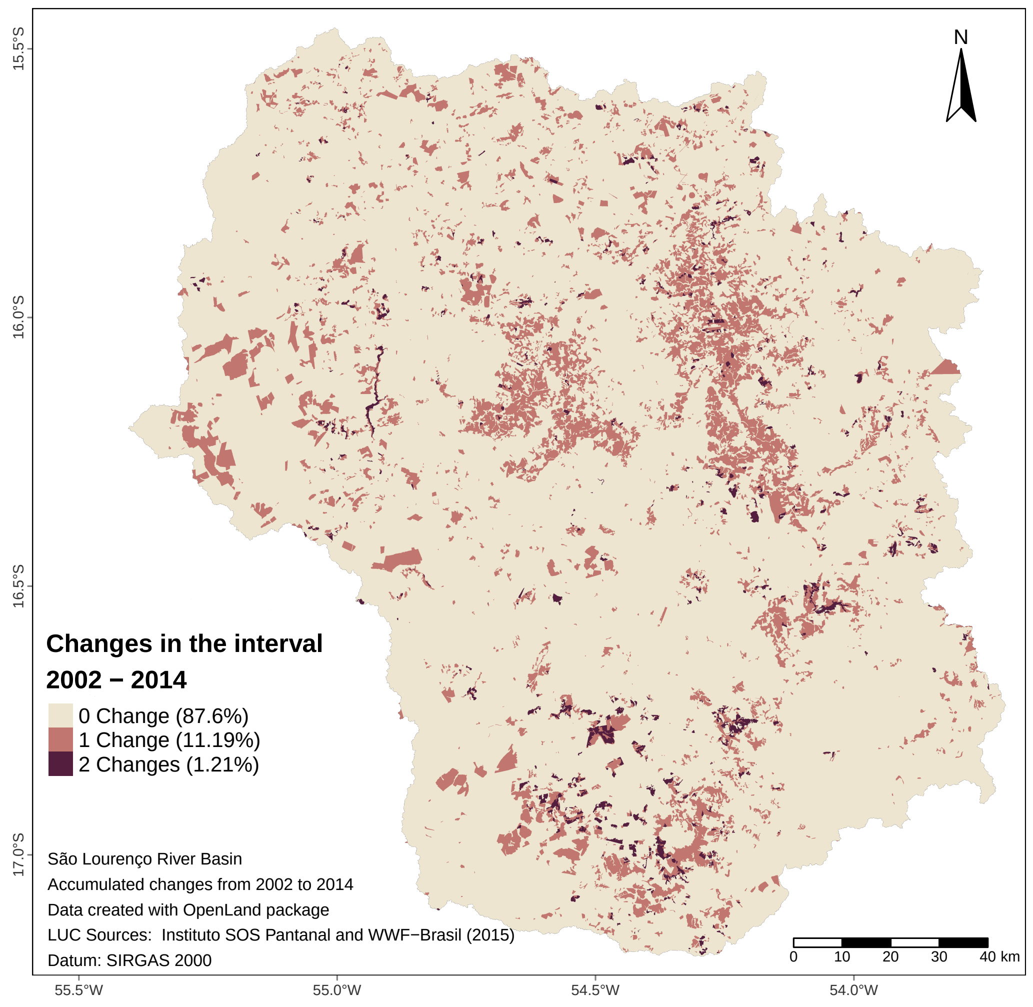


Figure 2: Number of LUCC between 2002 and 2014 at four time points (2002, 2008, 2010, 2012, 2014) in the São Lourenço river basin (Instituto SOS Pantanal and WWF-Brasil, 2015). LUCC occurred throughout the basin, with one-step changes concentrated in the central-northern regions and two-step changes in the central south.

As most deforestation occurred in the 20th century, unchanged areas totalled 87.6%. Approximately 35% of the watershed had already been deforested in 1985, versus 46% in 2014 (Project MapBiomas, 2019). 11.19% showed a unique alteration and 1.21% a two-folded alteration for the five time points of the input series considered. All further analytical and visualization tools are based on the `contingencyTable()` function, which builds a matrix of transitions between LUC categories according to the temporal resolution of the original time series. Multiple grid scanning by `contingencyTable()` returns 5 objects: `lulc_Multistep`, `lulc_Onestep`, `tb_legend`, `totalArea`, `totalInterval`. The first two objects are contingency tables; the first (`lulc_Multistep`) takes into account the grid cells of the entire time series, whereas the second (`lulc_Onestep`) calculates LUC transitions only between the first and last year of the series. The third object (`tb_legend`) is a table containing the category name associated with pixel values and a colour scheme. As category values and colours are initially created randomly, their values must be edited to produce meaningful plot legends and colour schemes. The fourth object (`totalArea`) is a table containing the extent of the study area in km^2 and in pixel units. The fifth table (`totalInterval`) stores the range of years between the first ($Y_{t=1}$) and last year (Y_T) of the series. Table 2 presents the fields created, together with their format, description and labelling as per a table output by the `contingencyTable()` function.

As mentioned, the **tb_legend** object must be edited with the real category names and colours associated with the category values. In our case, the category names and colours follow the conventions given by (Instituto SOS Pantanal and WWF-Brasil, 2015) (access document here, page 17) like the values in (Table 1).

Table 2: Structure of the contingency table that stores LUC transitions at chosen time intervals.

$[Y_t, Y_{t+1}]$	Category_i	Category_j	C_{tij} (km^2)	C_{tij} (pixel)	$Y_{t+1} - Y_t$	Y_t	Y_{t+1}
chr	int	int	dbl	int	int	int	int
Period of analysis from time point *t* to time point *t+1*	A category at interval's initial time point	A category at interval's final time point	Number of elements in km^2 that transits from category *i* to category *j*	Number of elements in pixel that transits from category *i* to category *j*	Interval in years between time point *t* and time point *t+1*	Initial Year of the interval	Final Year of the interval
Period	**From**	**To**	**km2**	**QtPixel**	**Interval**	**yearFrom**	**yearTo**

Users should be aware that acc_changes() and contingencyTable() process the entire input time series analyzing successive raster pairs. Processing time of contingencyTable(), the computationally most demanding function of **OpenLand** may range between 3 and 7 minutes for the SaoLourencoBasin dataset if using common desktop computers (8 MB Ram, i3-i7 processors, Windows 64bits versions).

```
## creating the contingency table
SL_2002_2014 <- contingencyTable(input_raster = SaoLourencoBasin,
                                 pixelresolution = 30)
names(SL_2002_2014)
```

```
#> [1] "lulc_Multistep" "lulc_Onestep"   "tb_legend"      "totalArea"
#> [5] "totalInterval"
```

```
## editing the category names
SL_2002_2014$tb_legend$categoryName <- factor(c("Ap", "FF", "SA", "SG", "aa", "SF",
                                              "Agua", "Iu", "Ac", "R", "Im"),
                                       levels = c("FF", "SF", "SA", "SG", "aa", "Ap",
                                                  "Ac", "Im", "Iu", "Agua", "R"))

## adding the colours by the same order of the legend
SL_2002_2014$tb_legend$color <- c("#DDCC77", "#117733", "#44AA99", "#88CCEE",
                                  "#CC6677", "#999933", "#332288", "#AA4499",
                                  "#661100", "#882255", "#6699CC")
```

Miscellaneous non-spatial visualization tools

Evolution of LUC areas

Exploratory data analysis based on the contingencyTable() function may begin with the visualization of the absolute and/or percentage area of each LUC category at each time point using a grouped bar plot (Fig. 3), in order to show the evolution of the LUC categories at each time point of the series.

```
barplotLand(dataset = SL_2002_2014$lulc_Multistep,
            legendtable = SL_2002_2014$tb_legend,
            xlab = "Year",
            ylab = bquote("Area (" ~ km^2~ ")"),
            area_km2 = TRUE)
```

Net and gross changes

For the analysis of long time series with high temporal resolution, information extraction from evolution bar plots may become demanding and alterations between categories through time points are only as net changes. For a category-wise, simultaneous assessment of net and gross changes, multistep transitions can be balanced through a stacked bar chart (Fig. 4). In the São Lourenço river basin, all natural vegetation categories suffered an equal net and gross loss (FF, SF, SA, SG) between 2002 and 2014. In contrast, mining (Im) and urbanized (Iu) areas as well as waterbodies (Agua) and reforestation (R) had equal amounts of net and gross gains. Principally anthropogenized vegetation (aa) and pastures showed differences in net and gross changes, pointing to more complex underlying LUCC processes.

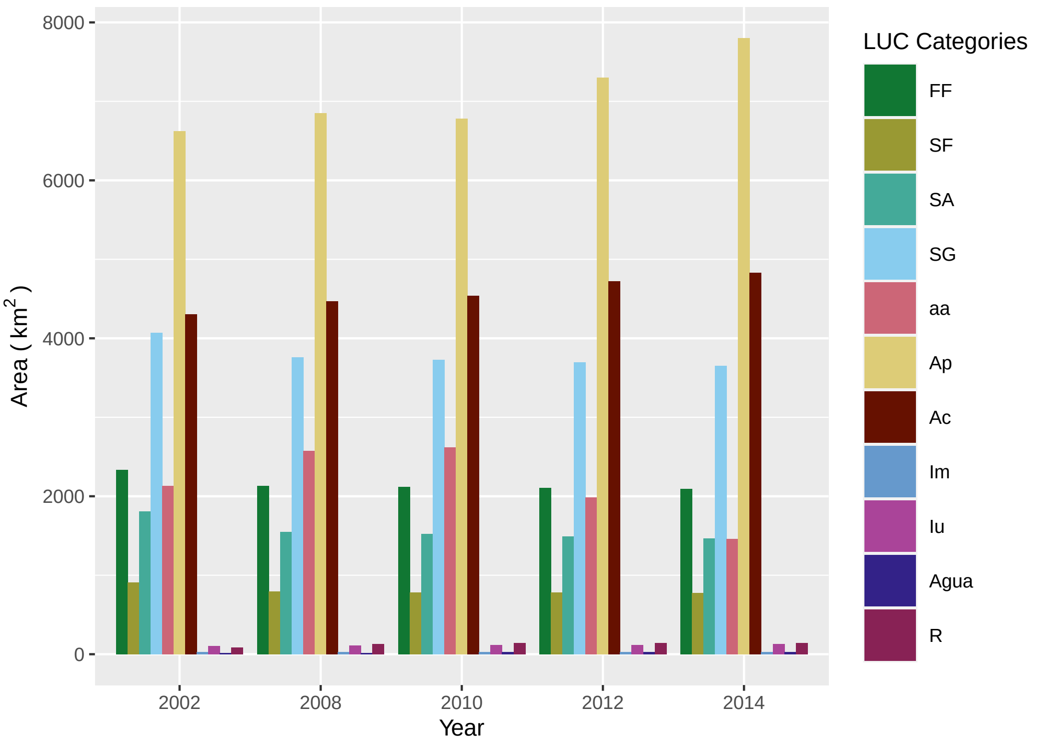


Figure 3: Evolution bar plot quantifying LUC categories at the five time points between 2002 and 2014, highlighting the increase of cattle farming (Ap) in the river basin.

```
netgrossplot(dataset = SL_2002_2014$lulc_Multistep,
             legendtable = SL_2002_2014$tb_legend,
             xlab = "LUC Category",
             ylab = bquote("Area (" ~ km^2 ~ ")"),
             changesLabel = c(GC = "Gross Changes", NG = "Net Gain", NL = "Net Loss"),
             color = c(GC = "#e0c2a2", NG = "#541f3f", NL = "#c1766f"),
             area_km2 = TRUE)
```

To further explore specifically those non-linear transitions, single-step Chord (Fig. 5) and Sankey diagrams (Fig. 6a and 6b) can be composed for each time point in the series. Considering the entire observation period, the major gross change was from anthropogenized areas (aa) to cattle farming (Ap). Both the Chord and Sankey diagrams show however that pastures did not directly gain from clear-cut deforestation, but from the previous degradation of natural vegetation categories principally until 2008 (FF, SF, SA, SG), and a subsequent transition from aa to Ap between 2010 and 2014.

Chord diagram (2002 – 2014)

```
chordDiagramLand(dataset = SL_2002_2014$lulc_Onestep,
                 legendtable = SL_2002_2014$tb_legend,
                 area_km2 = TRUE)
```

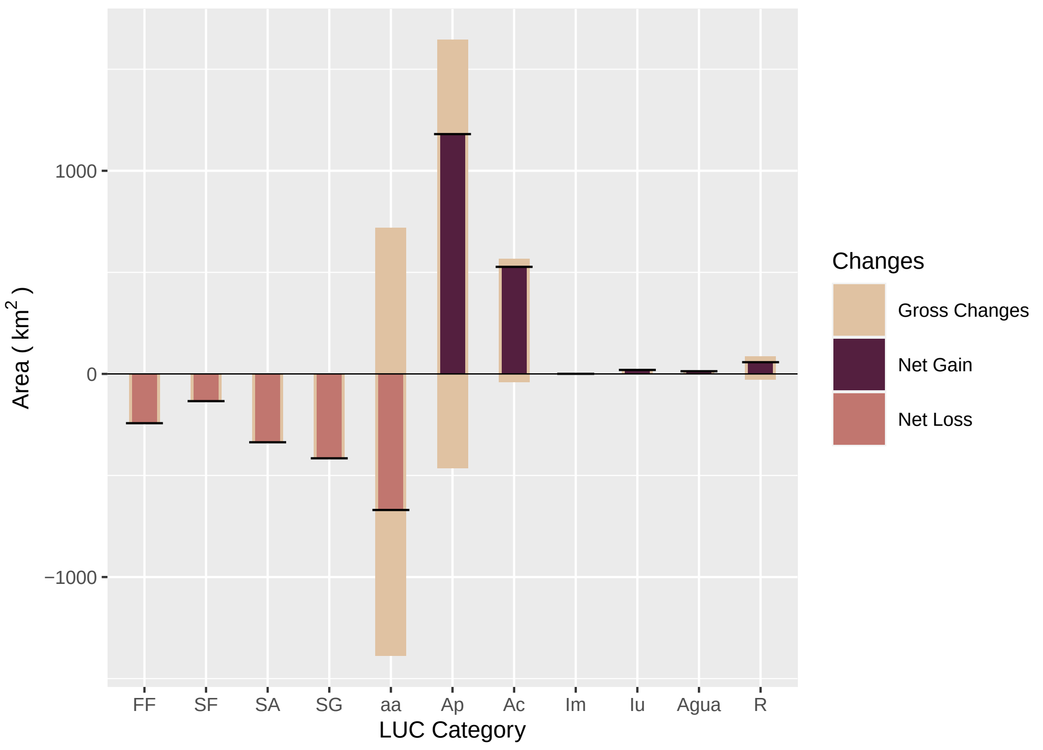


Figure 4: Combined stacked bar plot of gross changes, net gains and net losses of LUC categories (2002 – 2014). Anthropogenized vegetation (aa) and cattle farming (Ap) experimented both gross gains and losses, however net losses and net gains, respectively.

Single-step Sankey diagram (2002 – 2014)

```
sankeyLand(dataset = SL_2002_2014$lulc_Onestep,
           legendtable = SL_2002_2014$tb_legend)
```

Multistep sankey diagram

If between-category transitions have to be visualized simultaneously for the entire time series, a multistep version of the Sankey plot can be output (Fig. 6b).

```
sankeyLand(dataset = SL_2002_2014$lulc_Multistep,
           legendtable = SL_2002_2014$tb_legend)
```

The `sankeyLand()` function returns html output, as it depends on the **networkD3** R package which uses such format as default.

Intensity analysis

Intensity Analysis (IA) is a quantitative method for the analysis of LUC maps over several time steps, using cross-tabulation matrices, where each matrix summarizes the LUC change for each time interval. IA evaluates the deviation between observed change intensity and hypothesized uniform change intensity in three levels. Thereby, each level details information given by the previous analysis level. First, the **interval level** indicates how the size and rate of change vary over time intervals. Second, the **category level** examines for each time interval how the size and intensity of gross losses and gross gains in each category vary across categories for each time interval. Third, the **transition level** determines the size and intensity of each transition from one category to another during each time interval. At each level, the method also tests for stationarity of patterns across time intervals (Aldwaik and Pontius, 2012). In the **OpenLand** package, the `intensityAnalysis()` function computes the three levels of analysis. It requires the object returned by the `contingenceTable()` function and that the user predefines two LUC categories `n` and `m`. Generally, `n` is a target category that experienced relevant gains and `m` a category with important losses.

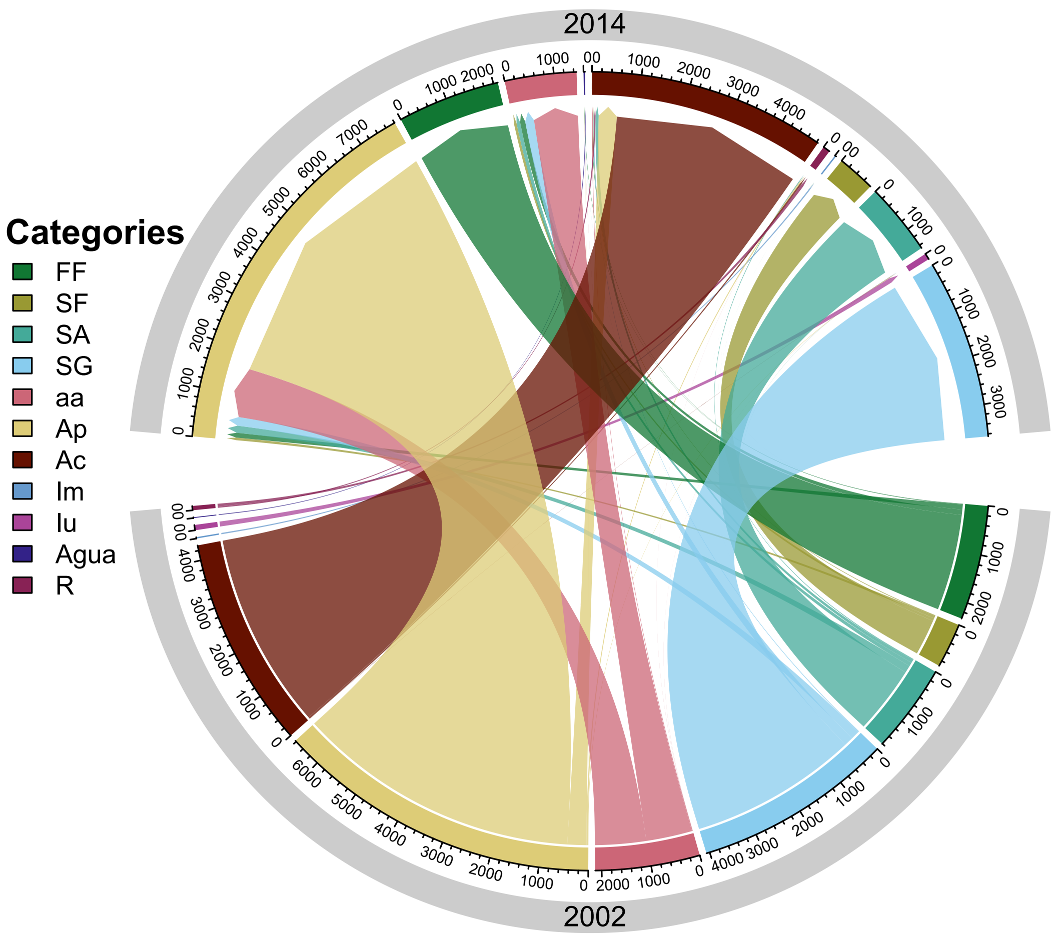


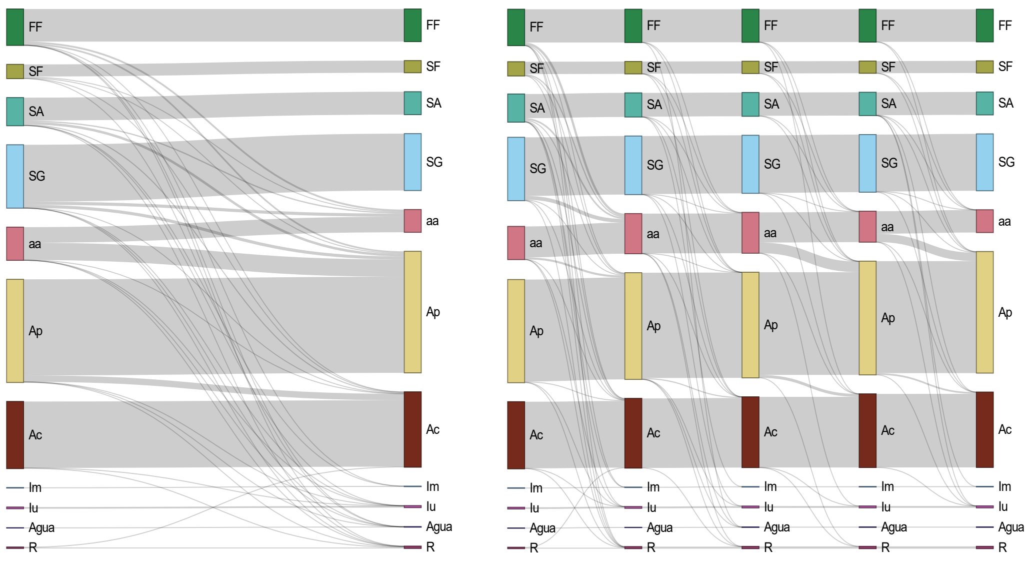
Figure 5: Chord diagram of single-step transitions between LUC categories (2002 – 2014). The major change during the period was from anthropogenized vegetation (aa) to cattle farming (Ap).

```
testSL <- intensityAnalysis(dataset = SL_2002_2014, category_n = "Ap",
                             category_m = "SG", area_km2 = TRUE)

# it returns a list with 6 objects
names(testSL)

#> [1] "lulc_table"           "interval_lvl"         "category_lvlGain"
#> [4] "category_lvlLoss"     "transition_lvlGain_n" "transition_lvlLoss_m"
```

The `intensityAnalysis()` function returns 6 objects: `lulc_table`, `interval_lvl`, `category_lvlGain`, `category_lvlLoss`, `transition_lvlGain_n`, `transition_lvlLoss_m`. The object-oriented approach adopted here allowed us to set specific methods for plotting the intensity objects. Specifically, we used the S4 class, which requires the formal definition of classes and methods (Chambers, 2008). The first object is a contingency table similar to the `lulc_Multistep` object with the unique difference that the columns `From` and `To` are replaced by their appropriate denominations in the LUC legend.

The second object ***interval_lvl*** is an `"Interval"` object, the third ***category_lvlGain*** and the fourth ***category_lvlLoss*** are `"Category"` objects, while the fifth ***transition_lvlGain_n*** and the ***transition_lvlLoss_m*** are `"Transition"` objects.

An `"Interval"` object has one slot containing a table of the **interval level** results (*(S_t equation (1) and U equation (2) values)*. A `"Category"` object has three slots: the first contains the colour associated with the legend item as name attribute, the second slot contains a table of the **category level** results *(gain (G_{tj}) equation (3) or loss (L_{ti}) equation (4) values)*, and the third slot contains a table storing the results of a stationarity test. A `"Transition"` object also has three slots: the first contains the colour associated with the respective legend item defined as name attribute, the second slot contains a table of the **transition level** results *(gain n (R_{tin} equation (5) and W_{tn} equation (6)) or loss m (Q_{tmj} equation (7) and V_{tm} equation (8)) values)*. The third slot contains a table storing the results of a stationarity test. Aldwaik and Pontius (2012) consider a case stationary only when the intensities for all time intervals



(a) Onestep transition 2002 - 2014 **(b)** Multistep transition 2002-2008-2010-2012-2014

Figure 6: Sankey plots of single- and multistep transitions between LUC categories (2002 – 2014). Major overall transition was from anthropogenized vegetation (aa) to cattle farming (Ap) (a), but aa received important contributions by the loss of natural vegetation classes (FF, SF, SA, SG) as well, principally from 2002 to 2008 (b).

are on one side of the uniform intensity, i.e. they are consistently either smaller or larger than the uniform rate over the entire period.

$$S_t = \frac{\sum_{j=1}^{J}\left[\left(\sum_{i=1}^{J} C_{tij}\right) - C_{tjj}\right]}{\left(Y_{t+1} - Y_t\right)\left(\sum_{j=1}^{J}\sum_{i=1}^{J} C_{tij}\right)} \times 100\% \tag{1}$$

$$U = \frac{\sum_{t=1}^{T-1}\left\{\left(Y_{t+1} - Y_t\right)\sum_{j=1}^{J}\left[\left(\sum_{i=1}^{J} C_{tij}\right) - C_{tjj}\right]\right\}}{\left(Y_T - Y_1\right)\sum_{t=1}^{T-1}\left[\left(Y_{t+1} - Y_t\right)\left(\sum_{j=1}^{J}\sum_{i=1}^{J} C_{tij}\right)\right]} \times 100\% \tag{2}$$

$$G_{tj} = \frac{\left[\left(\sum_{i=1}^{J} C_{tij}\right) - C_{tjj}\right] / \left(Y_{t+1} - Y_t\right)}{\sum_{i=1}^{J} C_{tij})} \times 100\% \tag{3}$$

$$L_{ti} = \frac{\left[\left(\sum_{j=1}^{J} C_{tij}\right) - C_{tii}\right] / \left(Y_{t+1} - Y_t\right)}{\sum_{j=1}^{J} C_{tij})} \times 100\% \tag{4}$$

$$R_{tin} = \frac{C_{tin} / \left(Y_{t+1} - Y_t\right)}{\sum_{j=1}^{J} C_{tij})} \times 100\% \tag{5}$$

$$W_{tn} = \frac{\left[\left(\sum_{i=1}^{J} C_{tin}\right) - C_{tnn}\right] / \left(Y_{t+1} - Y_t\right)}{\sum_{j=1}^{J}\left[\left(\sum_{i=1}^{J} C_{tij}\right) - C_{tnj}\right]} \times 100\% \tag{6}$$

$$Q_{tmj} = \frac{C_{tmj} / \left(Y_{t+1} - Y_t\right)}{\sum_{i=1}^{J} C_{tij}} \times 100\% \tag{7}$$

$$V_{tm} = \frac{\left[\left(\sum_{j=1}^{J} C_{tmj}\right) - C_{tmm}\right] / \left(Y_{t+1} - Y_t\right)}{\sum_{i=1}^{J}\left[\left(\sum_{j=1}^{J} C_{tij}\right) - C_{tim}\right]} \times 100\% \tag{8}$$

Graphs (output of the intensity analysis)

Visualizations of the IA results are obtained with the `plot(intensity-object)` method. For more details on the function arguments, please see the documentation of the `plot()` method implemented.

Interval Level

The IA interval level is a measure of the overall rate of LUC changes over consecutive time intervals in the series. The plot (Fig. 7) has two sides: on the left the percentage of change during the corresponding time step, and on the right the percentage of change per year showing the reference line of the Uniform rate.

```
plot(testSL$interval_lvl,
     labels = c(leftlabel = "Interval change area (%)",
                rightlabel = "Annual change area (%)"),
     marginplot = c(-8, 0), leg_curv = c(x = .3, y = .1),
     color_bar = c(fast = "#541f3f", slow =  "#c1766f", area = "#888888"))
```

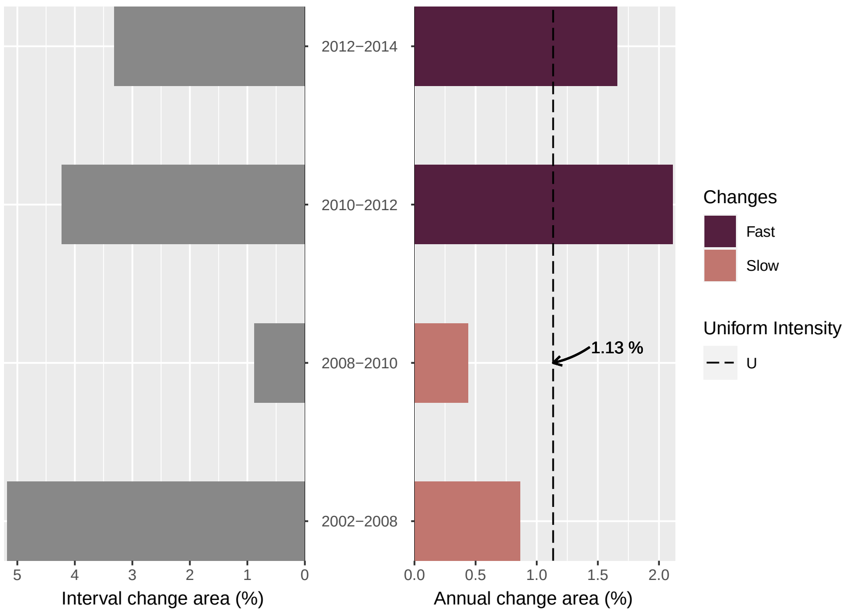


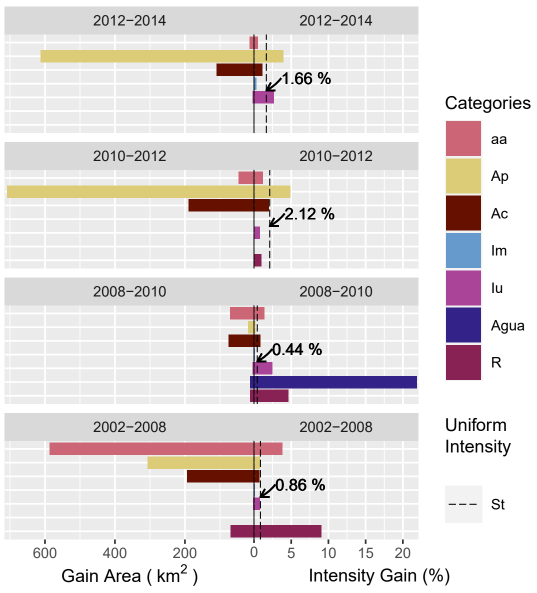
Figure 7: IA Interval Level plot of LUCC for the time intervals of the sample data set. Left-side bars represent LUCC during the time interval and right-side bars show the annualized rates. Strongest annual LUCC occurred between 2010 and 2012.

In the São Lourenço river basin, the interval level plot shows that LUC change has accelerated over the last decade, with a peak between 2010 and 2012, when the actual rate of change was almost double the Uniform rate. This recent acceleration in LUC in the Cerrado biome has been interpreted partially as a spillover effect of conservation efforts in the Amazon basin (Dou et al., 2018).

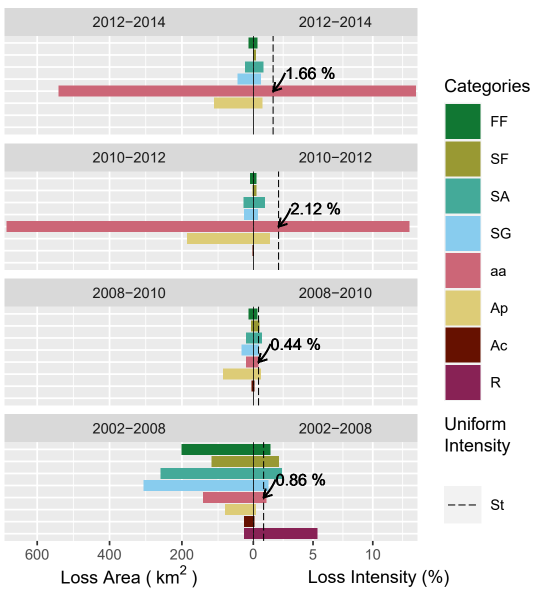
Category Level (Gain and Loss Area)

After the analysis of LUC change intensity independently from LUC categories, the category level allows further examination into which land categories are relatively dormant versus active in a given time interval and whether this pattern is stable across time intervals (Aldwaik and Pontius, 2012).

```
# Gain area
plot(testSL$category_lvlGain,
     labels = c(leftlabel = bquote("Gain Area (" ~ km ^ 2 ~ ")"),
                rightlabel = "Intensity Gain (%)"),
     marginplot = c(.3, .3), leg_curv = c(x = 1, y = .5))

# Loss area
plot(testSL$category_lvlLoss,
     labels = c(leftlabel = bquote("Loss Area (" ~ km ^ 2 ~ ")"),
                rightlabel = "Loss Intensity (%)"),
     marginplot = c(.3, .3), leg_curv = c(x = 1, y = .5))
```



(a) Gain area



(b) Loss area

Figure 8: IA Category level (a) gain and (b) loss plots. Cattle farming (Ap), for example, had the highest areal gain of all categories (2010 – 2012), whereas water bodies had a very high intensity gain between 2008 and 2010 due to the implantation of a large hydropower plant reservoir.

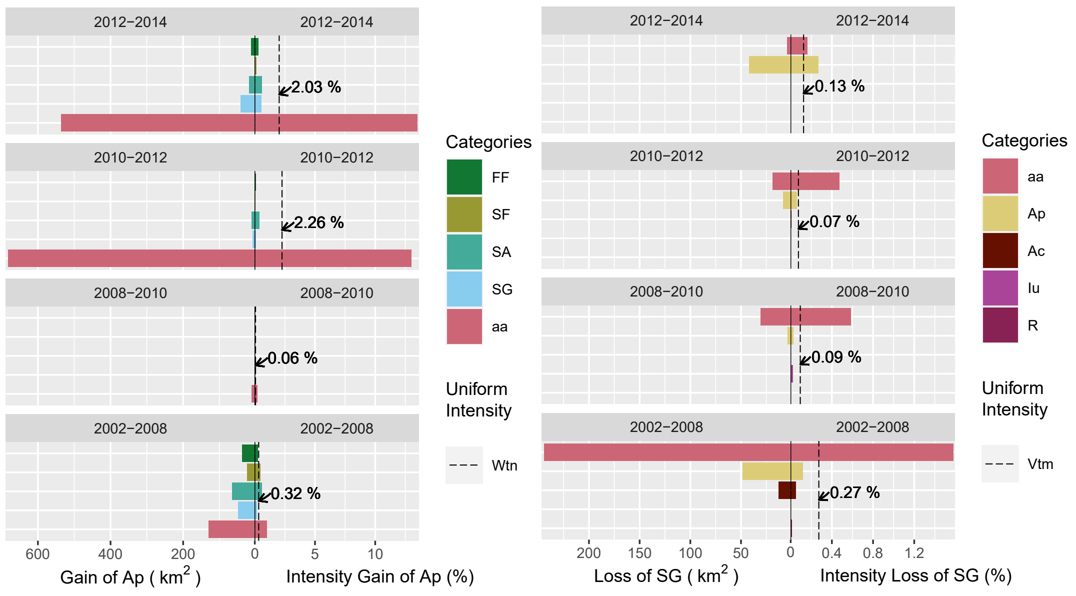
To facilitate legibility, we chose to split the category level plots (Fig. 8) into area gains (Fig. 8a) and losses (Fig. 8b). Area gains in land categories (R) and (aa) were more intense over the first six-year period (2002-2008) than they were at any other subsequent time point. The very intense area gain in water bodies during the second time interval (2008-2010) corresponds to the São Lourenço hydropower plant reservoir being filled. In the third and fourth intervals, the expansion of pasture areas (Ap) was more intense than during previous time steps. In parallel, the land category (aa) is in sharp decline.

Transition level (gain of the category `n` "Ap" and loss of the category `m` "SG")

In the transition level, the analysis focuses on the intensity of gain of a particular category `n` from all the individual categories in the landscape and/or on the intensity of loss of a particular category `m` to all the individual categories in the landscape for each time interval.

```
# Gain of the category `n` "Ap"
plot(testSL$transition_lvlGain_n,
     labels = c(leftlabel = bquote("Gain of Ap (" ~ km^2 ~ ")"),
                rightlabel = "Intensity Gain of Ap (%)"),
     marginplot = c(.3, .3),
     leg_curv = c(x = 1, y = .2))

# Loss of the category `m` "SG"
plot(testSL$transition_lvlLoss_m,
     labels = c(leftlabel = bquote("Loss of SG (" ~ km^2 ~ ")"),
                rightlabel = "Intensity Loss of SG (%)"),
     marginplot = c(.3, .3),
     leg_curv = c(x = .1, y = .4))
```

The area gains in the (Ap) category and losses in the SG category of the SaoLourencoBasin dataset are used here as an example. Results in (Fig. 9a) show that areas in (Ap) were principally gained from the aa category, and that this transition was particularly intense in the third and fourth time periods. Meanwhile, area losses in the (SG) savannah formation persisted over the entire span of the time series (Fig. 9b). However, there was a change in the land uses these areas were lost to: during the first three intervals, (SG) was lost principally to (aa), while in the last time interval, the loss was more intensely due directly to the (Ap) category.



(a) Gain of category n (Ap) **(b)** Loss of category m (SG)

Figure 9: IA Transition level plots of cattle farming (Ap) gains and Gramineous savanna (SG) losses. Left-side bars represent the areal gain/loss of a LUC category and right-side bars show the intensity of gain/loss. The SG category had the highest and most intense loss to anthropogenized vegetation (aa) category between 2002 and 2008.

Conclusions and further research

In response to added and refined temporal, spatial and thematic dimensions increasing the volume of LUC data, the **OpenLand** package provides a comprehensive and integrated suite for the exploratory analysis of LUC changes. It offers seamless processing workflows beginning with time series consistency checking, data extraction, analysis and plotting of commonly used LUC metrics, as well as an implementation of Intensity Analysis, a state-of-the-art top-down hierarchical methodological framework to quantify the intensity of LUC changes. Regardless of the complexity of an LUC time series, all transitions and metrics are automatically extracted, quantified and stored as objects, without any need for further tabular data manipulation for analysis. Visualization tools create pre-formatted print-ready plots, which can be easily modified through function arguments.

Aldwaik and Pontius (2013) presented an extension of IA, which allows us to consider hypothetical classification errors in input LUC maps as part of the comparison between observed and uniform intensities in IA. The implementation of their method in **OpenLand** could further help users to assess the implications of errors on the strength of the evidence in the outputs of their Intensity Analysis and therefore, improve their understanding of LUC change processes.

Acknowledgements

Reginal Exavier is supported by the Brazilian Funding Agency CAPES (Coordination for the Improvement of Higher Level Personnel) through a Master studentship (2018 - 2020) at the Department of Geography of the Federal University of Mato Grosso. The research received financial support from the Critical Ecosystem Partnership Fund - CEPF (Grant Agreement CEPF-104254) and FAPEMAT (Grant Agreement 219228/2015).

Bibliography

F. O. Akinyemi, R. G. Pontius, and A. K. Braimoh. Land change dynamics: insights from Intensity Analysis applied to an African emerging city. *J. Spat. Sci.*, 62(1):69–83, 2017. ISSN 14498596. doi: 10.1080/14498596.2016.1196624. URL `http://dx.doi.org/10.1080/14498596.2016.1196624`. [p373]

S. Z. Aldwaik and R. G. Pontius. Intensity analysis to unify measurements of size and stationarity of land changes by interval, category, and transition. *Landsc. Urban Plan.*, 106(1):103–114, 2012. ISSN 01692046. doi: 10.1016/j.landurbplan.2012.02.010. URL http://dx.doi.org/10.1016/j.landurbplan.2012.02.010. [p373, 374, 380, 381, 383]

S. Z. Aldwaik and R. G. Pontius. Map errors that could account for deviations from a uniform intensity of land change. *Int. J. Geogr. Inf. Sci.*, 27(9):1717–1739, 2013. ISSN 13658816. doi: 10.1080/13658816.2013.787618. URL http://dx.doi.org/10.1080/13658816.2013.787618. [p374, 385]

J. J. Allaire, C. Gandrud, K. Russell, and C. J. Yetman. networkD3: D3 JavaScript Network Graphs from R, 2017. URL https://cran.r-project.org/package=networkD3. [p374]

B. Auguie. *gridExtra: Miscellaneous Functions for "Grid" Graphics*, 2017. URL https://cran.r-project.org/package=gridExtra. [p374]

A. P. Ballantyne, R. Andres, R. Houghton, B. D. Stocker, R. Wanninkhof, W. Anderegg, L. A. Cooper, M. DeGrandpre, P. P. Tans, J. B. Miller, C. Alden, and J. W. White. Audit of the global carbon budget: Estimate errors and their impact on uptake uncertainty. *Biogeosciences*, 12(8):2565–2584, 2015. ISSN 17264189. doi: 10.5194/bg-12-2565-2015. [p373]

V. Brovkin, L. Boysen, V. K. Arora, J. P. Boisier, P. Cadule, L. Chini, M. Claussen, P. Friedlingstein, V. Gayler, B. J. Van den hurk, G. C. Hurtt, C. D. Jones, E. Kato, N. De noblet ducoudre, F. Pacifico, J. Pongratz, and M. Weiss. Effect of anthropogenic land-use and land-cover changes on climate and land carbon storage in CMIP5 projections for the twenty-first century. *J. Clim.*, 26(18):6859–6881, 2013. ISSN 08948755. doi: 10.1175/JCLI-D-12-00623.1. [p373]

J. Chambers. *Software for Data Analysis*. Statistics and Computing. Springer New York, New York, NY, 2008. ISBN 978-0-387-75935-7. doi: 10.1007/978-0-387-75936-4. URL http://link.springer.com/10.1007/978-0-387-75936-4. [p381]

G. Chaudhuri and N. B. Mishra. Spatio-temporal dynamics of land cover and land surface temperature in Ganges-Brahmaputra delta: A comparative analysis between India and Bangladesh. *Appl. Geogr.*, 68:68–83, mar 2016. ISSN 01436228. doi: 10.1016/j.apgeog.2016.01.002. URL https://linkinghub.elsevier.com/retrieve/pii/S0143622816300029. [p373]

C. H. W. de Souza, W. R. Cervi, J. C. Brown, J. V. Rocha, and R. A. C. Lamparelli. Mapping and evaluating sugarcane expansion in Brazil's savanna using MODIS and intensity analysis: a case-study from the state of Tocantins. *J. Land Use Sci.*, 12(6):457–476, nov 2017. ISSN 1747-423X. doi: 10.1080/1747423X.2017.1404647. URL https://www.tandfonline.com/doi/full/10.1080/1747423X.2017.1404647. [p373]

Y. Dou, R. F. B. da Silva, H. Yang, and J. Liu. Spillover effect offsets the conservation effort in the Amazon. *J. Geogr. Sci.*, 28(11):1715–1732, nov 2018. ISSN 1009-637X. doi: 10.1007/s11442-018-1539-0. URL http://link.springer.com/10.1007/s11442-018-1539-0. [p383]

R. Exavier and P. Zeilhofer. *OpenLand: Quantitative Analysis and Visualization of LUCC*, 2020. URL https://CRAN.R-project.org/package=OpenLand. R package version 1.0.1. [p374]

Z. Gu, L. Gu, R. Eils, M. Schlesner, and B. Brors. circlize implements and enhances circular visualization in R. *Bioinformatics*, 30(19):2811–2812, 2014. [p374]

M. Herold, J. S. Latham, A. Di Gregorio, and C. C. Schmullius. Evolving standards in land cover characterization. *J. Land Use Sci.*, 1(2-4):157–168, dec 2006. ISSN 1747-423X. doi: 10.1080/17474230601079316. URL http://www.tandfonline.com/doi/abs/10.1080/17474230601079316. [p373]

R. J. Hijmans. *raster: Geographic Data Analysis and Modeling*, 2019. URL https://cran.r-project.org/package=raster. [p374]

B. Huang, J. Huang, R. Gilmore Pontius, and Z. Tu. Comparison of Intensity Analysis and the land use dynamic degrees to measure land changes outside versus inside the coastal zone of Longhai, China. *Ecol. Indic.*, 89:336–347, jun 2018. ISSN 1470160X. doi: 10.1016/j.ecolind.2017.12.057. URL https://linkinghub.elsevier.com/retrieve/pii/S1470160X1730849X. [p373]

G. C. Hurtt, L. P. Chini, S. Frolking, R. A. Betts, J. Feddema, G. Fischer, J. P. Fisk, K. Hibbard, R. A. Houghton, A. Janetos, C. D. Jones, G. Kindermann, T. Kinoshita, K. Klein Goldewijk, K. Riahi, E. Shevliakova, S. Smith, E. Stehfest, A. Thomson, P. Thornton, D. P. van Vuuren, and Y. P. Wang. Harmonization of land-use scenarios for the period 1500–2100: 600 years of global gridded annual land-use transitions, wood harvest, and resulting secondary lands. *Clim. Change*, 109(1-2):117–161, nov 2011. ISSN 0165-0009. doi: 10.1007/s10584-011-0153-2. URL http://link.springer.com/10.1007/s10584-011-0153-2. [p373]

Instituto SOS Pantanal and WWF-Brasil. Monitoramento das alterações da cobertura vegetal e uso do Solo na Bacia do Alto Paraguai – Porção Brasileira – Período de Análise: 2012 a 2014. Technical report, Brasilia, 2015. URL https://d3nehc6yl9qzo4.cloudfront.net/downloads/publicacao_bap_relatorio_2012_2014_web.pdf. [p375, 376, 377]

E. F. Lambin. Modelling and monitoring land-cover change processes in tropical regions. *Prog. Phys. Geogr. Earth Environ.*, 21(3):375–393, sep 1997. ISSN 0309-1333. doi: 10.1177/030913339702100303. URL http://journals.sagepub.com/doi/10.1177/030913339702100303. [p373]

E. F. Lambin and H. Geist, editors. *Land-Use and Land-Cover Change*. Global Change - The IGBP Series. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006. ISBN 978-3-540-32201-6. doi: 10.1007/3-540-32202-7. URL http://link.springer.com/10.1007/3-540-32202-7. [p373]

M. R. d. S. Melo, J. V. Rocha, V. D. Manabe, and R. A. C. Lamparelli. Intensity of land use changes in a sugarcane expansion region, Brazil. *J. Land Use Sci.*, 00(00):1–16, 2018. ISSN 17474248. doi: 10.1080/1747423X.2018.1499829. URL https://doi.org/10.1080/1747423X.2018.1499829. [p373]

A. Morin, J. Urban, P. D. Adams, I. Foster, A. Sali, D. Baker, and P. Sliz. Shining Light into Black Boxes. *Science (80-.).*, 336(6078):159–160, apr 2012. ISSN 0036-8075. doi: 10.1126/science.1218263. URL http://www.sciencemag.org/cgi/doi/10.1126/science.1218263. [p374]

S. Moulds, W. Buytaert, and A. Mijic. An open and extensible framework for spatially explicit land use change modelling: the lulcc R package. *Geosci. Model Dev.*, 8(10):3215–3229, oct 2015. ISSN 1991-9603. doi: 10.5194/gmd-8-3215-2015. URL https://www.geosci-model-dev.net/8/3215/2015/. [p374]

D. Müller and D. K. Munroe. Current and future challenges in land-use science. *J. Land Use Sci.*, 9(2):133–142, apr 2014. ISSN 1747-423X. doi: 10.1080/1747423X.2014.883731. URL http://www.tandfonline.com/doi/abs/10.1080/1747423X.2014.883731. [p373]

H. Mwangi, P. Lariu, S. Julich, S. Patil, M. McDonald, and K.-H. Feger. Characterizing the Intensity and Dynamics of Land-Use Change in the Mara River Basin, East Africa. *Forests*, 9(1):8, dec 2017. ISSN 1999-4907. doi: 10.3390/f9010008. URL http://www.mdpi.com/1999-4907/9/1/8. [p373]

G. Nelson, M. Rosegrant, A. Palazzo, and I. G. ... *Food security and climate change: Challenges to 2050 and beyond*. International Food Policy Research Institute, 2010. doi: 10.2499/9780896291874. URL https://cgspace.cgiar.org/handle/10568/33400http://ebrary.ifpri.org/cdm/ref/collection/p15738coll2/id/6778. [p373]

R. D. Peng. Reproducible Research in Computational Science. *Science (80-.).*, 334(6060):1226–1227, dec 2011. ISSN 0036-8075. doi: 10.1126/science.1213847. URL http://www.sciencemag.org/cgi/doi/10.1126/science.1213847. [p374]

R. G. Pontius Jr. and S. Khallaghi. intensity.analysis: Intensity of Change for Comparing Categorical Maps from Sequential Intervals, 2019. URL https://cran.r-project.org/package=intensity.analysis. [p374]

R. Prestele, P. Alexander, M. D. A. Rounsevell, A. Arneth, K. Calvin, J. Doelman, D. A. Eitelberg, K. Engström, S. Fujimori, T. Hasegawa, P. Havlik, F. Humpenöder, A. K. Jain, T. Krisztin, P. Kyle, P. Meiyappan, A. Popp, R. D. Sands, R. Schaldach, J. Schüngel, E. Stehfest, A. Tabeau, H. Van Meijl, J. Van Vliet, and P. H. Verburg. Hotspots of uncertainty in land-use and land-cover change projections: a global-scale model comparison. *Glob. Chang. Biol.*, 22(12):3967–3983, dec 2016. ISSN 13541013. doi: 10.1111/gcb.13337. URL http://doi.wiley.com/10.1111/gcb.13337. [p373]

Project MapBiomas. Collection 3.1 of Brazilian Land Cover & Use Map Series, 2019. URL https://mapbiomas.org/. [p377]

I. M. D. Rosa, S. E. Ahmed, and R. M. Ewers. The transparency, reliability and utility of tropical rainforest land-use and land-cover change models. *Glob. Chang. Biol.*, 20(6):1707–1722, jun 2014. ISSN 13541013. doi: 10.1111/gcb.12523. URL http://doi.wiley.com/10.1111/gcb.12523. [p374]

RStudio Team. RStudio: Integrated Development Environment for R, 2016. URL http://www.rstudio.com/. [p374]

X. P. Song, M. C. Hansen, S. V. Stehman, P. V. Potapov, A. Tyukavina, E. F. Vermote, and J. R. Townshend. Global land change from 1982 to 2016. *Nature*, 560(7720):639–643, 2018. ISSN 14764687. doi: 10.1038/s41586-018-0411-9. URL http://dx.doi.org/10.1038/s41586-018-0411-9. [p373]

S. Subasinghe, R. Estoque, and Y. Murayama. Spatiotemporal Analysis of Urban Growth Using GIS and Remote Sensing: A Case Study of the Colombo Metropolitan Area, Sri Lanka. *ISPRS Int. J. Geo-Information*, 5(11):197, oct 2016. ISSN 2220-9964. doi: 10.3390/ijgi5110197. URL http://www.mdpi.com/2220-9964/5/11/197. [p373]

O. G. Varga, R. G. Pontius, S. K. Singh, and S. Szabó. Intensity Analysis and the Figure of Merit's components for assessment of a Cellular Automata – Markov simulation model. *Ecol. Indic.*, 101(January):933–942, jun 2019. ISSN 1470160X. doi: 10.1016/j.ecolind.2019.01.057. URL https://linkinghub.elsevier.com/retrieve/pii/S1470160X19300743https://doi.org/10.1016/j.ecolind.2019.01.057. [p373]

P. H. Verburg, N. Crossman, E. C. Ellis, A. Heinimann, P. Hostert, O. Mertz, H. Nagendra, T. Sikor, K. H. Erb, N. Golubiewski, R. Grau, M. Grove, S. Konaté, P. Meyfroidt, D. C. Parker, R. R. Chowdhury, H. Shibata, A. Thomson, and L. Zhen. Land system science and sustainable development of the earth system: A global land project perspective. *Anthropocene*, 12(November):29–41, 2015. ISSN 22133054. doi: 10.1016/j.ancene.2015.09.004. URL http://dx.doi.org/10.1016/j.ancene.2015.09.004. [p373]

H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. ISBN 978-3-319-24277-4. URL http://ggplot2.org. [p374]

H. Wickham and L. Henry. *tidyr: Tidy Messy Data*, 2019. URL https://cran.r-project.org/package=tidyr. [p374]

H. Wickham, R. François, L. Henry, and K. Müller. *dplyr: A Grammar of Data Manipulation*, 2019. URL https://cran.r-project.org/package=dplyr. [p374]

H. Yang, S. Li, J. Chen, X. Zhang, and S. Xu. The Standardization and Harmonization of Land Cover Classification Systems towards Harmonized Datasets: A Review. *ISPRS Int. J. Geo-Information*, 6(5):154, may 2017. ISSN 2220-9964. doi: 10.3390/ijgi6050154. URL http://www.mdpi.com/2220-9964/6/5/154. [p373]

J. Yu, W. Li, and C. Zhang. A framework of experimental transiogram modelling for Markov chain geostatistical simulation of landscape categories. *Comput. Environ. Urban Syst.*, 73:16–26, jan 2019. ISSN 01989715. doi: 10.1016/j.compenvurbsys.2018.07.007. URL https://linkinghub.elsevier.com/retrieve/pii/S0198971517305653. [p373]

Reginal Exavier
Department of Geography
Federal University of Mato Grosso
Avenida Fernando Corrêa da Costa, 2367 – Boa Esperança, Cuiabá – MT, 78060-900
ORCiD: 0000-0002-5237-523X
reginalexavier@rocketmail.com

Peter Zeilhofer
Department of Geography
Federal University of Mato Grosso
Avenida Fernando Corrêa da Costa, 2367 – Boa Esperança, Cuiabá – MT, 78060-900
zeilhoferpeter@gmail.com

FarmTest: An R Package for Factor-Adjusted Robust Multiple Testing

by Koushiki Bose, Jianqing Fan, Yuan Ke, Xiaoou Pan and Wen-Xin Zhou

Abstract We provide a publicly available library **FarmTest** in the R programming system. This library implements a factor-adjusted robust multiple testing principle proposed by Fan et al. (2019) for large-scale simultaneous inference on mean effects. We use a multi-factor model to explicitly capture the dependence among a large pool of variables. Three types of factors are considered: observable, latent, and a mixture of observable and latent factors. The non-factor case, which corresponds to standard multiple mean testing under weak dependence, is also included. The library implements a series of adaptive Huber methods integrated with fast data-driven tuning schemes to estimate model parameters and to construct test statistics that are robust against heavy-tailed and asymmetric error distributions. Extensions to two-sample multiple mean testing problems are also discussed. The results of some simulation experiments and a real data analysis are reported.

Introduction

In the era of big data, large-scale multiple testing problems arise from a wide range of fields, including biological sciences such as genomics and neuroimaging, social science, signal processing, marketing analytics, and financial economics. When testing multitudinous statistical hypotheses simultaneously, researchers appreciate statistically significant evidence against the null hypothesis with a guarantee of controlled false discovery rate (FDR) (Benjamini and Hochberg, 1995). Since the seminal work of Benjamini and Hochberg (1995), multiple testing with FDR control has been extensively studied and successfully used in many applications. Most of the existing testing procedures are tailored to independent or weakly dependent hypotheses or tests. See, Storey (2002), Genovese and Wasserman (2004) and Lehmann and Romano (2005), to name a few. The independence assumption, however, is restricted in real applications as correlation effects are ubiquitous in high dimensional measurements. Ignoring such strong dependency and directly applying standard FDR controlling procedures can lead to inaccurate false discovery control, loss of statistical power, and unreliable scientific conclusions.

Over the past decade, a multi-factor model has proven to be an effective tool for modeling cross-sectional dependence, with applications in genomics, neuroscience, and financial economics. Related references in the context of multiple testing include Leek and Storey (2008), Friguet et al. (2009), Fan et al. (2012), Desai and Storey (2011) and Fan and Han (2017). A common thread of the aforementioned works is that the construction of test statistics and p-values heavily relies on the assumed joint normality of factors and noise, which is arguably another folklore regarding high dimensional data. Therefore, it is imperative to develop large-scale multiple testing tools that adjust cross-sectional dependence properly and are robust to heavy-tailedness at the same time.

Recently, Fan et al. (2019) developed a Factor-Adjusted Robust Multiple Test (FarmTest) procedure for large-scale simultaneous inference with highly correlated and heavy-tailed data. Their emphasis is on achieving robustness against both strong cross-sectional dependence and heavy-tailed sampling distribution. Specifically, let $\boldsymbol{X} = (X_1, \dots, X_p)^\intercal$ be a random vector with mean $\boldsymbol{\mu} = (\mu_1, \dots, \mu_p)^\intercal$. We are interested in testing the p hypotheses $H_{0j} : \mu_j = 0$, and wish to find a multiple comparison procedure to test individual hypotheses while controlling the FDR. The FarmTest method models the dependency among X_j's through an approximate multi-factor model, namely $X_j = \mu_j + \boldsymbol{b}_j^\intercal \boldsymbol{f} + u_j$, where $\boldsymbol{f}$ is a zero-mean random vector capturing the dependence structure of $\boldsymbol{X}$. The method applies to either observable or unobservable factor $\boldsymbol{f}$. The former includes the non-factor case which corresponds to the standard multiple mean testing problem. For the latter, we estimate the factors in a data-driven way. Test statistics are then calculated by subtracting out the realized common factors. Multiple comparisons are then applied to these weakly dependent factor-adjusted test statistics. Also, adjusting the factors before testing reduces signal-to-noise ratios, which enhances statistical power. Since a data-driven eigenvalue ratio method is used to estimate the number of (latent) factors, the testing procedure still works when the dependence is weak and therefore is rather flexible.

This article describes an R library named **FarmTest**, which implements the FarmTest procedure(s) developed in Fan et al. (2019). It is a user-friendly tool to conduct large-scale hypothesis testing, especially when one or several of the following scenarios are present: the dimensionality is far larger than the sample size available; the data is heavy-tailed and/or asymmetric; there is strong cross-sectional dependence among the data. **FarmTest** is implemented using the Armadillo library (Sanderson and Curtin, 2016) with **Rcpp** interfaces (Eddelbuettel and Francois, 2011; Eddelbuettel and Sanderson, 2014). A simple call of **FarmTest** package only requires the input of a data matrix and the null hypotheses to be tested. It outputs the hypotheses that are rejected, along with the p-values and some estimated parameters which may be of use in further analysis. Testing can be carried out for both one-sample and two-sample problems.

Another key feature of our package is that it implements several recently developed robust methods for fitting regression models (Zhou et al., 2018; Sun et al., 2020) and covariance estimation (Ke et al., 2019). When data is generated from a heavy-tailed distribution, test statistics that are based on the least-squares method are sensitive to outliers, which often causes significant false discoveries and suboptimal power (Zhou et al., 2018). The effect of heavy-tailedness is amplified by high dimensionality; even moderate-tailed distributions can generate very large outliers by chance, making it difficult to separate the true signals from spurious variables. As a result, large-scale multiple testing based on non-robust statistics may engender an excessive false discovery rate, which arguably is one of the causes of the current crisis in reproducibility in science. Moreover, to choose the multiple tuning parameters in robust regression and covariance estimation, we employ the recently developed data-driven procedures (Wang et al., 2020; Ke et al., 2019), which are particularly designed for adaptive Huber regression and are considerably faster than the cross-validation method used in Fan et al. (2019).

We further remark that most existing multiple testing R packages do not address the robustness against both heavy-tailed distribution and strong dependence. The hypothesis testing function in R, named `t.test`, neither adjusts for strong dependence in the data nor estimates the parameters in focus robustly. The built-in function `p.adjust` or the package **qvalue** (Storey, 2002) only adjust user-input p-values for multiple testing and do not address the problem of estimating the p-values themselves. The package **multcomp** (Hothorn et al., 2008) provides simultaneous testing tools for general linear hypotheses in parametric models under the assumptions that the central limit theorem holds. The package **multtest** (Pollard et al., 2005) is developed to implement non-parametric bootstrap and permutation resampling-based multiple testing procedures. The **multtest** can calculate test statistics based on ranked data which is robust against outliers but yields biased mean estimators. In addition, **multtest** cannot explicitly model the dependence structure in data. The package **mutoss** is designed to apply many existing multiple hypothesis testing procedures with FDR control and p-value correction. Nevertheless, none of the tools in **mutoss** is suitable to deal with both strong dependency and heavy-tailedness. Moreover, existing packages are often difficult to navigate since users need to combine many functions to perform multiple tests.

Factor-adjusted robust multiple testing

In this section, we revisit the problem of simultaneous inference on the mean effects under a factor model and discuss the main ideas behind the FarmTest method developed by Fan et al. (2019).

Multiple testing with false discovery rate control

Suppose we observe n independent data vectors $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_n$ from a p-dimensional random vector $\boldsymbol{X} = (X_1, \ldots, X_p)^\intercal$. Further, let $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_p)^\intercal$ and $\boldsymbol{\Sigma} = \left(\sigma_{jk}\right)_{1\le j,k\le p}$ denote the mean vector and covariance matrix of $\boldsymbol{X}$, respectively. In the language of hypothesis testing, we are interested in one of the following three types of hypotheses:

$$H_{0j} : \mu_j = h_j^0 \quad \text{versus} \quad H_{1j} : \mu_j \neq h_j^0; \tag{1}$$

$$H_{0j} : \mu_j \le h_j^0 \quad \text{versus} \quad H_{1j} : \mu_j > h_j^0; \tag{2}$$

$$H_{0j} : \mu_j \ge h_j^0 \quad \text{versus} \quad H_{1j} : \mu_j < h_j^0; \tag{3}$$

for $j = 1, \ldots, p$. In the default setting, $h_j^0 = 0$ for all j.

Here we take the two-sided test (1) as an example to discuss the false discovery rate (FDR) control. For $1 \le j \le p$, let T_j be a generic test statistic for the jth hypothesis. Given a prespecified threshold $z > 0$, we reject the jth null hypothesis if $|T_j| \ge z$. The FDR is defined as the expected value of the false discovery proportion (FDP): $\text{FDR}(z) = \mathbb{E}\{\text{FDP}(z)\}$ with $\text{FDP}(z) = V(z) / \max\{R(z), 1\}$, where $R(z) = \sum_{j=1}^{p} 1\left(|T_j| \ge z\right)$ is the number of total rejections and $V(z) = \sum_{j:\mu_j = h_j^0} 1\left(|T_j| \ge z\right)$ is the number of false discoveries. If the FDP (z) were known, the rejection threshold will be $z_\alpha = \inf\{z \ge 0 : \text{FDP}(z) \le \alpha\}$ in order to achieve FDP control. Notice that $R(z)$ is observable given the data while $V(z)$ is an unobserved random quantity that needs to be estimated.

Assume that there are $p_0 = \pi_0 p$ true nulls and $p_1 = (1 - \pi_0)\, p$ true alternatives. Suppose the constructed test statistic T_j is close in distribution to standard normal for every $j = 1, \ldots, p$, if the test statistics are weakly dependent. Heuristically the number of false discoveries $V(z)$ is close to

$2p_0\,\Phi(-z)$ for any $z \geq 0$. A conservative way is to replace $V(z)$ by $2p\,\Phi(-z)$. Assuming the normal approximation is sufficiently accurate, $2p\,\Phi(-z)$ provides an overestimate of the number of false discoveries, resulting in an underestimate of the FDP (z). A more accurate method is to estimate the unknown proportion of null hypotheses $\pi_0 = p_0/p$ from the data. Let $\left\{P_j = 2\Phi\left(-|T_j|\right)\right\}_{j=1}^{p}$ be the approximate p-values. For a predetermined $\lambda \in [0,1)$, Storey (2002) suggest to estimate π_0 by $\widehat{\pi}_0(\lambda) = \{(1-\lambda)\,p\}^{-1}\sum_{j=1}^{p} 1\left(P_j > \lambda\right)$, because larger p-values are more likely to come from the true null hypotheses. Consequently, a data-driven rejection threshold is $\widehat{z}_\alpha = \inf\left\{z \geq 0 : \widehat{\mathrm{FDP}}(z) \leq \alpha\right\}$, where $\widehat{\mathrm{FDP}}(z) = 2\widehat{\pi}_0(\lambda)\,p\,\Phi(-z)/R(z)$.

Factor-adjusted test statistics

In this section, we discuss the construction of test statistics under strong cross-sectional dependency captured by common factors. Specifically, we allow the p coordinates of $\boldsymbol{X}$ to be strongly correlated through an approximate factor model of the form $\boldsymbol{X} = \boldsymbol{\mu} + \mathbf{B}\boldsymbol{f} + \boldsymbol{u}$, where $\mathbf{B} = (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_p)^\intercal \in \mathbb{R}^{p\times K}$ represents the factor loading matrix, $\boldsymbol{f} = (f_1, \ldots, f_K)^\intercal \in \mathbb{R}^K$ is the common factor, and $\boldsymbol{u} = (u_1, \ldots, u_p)^\intercal \in \mathbb{R}^p$ denotes a vector of idiosyncratic errors uncorrelated with $\boldsymbol{f}$. The observed samples thus follow

$$\boldsymbol{X}_i = \boldsymbol{\mu} + \mathbf{B}\boldsymbol{f}_i + \boldsymbol{u}_i, \quad i = 1, \ldots, n, \tag{4}$$

where $(\boldsymbol{f}_i, \boldsymbol{u}_i)$'s are independent copies of $(\boldsymbol{f}, \boldsymbol{u})$. Assume that both $\boldsymbol{f}$ and $\boldsymbol{u}$ have zero means. Further, denote by $\boldsymbol{\Sigma}_f$ and $\boldsymbol{\Sigma}_{\boldsymbol{u}} = \left(\sigma_{u,jk}\right)_{1\leq j,k\leq p}$ the covariance matrices of $\boldsymbol{f}$ and $\boldsymbol{u}$, respectively.

Our package allows the common factor $\boldsymbol{f}$ to be either observable or unobservable. In the former case, we observe $\{(\boldsymbol{X}_i, \boldsymbol{f}_i)\}_{i=1}^n$ so that model (4) is reduced to a multi-response linear regression problem; for the latter, we only observe $\{\boldsymbol{X}_i\}_{i=1}^n$ and therefore need to recover the latent factors. The latent factor model has identifiability issues; see Bai and Li (2012) for a set of possible solutions. For simplicity, we assume that $\boldsymbol{\Sigma}_f = \mathbf{I}_K$ and $\mathbf{B}^\intercal\mathbf{B}$ is diagonal.

Robust estimation

As another key feature, the FarmTest method is robust against heavy-tailed sampling distributions. Under such scenarios, the ordinary least squares estimators can be suboptimal. Recently, Fan et al. (2017) and Sun et al. (2020) proposed the adaptive Huber regression method, the core of which is Huber's M-estimator (Huber, 1964) with a properly calibrated robustification parameter that adapts to the sample size, dimensionality and noise level. They showed that the adaptive Huber estimator admits a sub-Gaussian-type deviation bound under mild moment conditions. This package exploits this approach to estimate the unknown parameters and to construct test statistics.

Algorithms

In this section, we formally describe the algorithms for the FarmTest procedure. We revisit and discuss procedures for the two scenarios with observable and unobservable/latent factors (Zhou et al., 2018; Fan et al., 2019). Notice that the two scenarios are inherently different in terms of estimating unknown parameters and constructing test statistics. Moreover, the selection of tuning parameters is based on the recent methods proposed by Ke et al. (2019) and Wang et al. (2020).

Observable factors

Suppose we observe independent data vectors $\{(\boldsymbol{X}_i, \boldsymbol{f}_i)\}_{i=1}^n$ from model (4). The testing procedure for the hypotheses in (1)–(3) is described in Algorithm 1. Algorithm 1 automatically selects the robustification parameters $\left\{\tau_j, \upsilon_j\right\}_{j=1}^p$ following the data-driven method proposed by Ke et al. (2019). See the Section of Selection of tuning parameters for more details. To enhance the finite sample performance, alternatively we can use the weighted/multiplier bootstrap (Zhou et al., 2018; Chen and Zhou, 2019) to compute p-values for all the marginal hypotheses. For $b = 1, \ldots, B$, we obtain the corresponding bootstrap draw of $\left(\widehat{\mu}_j, \widehat{\boldsymbol{b}}_j\right)$ via $\left(\widehat{\mu}^\flat_{b,j}, \widehat{\boldsymbol{b}}^\flat_{b,j}\right) = \mathrm{argmin}_{\mu,\boldsymbol{b}}\sum_{i=1}^n w_{b,ij}\ell_{\tau_j}\left(X_{ij} - \mu - \boldsymbol{f}_i^\intercal\boldsymbol{b}\right)$, where $\left\{w_{b,ij}, i = 1, \ldots, n, j = 1, \ldots p\right\}$ are independent and identically distributed (iid) random variables that are independent from the data and satisfy $\mathbb{E}\left(w_{b,ij}\right) = 1$ and $\mathrm{var}\left(w_{b,ij}\right) = 1$. To retain convexity

Algorithm 1 FarmTest with known factors (Zhou et al., 2018)

```
Input: Data $\{(\boldsymbol{X}_i, \boldsymbol{f}_i)\}_{i=1}^n$, null hypotheses $\left\{h_j^0\right\}_{j=1}^p$, and $\alpha, \lambda \in (0,1)$
1: For $j = 1, \ldots, p$, obtain the Huber estimators
   $\left(\widehat{\mu}_j, \widehat{\boldsymbol{b}}_j\right) \in \operatorname{argmin}_{\mu, \boldsymbol{b}} \sum_{i=1}^n \ell_{\tau_j}\left(X_{ij} - \mu - \boldsymbol{f}_i^\intercal \boldsymbol{b}\right)$.
2: Estimation of residual variances $\sigma_{u,jj}$'s: compute
     (i) $\widehat{\boldsymbol{\Sigma}}_f = (1/n) \sum_{i=1}^n \boldsymbol{f}_i \boldsymbol{f}_i^\intercal$, $\widehat{\theta}_j \in \operatorname{argmin}_\theta \sum_{i=1}^n \ell_{v_j}\left(X_{ij}^2 - \theta\right)$ for $j = 1, \ldots, p$;
     (ii) $\widehat{\sigma}_{u,jj} = \widehat{\theta}_j - \widehat{\mu}_j^2 - \widehat{\boldsymbol{b}}_j^\intercal \widehat{\boldsymbol{\Sigma}}_f \widehat{\boldsymbol{b}}_j$ if $\widehat{\theta}_j > \widehat{\mu}_j^2 + \widehat{\boldsymbol{b}}_j^\intercal \widehat{\boldsymbol{\Sigma}}_f \widehat{\boldsymbol{b}}_j$; otherwise $\widehat{\sigma}_{u,jj} = \widehat{\theta}_j$.
3: Construct test statistics $T_j = \sqrt{n/\widehat{\sigma}_{u,jj}}\left(\widehat{\mu}_j - h_j^0\right)$ for $j = 1, \ldots, p$.
4: Compute p-values $\{P_j\}_{j=1}^p = \begin{cases} \left\{2\Phi\left(-\left|T_j\right|\right)\right\}_{j=1}^p & \text{for (1)}, \\ \left\{\Phi\left(-T_j\right)\right\}_{j=1}^p & \text{for (2)}, \\ \left\{\Phi\left(T_j\right)\right\}_{j=1}^p & \text{for (3)}. \end{cases}$
5: Estimate the proportion of true alternatives: $\widehat{\pi}_0(\lambda) = \frac{\operatorname{Card}\{P_j > \lambda\}}{(1-\lambda)p}$.
6: Order the p-values as $P_{(1)} \leq \cdots \leq P_{(p)}$.
   Compute the rejection threshold $t := \max\left\{1 \leq j \leq p : P_{(j)} \leq \frac{\alpha j}{\widehat{\pi}_0(\lambda) p}\right\}$
7: Reject each hypothesis in the set $\left\{1 \leq j \leq p : P_j \leq P_{(t)}\right\}$.
Output: Rejected hypotheses, p-values, other estimated parameters
```

of the loss function, nonnegative random weights are preferred, such as $w_{b,ij} \sim \operatorname{Exp}(1)$—exponential distribution with rate 1, or $w_{b,ij} \sim 2\operatorname{Ber}(1/2)$—$\mathbb{P}\left(w_{b,ij} = 0\right) = \mathbb{P}\left(w_{b,ij} = 2\right) = 1/2$. For two-sided alternatives, the bootstrap p-values are then defined as $P_j^\flat = (1/B) \sum_{b=1}^B I\left(|\widehat{\mu}_{b,j}^\flat - \widehat{\mu}_j| \geq |\widehat{\mu}_j|\right)$, followed by Steps 5–7 in Algorithm 1.

An extension of Algorithm 1 to the two-sample problem is also implemented in the package. Suppose we observe two independent samples $\{(\boldsymbol{X}_i, \boldsymbol{f}_i^X)\}_{i=1}^{n_1}$ and $\{(\boldsymbol{Y}_i, \boldsymbol{f}_i^Y)\}_{i=1}^{n_2}$ from the models

$$\boldsymbol{X} = \boldsymbol{\mu}^X + \mathbf{B}^X \boldsymbol{f}^X + \boldsymbol{u}^X \text{ and } \boldsymbol{Y} = \boldsymbol{\mu}^Y + \mathbf{B}^Y \boldsymbol{f}^Y + \boldsymbol{u}^Y. \tag{5}$$

We are interested in the p hypotheses $H_{0j} : \mu_j^X - \mu_j^Y = h_j^0$ versus $H_{1j} : \mu_j^X - \mu_j^Y \neq h_j^0$ or versus some one-sided alternatives. To begin with, applying Step 1 in Algorithm 1 separately to each dataset to obtain the estimates $\left\{\left(\widehat{\mu}_j^X, \widehat{\mu}_j^Y\right)\right\}_{j=1}^p$ and $\left\{\left(\widehat{\sigma}_{u,jj}^X, \widehat{\sigma}_{u,jj}^Y\right)\right\}_{j=1}^p$. Next, define the two-sample counterparts of the test statistics in Step 2 as $T_j = \left(\widehat{\mu}_j^X - \widehat{\mu}_j^Y - h_j^0\right) / \sqrt{\widehat{\sigma}_{u,jj}^X / n_1 + \widehat{\sigma}_{u,jj}^Y / n_2}$ for $j = 1, \ldots, p$. After that, we follow Steps 3–7 as in Algorithm 1 to obtain the p-values and rejected hypotheses.

Latent factors

In this section, suppose we are given independent observations $\{\boldsymbol{X}_i\}_{i=1}^n$. The strong dependency among the coordinates of $\boldsymbol{X}_i$ is captured by a latent factor $\boldsymbol{f}_i$ (Leek and Storey, 2008). Due to the need of recovering latent factors from the data, the corresponding testing procedure is more involved. We summarize the major steps in Algorithm 2. All the tuning parameters required for Algorithm 2, $\left\{\tau_j, v_j\right\}_{j=1}^p$ and $\left\{v_{jk}\right\}_{1 \leq j < k \leq p}$, are automatically selected from the data; see Selection of tuning parameters.

An extension of Algorithm 2 to the two-sample problem is also included in the library. Suppose we observe two independent samples $\{\boldsymbol{X}_i\}_{i=1}^{n_1}$ and $\{\boldsymbol{Y}_i\}_{i=1}^{n_2}$, and wish to test the hypotheses H_{0j} : $\mu_j^X - \mu_j^Y = h_j^0$ versus $H_{1j} : \mu_j^X - \mu_j^Y \neq h_j^0$ or some one-sided alternatives. In this case, Steps 1–5 in Algorithm 2 are applied separately to each dataset to obtain the estimates $\left\{\left(\widehat{\mu}_j^X, \widehat{\mu}_j^Y\right)\right\}_{j=1}^p$,

Algorithm 2 FarmTest with latent factors (Fan et al., 2019)

```
Input: Data $\{X_i\}_{i=1}^n$, null hypotheses $\left\{h_j^0\right\}_{j=1}^p$, and $\alpha, \lambda \in (0,1)$
 1: For $j = 1, \dots, p$, compute
      • $\widehat{\mu}_j = \mathrm{argmin}_\mu \sum_{i=1}^n \ell_{\tau_j}(X_{ij} - \mu)$, $\widehat{\theta}_j = \mathrm{argmin}_\theta \sum_{i=1}^n \ell_{v_j}\left(X_{ij}^2 - \theta\right)$,
      • $\widehat{\sigma}_{jj} = \begin{cases} \widehat{\theta}_j - \widehat{\mu}_j^2 & \text{if } \widehat{\theta}_j > \widehat{\mu}_j^2, \\ \widehat{\theta}_j & \text{otherwise.} \end{cases}$
 2: Define the paired data $\{Y_1, Y_2, \dots, Y_N\} = \{X_1 - X_2, X_1 - X_3, \dots, X_{n-1} - X_n\}$, where
    $N = n(n-1)/2$. For $1 \le j < k \le p$, compute
      • $\widehat{\sigma}_{jk} = \mathrm{argmin}_\theta \sum_{i=1}^N \ell_{v_{jk}}(Y_{ij}Y_{ik}/2 - \theta)$, and $\widehat{\sigma}_{kj} = \widehat{\sigma}_{jk}$.
 3: Define the covariance matrix estimator $\widehat{\mathbf{\Sigma}} = \left(\widehat{\sigma}_{jk}\right)_{1 \le j,k \le p}$.
      • Let $\lambda_1 \ge \lambda_2 \ge \cdots \ge \lambda_p$ be the ordered eigenvalues of $\widehat{\mathbf{\Sigma}}$ and denote by
        $v_1, v_2, \dots, v_p$ the corresponding eigenvectors.
      • Calculate $K = \mathrm{argmax}_{1 \le k \le \min(n,p)/2} \frac{\lambda_k}{\lambda_{k+1}}$. This step is omitted if $K$ is user-specified.
      • Calculate $\widehat{\mathbf{B}} = \left(\widehat{\boldsymbol{b}}_1, \dots, \widehat{\boldsymbol{b}}_p\right)^\intercal = \left(\lambda_1^{1/2} \boldsymbol{v}_1, \dots, \lambda_K^{1/2} \boldsymbol{v}_K\right) \in \mathbb{R}^{p \times K}$.
 4: $\bar{\boldsymbol{f}} = \mathrm{argmin}_{\boldsymbol{f} \in \mathbb{R}^K} \sum_{j=1}^p \ell_\gamma\left(\bar{X}_j - \widehat{\boldsymbol{b}}_j^\intercal \boldsymbol{f}\right)$, where $\bar{X}_j = (1/n)\sum_{i=1}^n X_{ij}$.
 5: For $j = 1, \dots, p$, compute $\widehat{\sigma}_{u,jj} = \begin{cases} \widehat{\sigma}_{jj} - \left\|\widehat{\boldsymbol{b}}_j\right\|_2^2 & \text{if } \widehat{\sigma}_{jj} > \left\|\widehat{\boldsymbol{b}}_j\right\|_2^2, \\ \widehat{\sigma}_{jj} & \text{otherwise.} \end{cases}$
 6: Construct test statistics $T_j = \sqrt{n/\widehat{\sigma}_{u,jj}}\left(\widehat{\mu}_j - \widehat{\boldsymbol{b}}_j^\intercal \bar{\boldsymbol{f}} - h_j^0\right)$, $j = 1, \dots, p$.
 7: Compute p-values $P_j = \begin{cases} \{2\Phi(-|T_j|)\}_{j=1}^p & \text{for (1)}, \\ \{\Phi(-T_j)\}_{j=1}^p & \text{for (2)}, \\ \{\Phi(T_j)\}_{j=1}^p & \text{for (3)}. \end{cases}$
 8: Estimate the proportion of true alternatives: $\widehat{\pi}_0(\lambda) = \frac{\mathrm{Card}\{P_j > \lambda\}}{(1-\lambda)p}$.
 9: Compute the rejection threshold $t := \max\left\{1 \le j \le p : P_{(j)} \le \frac{\alpha j}{\widehat{\pi}_0(\lambda)p}\right\}$
10: Reject each hypothesis in the set $\left\{1 \le j \le p : P_j \le P_{(t)}\right\}$.
Output: Rejected hypotheses, p-values, other estimated parameters
```

$\left\{\left(\widehat{\sigma}_{u,jj}^X, \widehat{\sigma}_{u,jj}^Y\right)\right\}_{j=1}^p$, $\widehat{\mathbf{B}}^X$, $\widehat{\mathbf{B}}^Y$, $\bar{\boldsymbol{f}}^X$ and $\bar{\boldsymbol{f}}^Y$. After replacing the test statistics in Step 6 with

$$T_j = \frac{\left(\widehat{\mu}_j^X - \left\langle \widehat{\boldsymbol{b}}_j^X, \bar{\boldsymbol{f}}^X \right\rangle\right) - \left(\widehat{\mu}_j^Y - \left\langle \widehat{\boldsymbol{b}}_j^Y, \bar{\boldsymbol{f}}^Y \right\rangle\right) - h_j^0}{\sqrt{\widehat{\sigma}_{u,jj}^X/n_1 + \widehat{\sigma}_{u,jj}^Y/n_2}}, \quad j = 1, \dots, p,$$

one can follow Steps 7–10 to obtain the p-values and rejected hypotheses.

Partially observable factors

Motivated by applications to comparative microarray experiments (Leek and Storey, 2008; Friguet et al., 2009) and mutual fund selection (Lan and Du, 2019), we further discuss the case where both explanatory variables and latent factors are present. The statistical model is of the form

$$\boldsymbol{X}_i = \boldsymbol{\mu} + \mathbf{B}\boldsymbol{f}_i + \mathbf{C}\boldsymbol{g}_i + \boldsymbol{u}_i, \; i = 1, \dots, n,$$

where $\boldsymbol{f}_i \in \mathbb{R}^K$ is a vector of explanatory variables and $\boldsymbol{g}_i \in \mathbb{R}^L$ represents the latent factor. Here $L \geq 1$ may be user-specified or unknown. For multiple comparison of the mean effects under this model, the **FarmTest** package can be used in a two-stage fashion. In the first stage, apply Algorithm 1 to fit model (4) with observed data $\{(\boldsymbol{X}_i, \boldsymbol{f}_i)\}_{i=1}^n$ and compute fitted residuals $\boldsymbol{X}_i^{\text{res}} = \boldsymbol{X}_i - \widehat{\mathbf{B}}\boldsymbol{f}_i$; in the second stage, run Algorithm 2 on $\{\boldsymbol{X}_i^{\text{res}}\}_{i=1}^n$ to conduct factor-adjusted multiple testing.

Selection of tuning parameters

The FarmTest procedure involves multiple tuning parameters, including the number of factors K (if not specified by the user) and robustification parameters for fitting factor models. For the former, we apply the eigenvalue ratio method (Lam and Yao, 2012; Ahn and Horenstein, 2013) to estimate K, that is, $\widehat{K} = \text{argmax}_{1 \leq k \leq K_{\max}} \lambda_k\left(\widehat{\boldsymbol{\Sigma}}\right) / \lambda_{k+1}\left(\widehat{\boldsymbol{\Sigma}}\right)$, where $\widehat{\boldsymbol{\Sigma}}$ is a generic covariance matrix estimator with eigenvalues $\lambda_1\left(\widehat{\boldsymbol{\Sigma}}\right) \geq \cdots \geq \lambda_p\left(\widehat{\boldsymbol{\Sigma}}\right)$, and $K_{\max}$ be a prescribed upper bound. In the library, we take $K_{\max} = \min(n, p)/2$. This method is chosen as it does not involve other hyperparameters (except $K_{\max}$). When the factors are unobservable, the estimation of K is essentially an un-supervised learning problem. We choose K to be the smallest nonnegative integer such that the residuals $\boldsymbol{X}_i - \mathbf{B}\boldsymbol{f}_i$ are weakly correlated. Therefore, slight overestimation of K does not affect much of the testing results. If K is set to be zero, the **FarmTest** library directly applies a robust multiple testing procedure based on Huber's M-estimation partnered with multiplier bootstrap. See Zhou et al. (2018) for more details.

The robustification parameter in the Huber loss plays an important role in controlling the bias-robustness tradeoff. According to the theoretical analysis in Zhou et al. (2018), the optimal choice of τ_j in Algorithm 1 depends on the variance of X_j. Due to heterogeneity, we have p different τ_j's that need to be selected from the data. Furthermore, the covariance estimation step in Algorithm 2 entails as many as $p(p-1)/2$ parameters v_{jk}. Cross-validation is therefore computationally expensive when the dimension is large. Recently, Ke et al. (2019) and Wang et al. (2020) proposed fast data-driven methods, which estimate the regression coefficients/covariances and calibrate the tuning parameter simultaneously by solving a system of equations. Numerical studies therein suggest that this data-driven method is considerably faster than cross-validation while performs equally as well.

Package overview

The **FarmTest** package is publicly available from the Comprehensive R Archive Network (CRAN) and its GitHub page `https://github.com/XiaoouPan/FarmTest`. It contains four core functions. The main function `farm.test` carries out the entire FarmTest procedure, and outputs the testing results along with several useful estimated model parameters. User-friendly summary, print, and plot functions that summarize and visualize the test outcome are equipped with `farm.test`. The other three functions, `huber.mean`, `huber.cov` and `huber.reg` implement data-driven robust methods for estimating the mean vector and covariance matrix (Ke et al., 2019) as well as the regression coefficients (Wang et al., 2020). In particular, the `huber.reg` function uses the gradient descent algorithm with Barzilai and Borwein step size (Barzilai and Borwein, 1988). In this section, we focus primarily on introducing the `farm.test` function, and demonstrate its usage with numerical experiments.

A showcase example

We first present an example by applying the package to a synthetic dataset. To begin with, we use the **rstiefel** package (Hoff, 2012) to simulate a uniformly distributed random orthonormal matrix as the loading matrix $\mathbf{B}$ after rescaling. With sample size $n = 120$, dimension $p = 400$ and number of factors $K = 5$, we generate data vectors $\{\boldsymbol{X}_i\}_{i=1}^n$ from model (4), where the factors $\boldsymbol{f}_i \in \mathbb{R}^K$ follow a standard multivariate normal distribution and the noise vectors $\boldsymbol{u}_i \in \mathbb{R}^p$ are drawn from a multivariate t_3 distribution with zero mean and identity covariance matrix. For the mean vector $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_p)^\intercal$, we set the first $p_1 = 100$ coordinates to be 1 and the rest to be 0.

```
library(FarmTest)
library(rstiefel)
library(mvtnorm)
n <- 120
p <- 400
K <- 5
set.seed(100)
B <- rustiefel(p, K) %*% diag(rep(sqrt(p), K))
```

```
FX <- rmvnorm(n, rep(0, K), diag(K))
p1 <- 100
strength <- 1
mu <- c(rep(strength, p1), rep(0, p - p1))
U <- rmvt(n, diag(p), 3)
X <- rep(1, n) %*% t(mu) + FX %*% t(B) + U
```

Function call with default parameters

Using the data generated above, let us call the main function `farm.test` with all default optional parameters, and then print the outputs.

```
output <- farm.test(X)
output

One-sample FarmTest with unknown factors
n = 120, p = 400, nFactors = 5
FDR to be controlled at: 0.05
Alternative hypothesis: two.sided
Number of hypotheses rejected: 104
```

As shown in the snapshot above, the function `farm.test` correctly estimates the number of factors, and rejects 104 hypotheses with 4 false discoveries. For this individual experiment, the FDP and power are 0.038 and 1, respectively as calculated below. Here the power is referred to as the ratio between the number of correct rejections and the number of nonnulls p_1.

```
FDP <- sum(output$reject > p1) / length(output$reject)
FDP

[1] 0.03846154

power <- sum(output$reject <= p1) / p1
power

[1]  1
```

All the outputs are incorporated into a list, which can be quickly examined by `names()` function. See Table 1 for detailed descriptions of the outputs.

```
names(output)

 [1] "means"       "stdDev"      "loadings"    "eigenVal"    "eigenRatio"  "nFactors"
 [7] "tStat"       "pValues"     "pAdjust"     "significant" "reject"      "type"
[13] "n"           "p"           "h0"          "alpha"       "alternative"
```

We can present the testing results using the affiliated `summary` function.

```
head(summary(output))

      means     p-values   p-adjusted significance
1 1.0947056 1.768781e-18 8.936997e-17            1
2 0.8403608 3.131733e-09 1.157817e-08            1
3 0.8668348 1.292850e-11 6.532295e-11            1
4 0.9273998 2.182485e-12 1.350281e-11            1
5 0.7257105 7.699350e-08 2.593465e-07            1
6 0.9473088 1.180288e-13 1.192712e-12            1
```

To visualize the testing results, in Figure 1 we present several plots based on the outputs. From the histograms of estimated means and test statistics, we see that data are generally categorized into two groups, one of which has $\widehat{\mu}_j$ concentrated around 1 and test statistics bounded away from 0. It is therefore relatively easy to identify alternatives/signals from the nulls. From the eigenvalue ratio plot, we see that the fifth ratio (highlighted as a red dot) is evidently above the others, thus determining the number of factors. The scree plot, on the other hand, reveals that the top 5 eigenvalues (above the red dashed line) together explain the vast majority of the variance, indicating that the proportion of common variance (due to common factors) is high.

Output	Implication	Data type	R class
`means`	estimated means	p-vector	matrix
`stdDev`	estimated standard deviations	p-vector	matrix
`loadings`	estimated loading matrix	$(p \times K)$-matrix	matrix
`eigenVal`	eigenvalues of estimated covariance	p-vector	matrix
`eigenRatio`	eigenvalue ratios of estimated covariance	(min $\{n, p\}$ /2)-vector	matrix
`nFactors`	(estimated) number of factors	positive integer	integer
`tStat`	test statistics	p-vector	matrix
`pValues`	p-values	p-vector	matrix
`pAdjust`	adjusted p-values	p-vector	matrix
`significant`	indicators of significance	boolean p-vector	matrix
`reject`	indices of rejected hypotheses	vector	integer
`type`	whether factor is known	string	character
`n`	sample size	positive integer	integer
`p`	data dimension	positive integer	integer
`h0`	null hypothesis	p-vector	numeric
`alpha`	nominal FDR level	numerical number	numeric
`alternative`	alternative hypothesis	string	character

Table 1: Objects in the output list of `farm.test` function with their implications, and description of data type and class in R language.

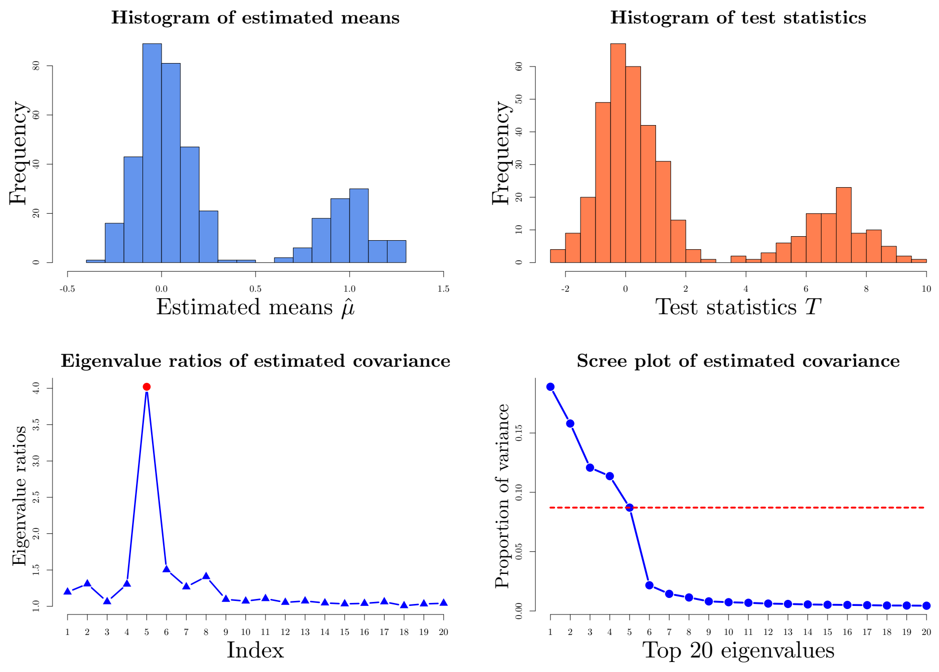


Figure 1: Upper panel: histograms of estimated means and test statistics. Lower panel: eigenvalue ratio plot with the largest ratio highlighted and scree plot of the eigenvalues of the estimated covariance matrix.

Function call with options

In this section, we illustrate `farm.test` function with other options that allow us to call it more flexibly. When the factors are observable, we can simply put the $n \times K$ factor matrix into argument `fX`, and the output is formatted the same as before. As a remark, among all the items listed in Table 1, `eigenVal` and `eigenRatio`, which are eigenvalues and eigenvalue ratios of estimated covariance matrix, are not available in this case; see Algorithm 1.

```
output <- farm.test(X, fX = FX)
output

One-sample FarmTest with known factors
n = 120, p = 400, nFactors = 5
```

```
FDR to be controlled at: 0.05
Alternative hypothesis: two.sided
Number of hypotheses rejected: 101
```

Consider one-sided alternatives $H_{1j} : \mu_j \geq 0, j = 1, \ldots p$ with a nominal FDR level 1%. We modify the arguments `alternative` and `alpha` as follows:

```
output <- farm.test(X, alternative = "greater", alpha = 0.01)
output

One-sample FarmTest with unknown factors
n = 120, p = 400, nFactors = 5
FDR to be controlled at: 0.01
Alternative hypothesis: greater
Number of hypotheses rejected: 101
```

Users can specify null hypotheses by passing any vector with length p into argument `h0`. In the next example, we consider the p null hypotheses as all the means are equal to 1, so that the number of true nonnulls becomes 300.

```
output <- farm.test(X, h0 = rep(1, p), alpha = 0.01)
output

One-sample FarmTest with unknown factors
n = 120, p = 400, nFactors = 5
FDR to be controlled at: 0.01
Alternative hypothesis: two.sided
Number of hypotheses rejected: 300
```

When the factors are unknown, users can also specify the number of factors based on some subjective grounds. In this case, Step 3 in Algorithm 2 is avoided. For example, we run the function with the number of factors chosen to be `KX = 2`, which is less than the true parameter 5. This misspecification results in a loss of power with two true alternatives unidentified.

```
output <- farm.test(X, KX = 2)
power <- sum(output$reject <= p1) / p1
power

[1] 0.98
```

As a special case, if we declare `KX = 0` in the function, a robust multiple test without factor-adjustment is conducted.

```
output <- farm.test(X, KX = 0)
output

One-sample robust multiple test without factor-adjustment
n = 120, p = 400
FDR to be controlled at: 0.05
Alternative hypothesis: two.sided
Number of hypotheses rejected: 95
```

Finally, we present an example of two-sample FarmTest. Using the same sampling distributions for the factor loading matrix, factors and noise vectors, we generate another sample $\{Y_i\}_{i=1}^m$ from model (5) with $m = 150$.

```
m <- 150
set.seed(200)
BY <- rustiefel(p, K) %*% diag(rep(sqrt(p), K))
FY <- rmvnorm(m, rep(0, K), diag(K))
uY <- rmvt(m, diag(p), 3)
Y <- FY %*% t(BY) + uY
```

Then `farm.test` function can be called with an additional argument `Y`.

```
output <- farm.test(X, Y = Y)
output

Two-sample FarmTest with unknown factors
X.n = 120, Y.n = 150, p = 400, X.nFactors = 5, Y.nFactors = 5
FDR to be controlled at: 0.05
Alternative hypothesis: two.sided
Number of hypotheses rejected: 105
```

The output is formatted similarly as in Table 1, except that `means`, `stdDev`, `loadings`, `eigenVal`, `eigenRatio`, `nFactors` and `n` now consist of two items for samples `X` and `Y`.

```
names(output$means)

[1] "X.mean" "Y.mean"
```

Simulations

In this section, we assess and compare the performance of `farm.test` function in the **FarmTest** package with the following methods:

- t-test using the R built-in function `t.test`;
- WMW-test (Wilcoxon-Mann-Whitney) using the `onesamp.marginal` function in the **mutoss** package;
- RmTest (Robust Multiple test) without factor-adjustment by claiming `KX = 0` in the `farm.test` function.

For t-test and WMW-test, the functions we call produce vectors of p-values, to which the method proposed in Storey (2002) is applied, see Steps 5–7 in Algorithm 1 or Steps 8–10 in Algorithm 2.

In all the numerical experiments, we consider two-sided alternatives with a nominal FDR level $\alpha = 5\%$. The true number of factors is 5. Factors and loadings are generated the same way as in A showcase example Section. To add dependency among idiosyncratic errors, the covariance matrix of $\boldsymbol{u}$, denoted by $\boldsymbol{\Sigma}_u$, is taken to be a block-diagonal symmetric matrix with block size 5×5. Within each block, the diagonal entries are all equal to 3 and the off-diagonal entries are generated from $\mathcal{U}[0,1]$. In the simulations, we drop the case where the generated $\boldsymbol{\Sigma}_u$ is not positive-definite. The distribution of $\boldsymbol{u}$ is specified in two models as follows.

- **Model 1**. $\boldsymbol{u} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{\Sigma}_u)$: centered multinormal distribution with covariance matrix $\boldsymbol{\Sigma}_u$;
- **Model 2**. $\boldsymbol{u} \sim t_3(\boldsymbol{0}, \boldsymbol{\Sigma}_u)$: multivariate t-distribution with degrees of freedom 3 and covariance matrix $\boldsymbol{\Sigma}_u$.

For each model, we consider various combinations of sample size n and dimensionality p, specifically, $n \in \{60, 80, 100, 120, 140\}$ and $p \in \{200, 400, 600, 800, 1000\}$. The number of true alternatives p_1 is taken to be $0.2\,p$, and the signal strength is set as $4\sqrt{\log(p)/n}$.

Figures 2 and 3 depict the FDR and power curves for either "fixed n growing p" or "fixed p growing n" based on 200 simulations. Across various settings, FarmTest consistently maintains high empirical power with FDR well controlled around the nominal level. In contrast, the competing methods may lose as many as 10% to 30% powers, which can be ascribed to not accounting for the common factors. In summary, we conclude that the **FarmTest** package provides an efficient implementation of the FarmTest method, which carries out multiple testing for multivariate data with heavy-tailed distribution and a strong dependency structure.

Real data example

In this section, we apply the **FarmTest** package to test the mean effects of stock returns. In capital asset pricing theory, the stock's risk-adjusted mean return or "alpha" is a quantity of interest since it indicates the excessive return incurred from investing in a particular stock. If the efficient equity market hypothesis holds, we expect "alpha" to be zero. Hence, detecting non-zero alphas can help investors to identify market inefficiencies, that is, whether certain stocks exhibit an abnormal rate

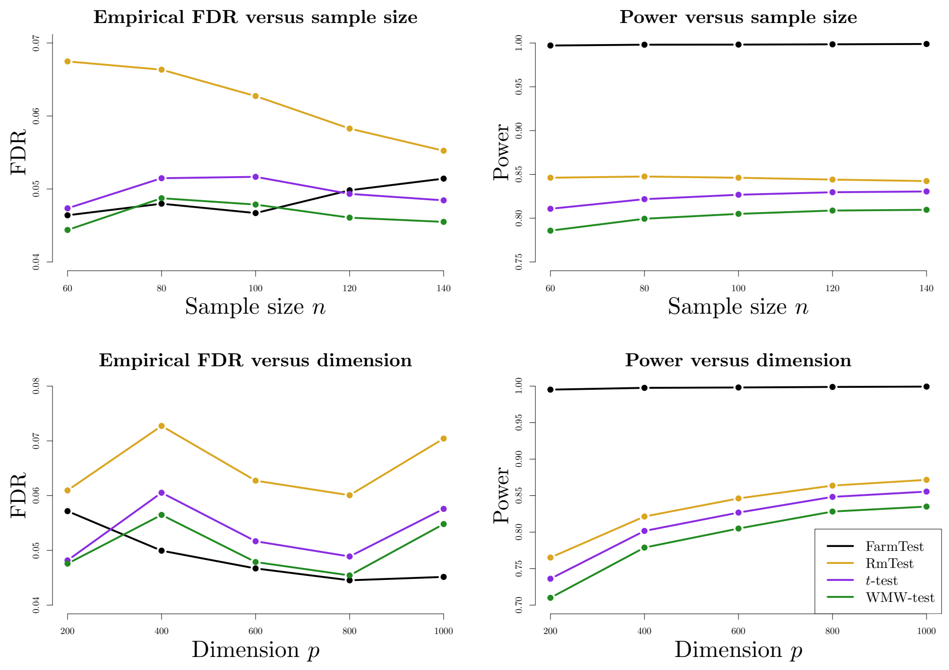


Figure 2: Comparison of FarmTest with three other methods in terms of FDR and power under Model 1 (multivariate normal distribution). In the upper panel, p is fixed at 600 and n grows from 60 to 140; in the lower panel, n is fixed at 100 and p ranges from 200 to 1000.

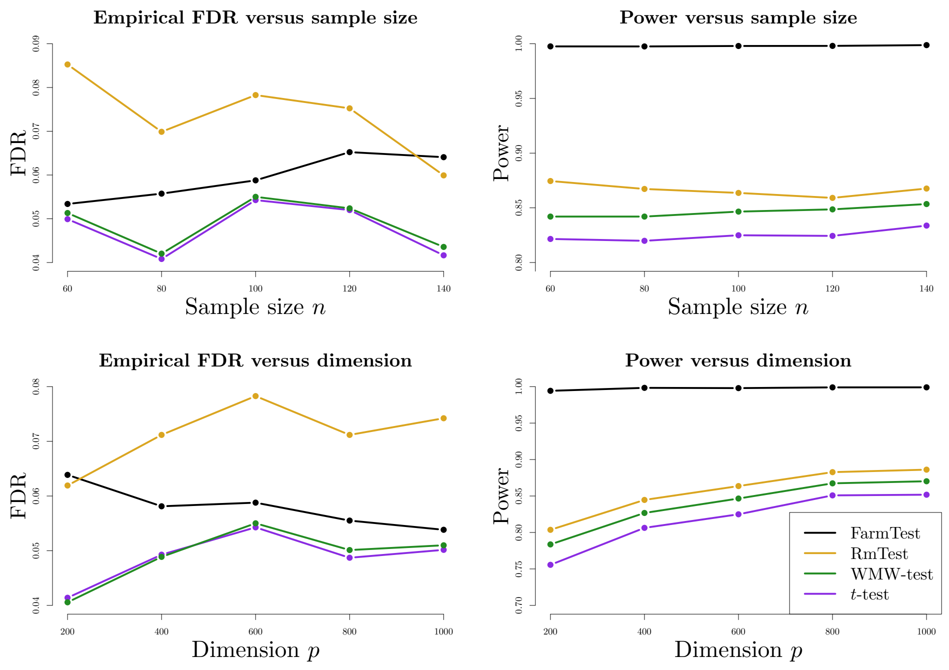


Figure 3: Comparison of FarmTest with three other methods in terms of FDR and power under Model 2 (multivariate t-distribution). In the upper panel, p is fixed at 600 and n grows from 60 to 120; in the lower panel, n is fixed at 100 and p ranges from 200 to 1000.

of return or are mispriced. As discussed in Cont (2001), both cross-sectional dependency and heavy tailedness are silent features of stock returns.

In this study, we test the annual mean effects of stocks in the S&P500 index. The data is available on COMPUSTAT and CRSP databases. We find that most of the stocks with continuous membership in the S&P500 index from 2008 to 2016 have excess kurtosises greater than zero, indicating tails heavier than that of a normal distribution. Also, more than 33% of the stocks are severely heavy-tailed as their

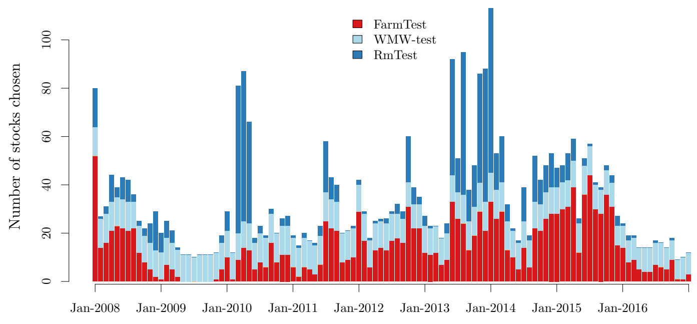


Figure 4: Stack bar plot of the numbers of discoveries via FarmTest, WMW-test and RmTest from 2008 to 2016, using rolling windows of one year. Within each time window, we report the number of stocks in the S&P500 index that show significant statistical evidence against null hypotheses that there are no excessive returns, with FDR controlled at 1%.

excess kurtosises exceed 6, which is the excess kurtosis of t_5-distribution. We collect monthly returns of stocks from the S&P500 index over rolling windows: for each month between 2008 and 2016, we collect monthly returns of stocks who have continuous records over the past year. The average number of stocks collected each year is 598. For each rolling window, we conduct multiple testing using the four methods considered in the previous section, that is, FarmTest, t-test, WMW-test, and RmTest.

The nominal FDR level is set as $\alpha = 1\%$. Within each rolling time window, we have $p \approx 600$ and $n = 12$. The numbers of discoveries of each method are depicted chronologically in Figure 4, and Table 2 displays several key summary statistics. Since the t-test barely discovers any stock throughout the whole procedure, we only present the results for the other three methods in Figure 4. It is interesting to observe that across different time rolling windows, the testing outcomes of the WMW-test are relatively stable and time-insensitive. FarmTest, on the other hand, selects much fewer stocks in the year of 2009, coinciding to some extent with the financial crisis during which the market volatility is much higher. RmTest typically selects the most stocks, which is partly due to the lack of FDR control under strong dependency. A major, noticeable impact of dependence is that it results in clusters of rejections: if a test is rejected, then there are likely to be further rejections for tests that are highly correlated with this one. This phenomenon is in accord with our simulation results, showing that FarmTest simultaneously controls the FDR and maintains high power while the other methods either make too many false discoveries or fail to detect true signals.

Method	Mean	Std. Dev.	Median	Min	Max
FarmTest	14.477	11.070	12	0	52
WMW-test	10.991	1.005	11	8	12
RmTest	8.147	14.414	3	0	68

Table 2: Summary statistics of the number of discoveries via FarmTest, WMW-test and RmTest between 2008 and 2016 using rolling windows of size 12 (months).

Summary

We provide an R package to implement FarmTest, a flexible large-scale multiple testing method that is robust against strongly dependent and heavy-tailed data. The factor-adjustment procedure helps to construct weakly dependent test statistics, and also enhances statistical power by reducing the signal-to-noise ratio. Moreover, by exploiting the idea of adaptive Huber regression, the testing procedure is robust against heavy-tailed noise. The efficacy of our package is demonstrated on both real and simulated datasets.

Bibliography

S. C. Ahn and A. R. Horenstein. Eigenvalue ratio test for the number of factors. *Econometrica*, 81(3): 1203–1227, 2013. URL https://doi.org/10.3982/ECTA8968. [p394]

J. Bai and K. Li. Statistical analysis of factor models of high dimension. *The Annals of Statistics*, 40(1): 436–465, 2012. URL https://doi.org/10.1214/11-AOS966. [p391]

J. Barzilai and J. M. Borwein. Two-point step size gradient methods. *IMA Journal of Numerical Analysis*, 8(1):141–148, 1988. URL https://doi.org/10.1093/imanum/8.1.141. [p394]

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 (Methodological)*, 57(1):289–300, 1995. URL https://doi.org/10.1111/j.2517-6161.1995.tb02031.x. [p389]

X. Chen and W.-X. Zhou. Robust inference via multiplier bootstrap. *The Annals of Statistics*, 2019. URL https://arxiv.org/abs/1903.07208. [p391]

R. Cont. Empirical properties of asset returns: stylized facts and statistical issues. *Quantitative Finance*, 1(2):223–236, 2001. URL https://doi.org/10.1080/713665670. [p399]

K. H. Desai and J. D. Storey. Cross-dimensional inference of dependent high-dimensional data. *Journal of the American Statistical Association*, 107(497):135–151, 2011. URL https://doi.org/10.1080/01621459.2011.645777. [p389]

D. Eddelbuettel and R. Francois. Rcpp: Seamless R and C++ integration. *Journal of Statistical Software*, 40(8):1–18, 2011. URL https://doi.org/10.18637/jss.v040.i08. [p389]

D. Eddelbuettel and C. Sanderson. RcppArmadillo: Accelerating R with high-performance C++ linear algebra. *Computational Statistics and Data Analysis*, 71:1054–1063, 2014. URL https://doi.org/10.1016/j.csda.2013.02.005. [p389]

J. Fan and X. Han. Estimation of the false discovery proportion with unknown dependence. *Journal of the Royal Statistical Society: Series B (Methodological)*, 79(4):1143–1164, 2017. URL https://doi.org/10.1111/rssb.12204. [p389]

J. Fan, X. Han, and W. Gu. Estimating false discovery proportion under arbitrary covariance dependence. *Journal of the American Statistical Association*, 107(499):1019–1035, 2012. URL https://doi.org/10.1080/01621459.2012.720478. [p389]

J. Fan, Q. Li, and Y. Wang. Estimation of high dimensional mean regression in the absence of symmetry and light tail assumptions. *Journal of the Royal Statistical Society: Series B (Methodological)*, 79(1): 247–265, 2017. URL https://doi.org/10.1111/rssb.12166. [p391]

J. Fan, Y. Ke, Q. Sun, and W.-X. Zhou. FarmTest: Factor-adjusted robust multiple testing with approximate false discovery control. *Journal of the American Statistical Association*, 114(528):1880–1893, 2019. URL https://doi.org/10.1080/01621459.2018.1527700. [p389, 390, 391, 393]

C. Friguet, M. Kloareg, and D. Causeur. A factor model approach to multiple testing under dependence. *Journal of the American Statistical Association*, 104(488):1406–1415, 2009. URL https://doi.org/10.1198/jasa.2009.tm08332. [p389, 393]

C. Genovese and L. Wasserman. A stochastic process approach to false discovery control. *The Annals of Statistics*, 32(3):1035–1061, 2004. URL https://doi.org/10.1214/009053604000000283. [p389]

P. D. Hoff. Simulation of the matrix Bingham–von Mises–Fisher distribution, with applications to multivariate and relational data. *Journal of Computational and Graphical Statistics*, 18(2):438–456, 2012. URL https://doi.org/10.1198/jcgs.2009.07177. [p394]

T. Hothorn, F. Bretz, and P. Westfall. Simultaneous inference in general parametric models. *Biometrical Journal*, 50(3):346–363, 2008. URL https://doi.org/10.1002/bimj.200810425. [p390]

P. J. Huber. Robust estimation of a location parameter. *The Annals of Mathematical Statistics*, 35(1): 73–101, 1964. URL https://doi.org/10.1214/aoms/1177703732. [p391]

Y. Ke, S. Minsker, Z. Ren, Q. Sun, and W.-X. Zhou. User-friendly covariance estimation for heavy-tailed distributions. *Statistical Science*, 34(3):454–471, 2019. URL https://doi.org/10.1214/10.1214/19-STS711. [p390, 391, 394]

C. Lam and Q. Yao. Factor modeling for high-dimensional time series: Inference for the number of factors. *The Annals of Statistics*, 40(2):694–726, 2012. URL https://doi.org/10.1214/12-AOS970. [p394]

W. Lan and L. Du. A factor-adjusted multiple testing procedure with application to mutual fund selections. *Journal of Business and Economic Statistics*, 37(1):147–157, 2019. URL https://doi.org/10.1080/07350015.2017.1294078. [p393]

J. T. Leek and J. D. Storey. A general framework for multiple testing dependence. *Proceedings of the National Academy of Sciences of the United States of America*, 105(48):18718–18723, 2008. URL https://doi.org/10.1073/pnas.0808709105. [p389, 392, 393]

E. L. Lehmann and J. P. Romano. Generalizations of the familywise error rate. *The Annals of Statistics*, 33(3):1138–1154, 2005. URL https://doi.org/10.1214/009053605000000084. [p389]

K. S. Pollard, S. Dudoit, and M. J. van der Laan. Multiple testing procedures: the multtest package and applications to genomics. *Bioinformatics and Computational Biology Solutions Using R and Bioconductor, Springer*, pages 249–271, 2005. URL https://link.springer.com/chapter/10.1007/0-387-29362-0_15. [p390]

C. Sanderson and R. Curtin. Armadillo: A template-based C++ library for linear algebra. *Journal of Open Source Software*, 1(2):26, 2016. URL https://doi.org/10.21105/joss.00026. [p389]

J. Storey. A direct approach to false discovery rates. *Journal of the Royal Statistical Society: Series B (Methodological)*, 64(3):479–498, 2002. URL https://doi.org/10.1111/1467-9868.00346. [p389, 390, 391, 398]

Q. Sun, W.-X. Zhou, and J. Fan. Adaptive Huber regression. *Journal of the American Statistical Association*, 115(529):254–265, 2020. URL https://doi.org/10.1080/01621459.2018.1543124. [p390, 391]

L. Wang, C. Zheng, W. Zhou, and W.-X. Zhou. A new principle for tuning-free huber regression. *Statistica Sinica*, to appear, 2020. URL https://doi:10.5705/ss.202019.0045. [p390, 391, 394]

W.-X. Zhou, K. Bose, J. Fan, and H. Liu. A new perspective on robust M-estimation: Finite sample theory and applications to dependence-adjusted multiple testing. *The Annals of Statistics*, 46(5): 1904–1931, 2018. URL https://doi.org/10.1214/17-AOS1606. [p390, 391, 392, 394]

Koushiki Bose, Jianqing Fan
Department of Operations Research and Financial Engineering
Princeton University, Princeton, NJ 08544
USA
koush.bose@gmail.com, jqfan@princeton.edu

Yuan Ke
Department of Statistics
University of Georgia, Athens, GA 30602
USA
Yuan.Ke@uga.edu

Xiaoou Pan, Wen-Xin Zhou
Department of Mathematics
University of California, San Diego, La Jolla, CA 92093
USA
xip024@ucsd.edu, wez243@ucsd.edu

Changes in R 3.6–4.0

by Tomas Kalibera, Sebastian Meyer and Kurt Hornik

Abstract We give a selection of the most important changes in R 4.0.0 and in the R 3.6 release series. Some statistics on source code commits and bug tracking activities are also provided.

R 4.0.0 selected changes

R 4.0.0 (codename "Arbor Day") was released on 2020-04-24. The following gives a selection of the most important changes.

- `matrix` objects now also inherit from class `"array"`, so e.g., `class(diag(1))` is `c("matrix","array")`. S3 methods for class `"array"` are now dispatched for `matrix` objects. This reduces the need of code duplication between `"array"` and `"matrix"` classes, but invalidates code incorrectly assuming that `class(matrix_obj))` has length one. In principle, to check whether an object inherits from (any) class, one should always use `inherits()` (or `is()`). See Martin Maechler's blog post for more details.
- There is a new syntax for specifying *raw* character constants similar to the one used in C++: `r"(...)"` with `...` any character sequence not containing the sequence '`)"`'. This makes it easier to write strings that contain backslashes and/or both single and double quotes:

  `r"(c:\Program files\R)"` specifies a Windows directory without escaping backslashes. `r"(use both "double" and 'single' quotes)"` mixes single and double quotes without the need to escape either of them. For more details see `?Quotes`.
- R now uses a '`stringsAsFactors = FALSE`' default, and hence by default no longer converts strings to factors in calls to `data.frame()` and `read.table()`. Automatic conversion of strings to factors regardless of the context of the study at hand seems conceptually wrong. In addition, when the automatically applied order is lexicographical order, the result is locale dependent and even so when only ASCII characters are used. Historically, automatic conversions to factors could have been disabled on demand, but unfortunately that meant that all code dealing with data frames would have to support both ways. That was not the case, leading to surprising or unpredictable results. A large number of packages relied on the previous behavior and so have needed updating. Unlike in the case of matrices being treated as arrays, this was a change to documented behavior, so even correct package code was affected. See Kurt Hornik's blog post for more details.
- Reference counting is now used instead of the `NAMED` mechanism for determining when objects can be safely mutated in base C code. This reduces the need for copying in some cases and should allow further optimizations in the future. It should help make the internal code easier to maintain. In principle, even the `NAMED` mechanism was a variant of reference counting, but a simple one where the number of references could only increase (up to a maximum value). Even as simple operations as passing an R object (value) to a function that would only read it would permanently increase the reference count of that object, even after that reading-only function would return. Any modification of that object later on would require a copy. This is one of the scenarios fixed by the new mechanism where the reference counts can and often do decrease as well, so that R knows much more often that some R values are in fact private and can be modified in place. This change should not impact existing code (does not break packages) using supported coding practices in C/C++. It has no direct impact on R code other than performance/memory usage.
- R now has a listening server socket object which allows to accept multiple incoming socket connections. This simplifies implementation of servers and allows them to accept multiple connections much faster. The time needed to set up a PSOCK cluster has been reduced using this new API particularly for clusters with a large number of nodes. See a blog post of Tomas Kalibera and Luke Tierney for more details.
- S3 method lookup now by default skips the elements of the search path between the global and base environments, and there is a new function `.S3method()` to register S3 methods in R scripts. See Kurt Hornik's blog post for more details.
- The `palette()` function has a new default set of colors which are less saturated and have better accessibility properties. There are also some new built-in palettes, which are listed by the new `palette.pals()` function. The new `palette.colors()` function allows a subset of colors to be selected from any of the built-in palettes. See a blog post of Achim Zeileis, Paul Murrell, Martin Maechler, and Deepayan Sarkar for more details.

- The internal implementation of **grid** units has changed, but the only visible effects at user-level should be a slightly different print format for some units (especially unit arithmetic), better performance (for unit operations) and two new functions `unitType()` and `unit.psum()`. Packages that were directly accessing elements of the unit implementation needed updating. See a blog post by Paul Murrell and Thomas Lin Pedersen for more details.
- The support for symbol fonts in cairo-based graphics devices has been improved and one can now specify which symbol font to use. See Paul Murrell's blog post for more details.

See `https://CRAN.R-project.org/doc/manuals/r-patched/NEWS.html` for all changes in the current release series of R, which at the time of this writing is R 4.0.z. Overall, there are 156 news entries for the 4.0.0 release, including 5 significant user-visible changes, 65 new features and 55 bug fixes.

R 3.6.z selected changes

R 3.6.0 (codename "Planting of a Tree") was released on 2019-04-26 and the R 3.6 series closed with the release of R 3.6.3 ("Holding the Windsock") on 2020-02-29, marking the 20th anniversary of the R 1.0.0 release. The following gives a selection of the most important changes in the 3.6 series.

- The default method for generating from a discrete uniform distribution (used in `sample()`, for instance) has been changed. This addresses the fact, pointed out by Ottoboni and Stark, that the previous method made `sample()` noticeably non-uniform on large populations. See PR#17494 for a discussion. The previous method can be requested using `RNGkind()` or `RNGversion()` if necessary for reproduction of old results. Thanks to Duncan Murdoch for contributing the patch and Gabe Becker for further assistance.

  The output of `RNGkind()` has been changed to also return the 'kind' used by `sample()`.
- Serialization format version 3 becomes the default for serialization and saving of the workspace (`save()`, `serialize()`, `saveRDS()`, `compiler::cmpfile()`). Serialized data in format 3 cannot be read by versions of R prior to version 3.5.0. Serialization format version 2 is still supported and can be selected by `version = 2` in the save/serialization functions. The default can be changed back for the whole R session by setting environment variables `R_DEFAULT_SAVE_VERSION` and `R_DEFAULT_SERIALIZE_VERSION` to `2`. For maximal back-compatibility, files 'vignette.rds' and 'partial.rdb' generated by `R CMD build` are in serialization format version 2, and resave by default produces files in serialization format version 2 (unless the original is already in format version 3). The new serialization format is already supported since R version 3.5.0. It allows compact representation of ALTREP objects, so that e.g. compact integer sequences are saved as compact. All elements of such sequence have to be enumerated in format version 2. The new serialization format also saves the current local encoding at the time of serialization and strings in native encoding are translated when de-serialized in an R session with different native encoding.
- `library()` and `require()` now allow more control over handling search path conflicts when packages are attached. The policy is controlled by the new `conflicts.policy` option. See Luke Tierney's blog post for more details.
- R now uses staged installation of R packages. A package is first installed into a temporary library invisible to other R sessions and then moved to the final library location. This reduces interference due to partially installed packages which has been observed particularly during parallel installation. See Tomas Kalibera's blog post for more details.
- New `hcl.colors()` function to provide wide range of HCL-based color palettes with much better perceptual properties than the existing RGB/HSV-based palettes like `rainbow()`. Also a new `hcl.pals()` function to list available palette names for `hcl.colors()`. Contributed by Achim Zeileis. See blog post of Achim Zeileis and Paul Murrell for more details.
- There are two new options, `keep.parse.data` and `keep.parse.data.pkgs`, which control whether parse data are included into source (source references) when `keep.source` or `keep.source.pkgs` is `TRUE`. By default, `keep.parse.data.pkgs` is now `FALSE`, which changes previous behavior and significantly reduces space and time overhead when sources are kept when installing packages. See Tomas Kalibera's blog post for more details on this and other performance optimizations in the parser.
- R 3.6.2 has been fixed to pass hidden string length arguments when calling LAPACK from C. Macros were provided also for packages that call LAPACK directly. This was urgently needed after a new GNU Fortran release introduced optimizations which caused crashes with code calling LAPACK (or other Fortran code) the "old way", yet widely used in numerical software including CBLAS and LAPACKE itself. GNU Fortran disabled again these optimizations by default in later releases as a result of these findings. For more details, see Writing R Extensions

and the first and second blog post by Tomas Kalibera on this issue (the changes in R were implemented and documented by Brian Ripley).

- New pointer protection C functions `R_PreserveInMSet` and `R_ReleaseFromMSet` have been introduced to replace `UNPROTECT_PTR`, which is not safe to mix with `UNPROTECT` (and with `PROTECT_WITH_INDEX`). Intended for use in parsers only. See Tomas Kalibera's blog post for more details.
- `S3method()` directives in 'NAMESPACE' can now also be used to perform *delayed* S3 method registration. Again, see Kurt Hornik's blog post for more details.

See `https://CRAN.R-project.org/doc/manuals/r-devel/NEWS.3.html` for all changes in the R 3.y.z releases. Overall, there are 233 news entries for the 3.6.z releases, including 2 significant user-visible changes, 75 new features and 106 bug fixes.

R 4.0.0 code statistics

From the source code Subversion repository, changes between April 27, 2019 and April 24, 2020, so the overall code change between R 3.6.0 and R 4.0.0 was: over 24,000 added lines, 12,000 deleted lines and 900 changed files. This is rounded to thousands/hundreds and excludes changes to common generated files, partially generated files, bulk re-organizations, etc. (translations, parsers, autoconf, LAPACK, R Journal bibliography, test outputs).

Figure 1 shows commits by month and weekday, respectively, counting line-based changes in individual commits, excluding the files as above. A noticeable increase of activity is in March, so right before code freeze for the release. A secondary peak of the number of commits can be observed in August. The low amount of changes in July 2019 may be due to conferences and vacations.

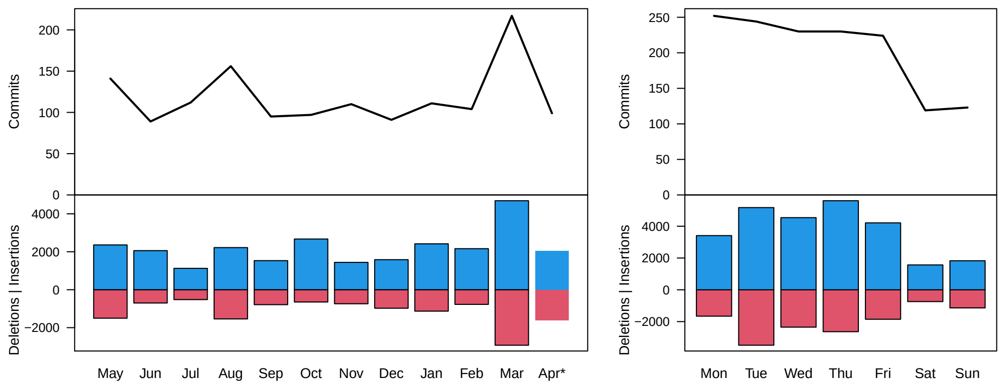


Figure 1: Commit statistics by month (left) and weekday (right) during R 4.0.0 development. *Note that the counts for April don't correspond to a unique month.

R 3.6.0 code statistics

Changes between April 23, 2018 and April 26, 2019, so the overall code change between R 3.5.0 and R 3.6.0 was: nearly 27,000 added lines, over 17,000 deleted lines and nearly 800 changed files. This is again rounded to thousands/hundreds and excludes changes to common generated files.

Figure 2 again shows large changes in March before code freeze and in August, and decreased activity in July during R conferences and usual vacations. The right panel suggests that R Core members work a lot even during the weekends and it was even more so when working on R 3.6.0 than on R 4.0.0 (compare Saturday and Wednesday).

R 4.0.0 bugs statistics

Summaries of bug-related activities during the development of R 4.0.0 (from April 27, 2019 to April 24, 2020) were derived from the database underlying R's Bugzilla system. Figure 3 shows statistics of reported/closed bugs and number of added comments (on any bug report) by calendar month and weekday, respectively.

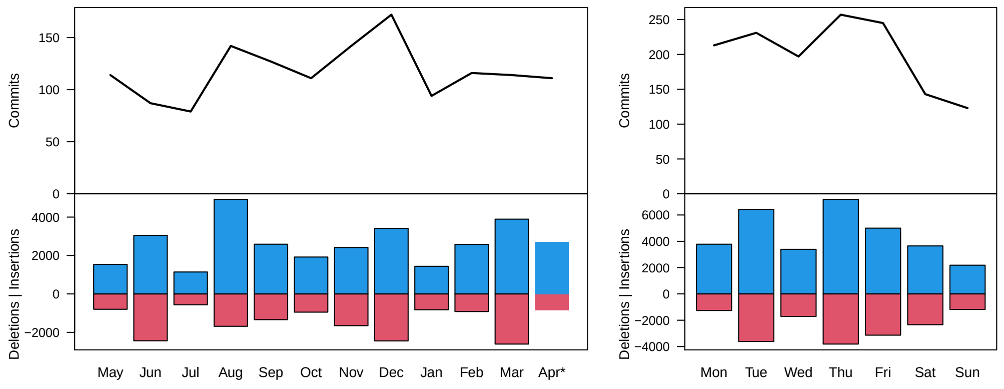


Figure 2: Commit statistics by month (left) and weekday (right) during R 3.6.0 development. *Note that the counts for April don't correspond to a unique month.

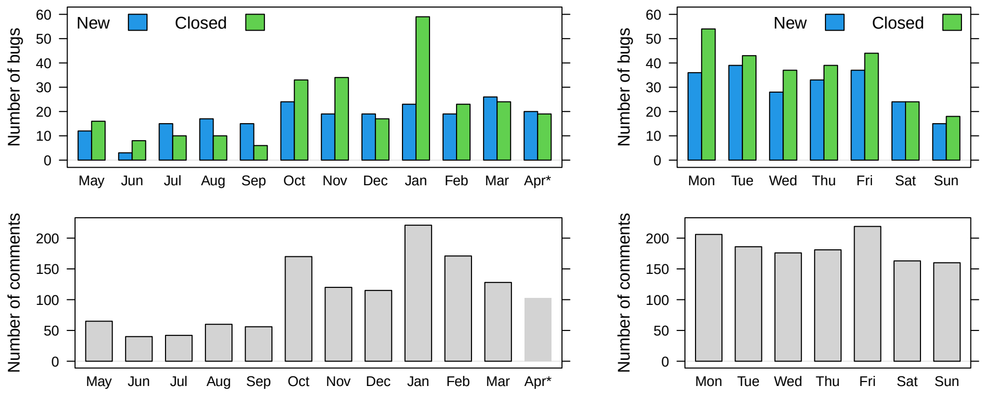


Figure 3: Bug tracking activity by month (left) and weekday (right) during R 4.0.0 development. *Note that the counts for April don't correspond to a unique month.

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.

The numbers were impacted by an increase in external contributions to analyzing bugs following a blog post of Tomas Kalibera and Luke Tierney, published October 9, 2019, asking the R community for help, and to contribute those analyzes in the form of comments to R bug reports. There was a considerable increase of comments in October which has lasted (at least) until April. Note that the April numbers don't cover a full month and are mostly from the 24 days of R 4.0 development in 2020, so after the blog post (4 days are from April 2019). The rate of closing bugs has increased as well since October. What the numbers don't show is that this is also due to increased activity of R Core that followed increased input from external volunteers. The numbers also seem to suggest that even new bug reports are submitted at a higher rate once more external volunteers focus on analyzing bugs in R.

From the numbers by weekday in the right panel of Figure 3 we again see that the R community keeps working during the weekends.

R 3.6.0 bugs statistics

Figure 4 summarizes bug tracking activities during the development of R 3.6.0 (from April 23, 2018 to April 26, 2019). The decline observed in coding activity in July does not exist in bug-related activities; the number of closed bugs actually peaked in July.

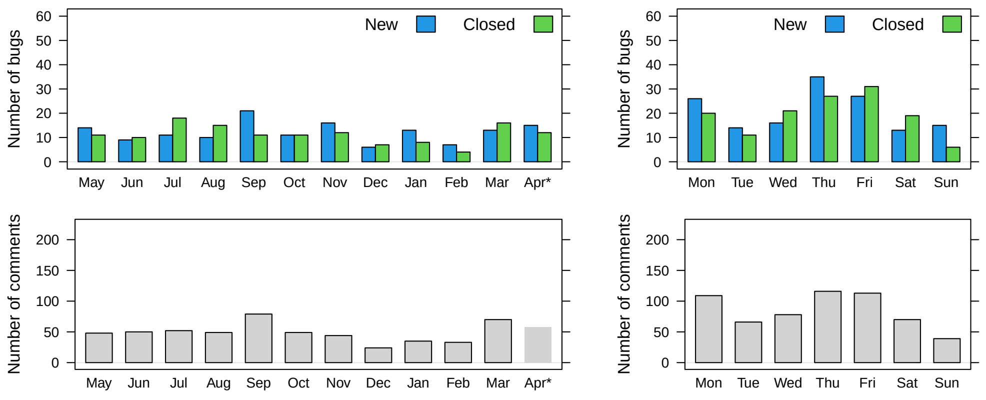


Figure 4: Bug tracking activity by month (left) and weekday (right) during R 3.6.0 development. *Note that the counts for April don't correspond to a unique month. For comparison with R 4.0.0, the y-axes use the same scales as in Figure 3.

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, and the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme, under grant agreement No. 695412.

Tomas Kalibera
Czech Technical University, Czech Republic
`Tomas.Kalibera@R-project.org`

Sebastian Meyer
Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany
`seb.meyer@fau.de`

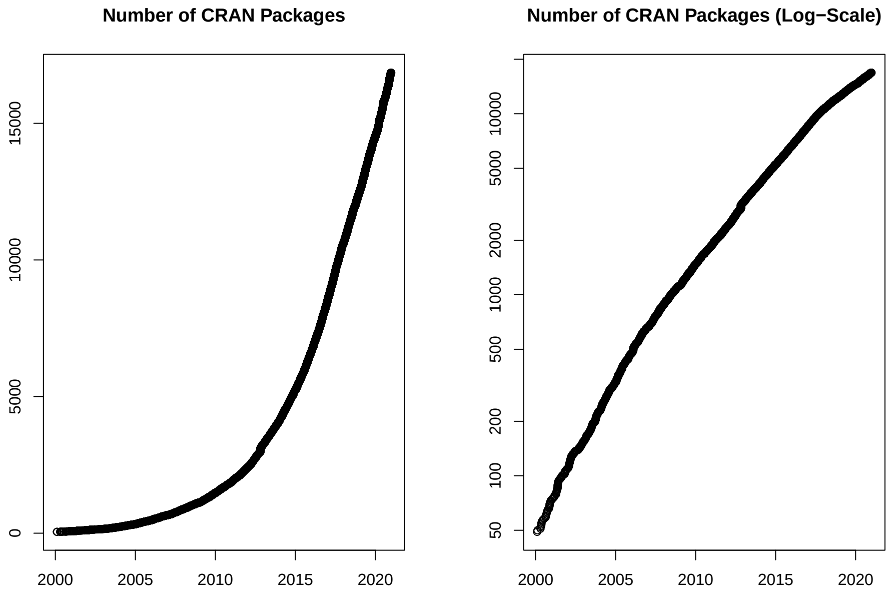
Kurt Hornik
WU Wirtschaftsuniversität Wien, Austria
`Kurt.Hornik@R-project.org`

Changes on CRAN

2020-09-01 to 2020-12-31

by Kurt Hornik, Uwe Ligges and Achim Zeileis

In the past 4 months, 818 new packages were added to the CRAN package repository. 100 packages were unarchived and 248 were archived. The following shows the growth of the number of active packages in the CRAN package repository:



On 2020-12-31, the number of active packages was around 16851.

Changes in the CRAN Repository Policy

The Policy now says the following:

- For R version 4.0 or later (hence a version dependency is required or only conditional use is possible), packages may store user-specific data, configuration and cache files in their respective user directories obtained from `tools::R_user_dir()`, provided that by default sizes are kept as small as possible and the contents are actively managed (including removing outdated material).
- Security provisions must not be cicrumvented, for example by not verifying SSL certificates.
- Downloads of additional [.........] For downloads of more than a few MB, ensure that a sufficiently large timeout is set.
- For a package update, please check that any packages depending on this one still pass `R CMD check`: [.........] If possible, check reverse strong dependencies, reverse suggests and the recursive strong dependencies of these (by `tools::package_dependencies(reverse = TRUE, which = "most", recursive = "strong")`).

The CRAN URL checks info now says

- The CRAN submission checks run by `R CMD check --as-cran` check the availability of URLs in files including 'DESCRIPTION', 'CITATION', 'NEWS.Rd', 'NEWS.md', 'README.md', and the '.Rd' help pages and HTML files in 'inst/doc'.
- A surprisingly large number of websites use redirection and the issues may apply to a site redirected to. [........] Where redirection is permanent you should use the redirected URL (see RFC 7231).

CRAN package submissions

CRAN mirror security

Currently, there are 104 official CRAN mirrors, 77 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 **BGVAR**, **LAWBL**, **bayestestR**, **blavaan**, **loo**.

Cluster **FCPS**, **crimCV**.

Distributions **CaDENCE**, **DPQ**, **Distributacalcul**, **ForestFit**, **MPS**, **NonNorMvtDist**, **PoissonBinomial**, **QBAsyDist**, **ROOPSD**, **betafunctions**, **cort**, **dgumbel**, **distributional**, **distributionsrd**, **elfDistr**, **ggamma**, **mniw**, **scModels**, **tvgeom**.

Econometrics **NNS**.

Finance **FFdownload**, **bmgarch**, **garchx**, **simfinapi**.

FunctionalData **FDboost***, **fdaoutlier**, **refund***.

Genetics **SNPassoc**.

HighPerformanceComputing **flexiblas**.

Hydrology **weathercan**.

MachineLearning **mlr3proba**.

MetaAnalysis **boot.heterogeneity**, **clubSandwich**, **concurve**, **estimraw**, **gemtc**, **metabolic**, **multinma**.

MissingData **SNPassoc**.

OfficialStatistics **reclin**.

Optimization **irace**, **qpmadr**.

Psychometrics **EstimateGroupNetwork**, **LAWBL**, **betafunctions**, **cops**.

ReproducibleResearch **ascii**, **flextable**, **flowr**, **groundhog**, **liftr**, **mschart**, **officer**, **openxlsx**, **readODS**, **switchr**, **trackr**, **tth**, **worcs**, **xaringan**, **zoon**.

Robust **rlme**.

TeachingStatistics **bivariate**.

TimeSeries **FKF.SP**, **FoReco**, **RobKF**, **TSA**, **TSdist**, **breakfast**, **diffusion**, **fredr**, **greybox**, **ifultools**, **modeltime**, **modeltime.ensemble**, **mssm**, **portes**, **readabs**, **tfarima**, **tsibbletalk**, **tsutils**.

WebTechnologies **Rlinkedin**, **ipaddress**, **rdrop2**.

gR **spectralGraphTopology**.

(* = 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 3.12 was released on 28 October, 2020. It is compatible with R 4.0.3 and consists of 1974 software packages, 398 experiment data packages, 968 up-to-date annotation packages, and 28 workflows. Books are a new addition, built regularly from source and therefore fully reproducible; an example is the community-developed Orchestrating Single-Cell Analysis with Bioconductor.

The Bioconductor 3.12 release announcement includes descriptions of 125 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 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 BioC 2020 and BioC Asia 2020, in addition to video recordings of training courses and developer forums.
- Our package submission repository for open technical review of new packages.

Recent Bioconductor conferences include BioC2020 (July 27-31), BioC Asia 2020 (October 15-18), and the European Bioconductor Meeting (December 14-18). Each had invited and contributed talks, as well as workshops and other sessions to enable community participation. Slides, videos, and workshop material for each conference are available on conference web sites as well as the Courses and Conferences section of the Bioconductor web site. BioC 2021 is planned for August 4-6, with an abstract submission due date of March 9; the virtual conference will be augmented by in-person activities if global health permits.

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 2020-09-09 and 2021-01-28.

Donations

WordPress Hosting Buddy (United States) b-data GmbH (Switzerland) JBL Digital Marketing (Australia) Essex Bricklayers (United Kingdom) Jacopo Cerri (Italy) Vancouver Drafting (Canada) The R Conference (United States) Lander Analytics (United States) The New York Open Statistical Programming Meetup (United States) RV Detailing Pros of San Diego (United States) Maple Ridge Handyman (Canada) Burnaby Handyman (Canada) Roger Koenker (United Kingdom) Oleg V Kolesnikov (Ukraine) Bulk CBD Providers (United States) Metal Roofing San Antonio (United States) Minato Nakazawa (Japan) Rashid Nassar (United States) Appstam Consulting GmbH (Germany) Careful Movers (United States) San Diego Piano Moving (United States) Bathroom Remodel Dayton (United States) Nursing Home Vancouver (United States) Clearwater Roofing (United States) Clearwater Windows (United States) Tree Service Brandon (United States) Jacksonville Pavers (United States) Tampe Tree (United States) Fast Movers Tampa (United States) Allen’s Tree Works (United States) Steve Smith (United States) Maple Ridge Tree Service (Canada) Rav Vaid (United States) Merck Research Laboratories, Kenilwort (United States) Statistik Aargau, Aarau (Switzerland)

Supporting benefactors

www.5slotsites.com , Alderley Edge (United Kingdom)

Supporting institutions

Code Ocean, New York (United States) Ef-prime, Inc., 日本橋茅場町 (Japan) Institute of Botany of the Czech Academy of Sciences, Pruhonice (Czechia)

Supporting members

Tim Appelhans (Germany) Christopher Beltz (United States) Gordon Blunt (United Kingdom) Gilberto Camara (Brazil) Susan M Carlson (United States) Cédric Chambru (Switzerland) Michael Chirico (United States) Tom Clarke (United Kingdom) Terry Cox (United States) Robin Crockett (United Kingdom) Robert Daly (Australia) Gergely Daroczi (Hungary) Jasja Dekker (Netherlands) Fraser Edwards (United Kingdom) Dane Evans (United States) Isaac Florence (United Kingdom) Neil Frazer (United States) Huancheng Fu (China) Keita Fukasawa (Japan) Sven Garbade (Germany) Eduardo García Galea (Spain) Anne Catherine Gieshoff (Switzerland) Brian Gramberg (Netherlands) Spencer Graves (United States) Krushi Gurudu (United States) Hlynur Hallgrímsson (Iceland) Joe Harwood (United Kingdom) Bela Hausmann (Austria) BaoGiang HoangVu (Vietnam) Lorenzo Isella (Belgium) Sebastian Jeworutzki (Germany) Grant Joslin (United States) June Kee Kim (Korea, Republic of) Miha Kosmac (United Kingdom) Daniel Krüerke (Switzerland) Jan Herman Kuiper (United Kingdom) Luca La Rocca (Italy) Mauro Lepore (United States) Chin Soon Lim (Singapore) Joseph Luchman (United States) Sharon Machlis (United States) Daniel McNichol (United States) Bogdan-Alexandru Micu (Luxembourg) Jairo Montenegro Arjona (Colombia) Guido Möser (Germany) yoshinobu nakahashi (Japan) Maciej Nasinski (Poland) Tilers in Nottingham (United Kingdom) Bernard Offman (France) Berk Orbay (Turkey) Dan Orsholits (Switzerland) George Ostrouchov (United States) Antonio Paez (Canada) Peter

Perez (United States) Elgin Perry (United States) jared peterson (United States) Kem Phillips (United States) Fergus Reig Gracia (Spain) Ingo Ruczinski (United States) Choonghyun Ryu (Korea, Republic of) Pieta Schofield (United Kingdom) Dejan Schuster (Germany) Jagat Sheth (United States) Rachel Smith-Hunter (United States) Gerardo Soto-Campos (United States) Tobias Strapatsas (Germany) Robert Szabo (Sweden) Ville Tenhunen (Netherlands) Con Tumass-o'Pool (Australia) Uku Vainik (Estonia) Marcus Vollmer (Germany) Jaap Walhout (Netherlands) Sandra Ware (Australia) Lim Zhong Hao (Singapore)

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 second half of 2020.

useR! 2020 breakout session: *Supporting diversity in the R community*

In this breakout session at *useR! 2020*, a panel shared their experience as members of marginalized groups or as allies, then responded to Q&A from *useR!* participants. The panel was chaired by Laura Ación (LatinR co-founder) and Shelmith Kariuki (AfricaR co-founder), and the panelists were Yanina Bellini Saibene (R-Ladies Global Team and LatinR co-founder), Laís Carvalho (Python Ireland board member), Richard Ngamita (KampalaR founder and Forwards Community Team member), Danielle Smalls-Perkins (MiR co-founder), Robin Williams (Blind R User Group member), and Greg Wilson (Software Carpentry co-founder and Education team member at RStudio).

The panel discussed a range of barriers to participation, such as language barriers, limited access to education and conferences, specific challenges faced by visually impaired folk and feelings of isolation due to location or identity. They highlighted some positive steps the R community has made to promote inclusion, for example, founding groups such as Forwards, R-Ladies, AfricaR, MiR and the Blind R User Group; offering diversity scholarships at R conferences and developing technical solutions to improve accessibility. However, the panel also raised the need for greater inclusion of people from minority groups in decision-making and for accessibility to be at the centre of R development and R community events. Allies were recommended to work closely with affinity groups and to base actions on established research, for example following the Ally Skills Workshop (material available under CC BY-SA 4.0). Further suggestions made in the Q&A included offering more tutorials/materials in languages other than English, subtitling videos and offering live streaming.

The full video of the session is available on YouTube with live chat replay. This session was organized by Forwards members Damiano Cerasuolo, Jonathan Godfrey, Liz Hare, Tatjana Kecojevic, Imke Mayer, Kevin O'Brien, Noa Tamir and Heather Turner.

R Contribution Working Group

Partly in response to the useR! breakout session, Forwards established a group to work on initiatives to encourage new contributors to R core, with a focus on diversity and inclusion. The R Contribution Working Group is open to anyone interested in working towards this goal and representatives from R Core, the R Foundation, Forwards, R-Ladies, MiR, the R Consortium Diversity and Inclusion Working Group, as well as members of the general R community have joined in. The group has met every 1-2 months since July 2020, alternating between the second Friday of the month, 15:00 UTC and the second Tuesday of the month, 21:00 UTC.

The group recently created the R Contribution Site to host information for people interested in contributing to R core, which has information on a Slack group that people can join to discuss related issues and support each other in progressing as R contributors. Other initiatives include planning contributor-focused events for useR! 2021. Minutes of meetings and work in progress is gathered in the public rcontribution repository on Forwards GitHub.

Introduction to R Workshop, Lomé, Togo

A 2-day Introduction to R workshop in Lomé, Togo, was held on 16-17 December 2020. The workshop was organized by Anicet Ebou, a member of the AfricaR leadership team based in Ivory Coast. The objective was to introduce people to R and plant the seeds for a local R User Group (as far as we are aware, there is no R-related meetup in Togo). The workshop was co-taught by Audrey Addablah, a leader of Abidjan R User Group (Ivory Coast) and supported by the Why R? Foundation and the R Consortium, as well as Forwards and AfricaR.

Figure 1: Anicet Ebou (left) and Audrey Addablah (right) teaching at the workshop in Lomé, Togo

Audrey and Anicet introduced the workshop participants to handling and visualising data in R. More than 20 people attended the event, including students and professionals from a range of sectors. The participants showed a real interest and we are hopeful that training will continue online and in person in future months.

Latin America Survey

Paola Corrales and Claudia Huaylla joined the Forwards survey team to collaborate on a survey of R users that were born or currently live in Latin America. The survey received close to 1000 responses and they are currently working with other Latin American R users to analyse the results, with a view to report further in 2021.

Package Development Modules

The teaching team have been working on modularizing the Forwards package development workshop materials (developed under a grant from the R Consortium to run Workshops for Women and Girls). Emma Rand and Mine Çetinkaya-Rundel plan to teach the first three modules online, February 1-3, 2021, at 14:30-15:30 UTC each day. You can register for the modules on eventbrite: Packages in a nutshell, Setting up your system, Your first package!.

Changes in Membership

New members

We welcome the following members to the taskforce:

- Community team: s gwynn sturdevant.
- Conferences team: Miljenka Vuko, Becca Wilson.
- On-ramps team: Jyoti Bhogal, Michael Chirico, Maya Gans, Saranjeet Kaur Bhogal.
- Social media team: Maria Prokofieva.

- Surveys team: Pavitra Chakravarty, Paola Corrales, Claudia Huaylla, Anna Vasylytsya (co-leader).
- Teaching team: Mine Çetinkaya-Rundel (co-leader).

Previous members

The following members have stepped down:

- Conferences team: Jesse Mostipak.
- On-ramps team: Zhian N. Kamvar, Charlotte Wickham.
- Social media team: David Smith.
- Teaching team: Angela Li (co-leader), Dorris Scott.

We thank them for their contribution to the taskforce.

Heather Turner
University of Warwick, UK
Heather.Turner@R-project.org

e-Rum2020: how we turned a physical conference into a successful virtual event

by Mariachiara Fortuna, Francesca Vitalini, Mirko Signorelli[1]*, Emanuela Furfaro, Federico Marini, Gert Janssenswillen, Riccardo Porreca, Riccardo L. Rossi, Andrea Guzzo, Roberta Sirovich, Andrea Melloncelli, Lorenzo Salvi, Serena Signorelli, Filippo Chiarello*

Abstract The European R Users Meeting 2020 (e-Rum2020) was a conference that was held virtually in June 2020. Originally, e-Rum2020 had been planned as a physical event to be held in Milano. However, the spread of the COVID-19 pandemic and the declaration of a nationwide lockdown induced the Organizing Committee to fully rethink the event, and to turn it into a live virtual conference. In this article, we describe the challenges that we encountered during the organization of e-Rum2020, and how we reacted to them. In doing so, we aim to provide future conference organizers with useful information on how to organize a successful virtual conference, and even to turn a physical conference into a virtual meeting on a relatively short notice.

Introduction

The European R Users Meeting (eRum) is a series of international conferences that aims to bring together members of the R Community from all over Europe. Hallmarks of this conference are its openness to both the academic and the business world, and low registration fees that aim to minimize the financial burden required to attend the conference.

The first two editions of eRum, eRum2016 and eRum2018, were hosted in Poznan, Poland, in October 2016 (Beresewicz et al., 2017), and in Budapest, Hungary, in May 2018 (Daróczi, 2018). The third edition of eRum, eRum2020, was originally planned to be held in Milano, Italy, in May 2020. The organization of the 2020 conference was already at an advanced stage in February 2020, when the outburst of the COVID-19 pandemic in Northern Italy casted serious doubts on the possibility to hold a physical event in Milano in the upcoming months. After evaluating several alternative options, the Organizing Committee took the decision to turn the event into a free virtual conference to be held from June 17 to June 20, 2020. To mark this change into a virtual event, the 2020 edition was renamed e-Rum2020.

The organization of an international conference typically requires the collaboration and coordination of several professionals, as well as continuous interactions with many stakeholders and service providers. If organizing a conference can already be considered a challenging experience itself, having to quickly turn a physical meeting into a virtual conference in the very middle of a pandemic and of an unprecedented lockdown posed additional, unforeseen organizational challenges.

With this article we would like not only to report how e-Rum2020 went, but also to describe the challenges that we encountered during the organization of the conference, and how we reacted to them. In doing so, we aim to provide future conference organizers with useful information on how to organize a successful virtual conference, and even to turn a physical conference into a virtual meeting on a relatively short notice.

Pre-pandemic arrangements

The original plan was to hold eRum2020 at two universities located in Milano, Italy. The first three days (May, 27th-29th) of the conference would have been hosted by the Università degli Studi di Milano-Bicocca, and the last day (May, 30th), dedicated to workshops, by the Politecnico di Milano. We planned a maximum capacity of 900 participants for the event. The conference would have included a social event that was going to take place at a facility located in Sempione Park, a park in the historical centre of Milano, next to the Sforza Castle.

To facilitate attendance from professionals with parenting responsibilities, at both venues we had arranged a childcare service for kids aged 2-8 years old. An open call for travel grants was published, resulting in 60 applications. A selection committee reviewed the applications and selected 10 awardees based on criteria such as financial need, potential career and development prospects, and expected community impact.

[1]Corresponding author

Decision to go virtual

In February 2020, Italy was the first European country to be hit by the COVID-19 pandemic, with initial outbreaks right in Northern Italy, a few kilometers away from Milano. At that time, the conference organization was already at a fairly advanced stage. In addition to having set the location (including catering facilities, childcare, social event, etc), the scientific program was also being finalised: we had already closed the call for contributions (which received about 220 submissions), we were in the process of notifying authors of accepted contributions, keynote and invited speakers, and we had already sold about 100 tickets. Sixteen between sponsors and organizing partners had signed up for specific sponsorship packages that relied on physical presence, interaction and visibility during the event.

All throughout February and March, we closely monitored the pandemic situation. We organized weekly meetings to discuss the steps to take based on the daily evolution of the containment measures and of the spread of COVID-19 itself. We initially considered three alternatives: (1) cancelling eRum2020, (2) postponing it, or (3) turning the event into a virtual conference. While we were preparing for the second option, Italy was becoming the first country to progressively implement a nationwide lockdown amid COVID-19, and the pandemic was hitting more and more countries. The uncertainty around the possibility of organizing large gatherings by the end of 2020 was growing, and postponing the event was becoming too much of a risk: hence, on April 6th we announced to our sponsors and organizing partners the intention of turning eRum2020 into an online conference (e-Rum2020) and of postponing the event by three weeks to allow more time to re-adjust the format.

Up until that moment, the conference had been completely conceived as an in-person meeting, with several arrangements driven by the pursuit of interaction and conviviality. Switching to an online format was the best choice to preserve the work that we had done, but it also meant additional work to completely rethink the event, and it required some courage to dismantle the old conference structure that we had been working on for a year. In rethinking e-Rum2020, we strived to provide a virtual experience that could be as close as possible to the physical one, and to make the transition as smooth as possible, living up to the expectations of sponsors, speakers and attendees.

While buying time for re-organizing the format and exploring online conferencing platforms, we needed to keep the attention high and the possible audience engaged. We therefore increased the presence of e-Rum2020 on social media (generating additional contents and creating a dedicated YouTube channel) and organized a contest featuring applications of R to data on the COVID-19 pandemic (CovidR contest). We adapted the sponsorship package to the new format by replacing those benefits that required physical interaction (e.g., physical sponsors' booths during the event) with virtual equivalents. Additional visibility was offered through social media, online pre-conference activities and virtual banners, and we also included the possibility of organizing virtual recruiting sessions. Given the reduced costs that the virtual event entailed, we cancelled the registration fees, turning e-Rum2020 in an event free of charge. We refunded previously purchased tickets, and reduced the cost of sponsorship packages.

Following our "as close as possible to physical" principle, we decided that all talks were going to be presented live, with Q&A sessions tailored to the different types of talks. In order to make up for the absence of in-person interaction, we also decided to have dedicated networking areas and to add yoga sessions at the beginning and at the end of each day. In a few days, we realised that the online format was actually a great opportunity to come up with new ideas, and that we could leverage it to bring the conference to a worldwide reach.

In the remainder of the article, we present the technical solutions that we adopted for the virtual conference, the promotion strategy that we implemented to increase the reach of e-Rum2020, and the organization and contents of e-Rum2020's scientific program.

Technical solutions

A primary challenge in the organization of the virtual conference was the identification of technological solutions that could be used to connect participants throughout the event. After comparing several alternative services, we chose to resort to an online conferencing platform called Hopin (`https://hopin.to`). This choice was made because Hopin could efficiently recreate the spaces of a live conference in digital format (reception area, main stage, parallel sessions, sponsors booths, etc). Key factors that motivated our choice were the possibility to hold all sessions live, to have dedicated spaces for both plenary events and parallel sessions, and networking spaces that made interactions between speakers, attendees and sponsors possible.

The conference platform, however, was not the only technological tool needed to provide a smooth conferencing experience. We soon realized that none of the available conferencing platforms, including Hopin, could provide us with a fully comprehensive set of tools, and therefore we decided to complement it with additional tools.

For example, a limitation of Hopin was the lack of a system to communicate effectively “behind the scenes”. To address this problem, we created a Slack channel to manage backstage communications between staff members, speakers and session chairs, and to provide the attendees with technical support. We also decided to manage the Q&A of keynote sessions with sli.do, and to stream part of the event live on YouTube. We used Voicemod to launch applauses and other sounds during the live streaming, and restream.io to handle breaking times and slideshows in the YouTube streaming.

An essential part of this technological setup was the need to explain how it would have worked, and test it with our staff, speakers and session chairs. Four mock events were held with the Organizing Committee members to check that the setup was functional and could accommodate our ideas. Furthermore, we decided to hold a pre-conference event, called CovidR, that was used as dry-run for the event platform, structure and backstage communication tools (we will come back to CovidR in a later section). We wrote a set of guides with detailed instructions for speakers, sponsors and sessions chairs. Lastly, we organized speaker tests to train speakers and session chairs. These tests proved extremely useful not only for the speakers, who learned to use a new and complex tool, but also for us, because it helped us to identify the most common problems and how to solve them before the virtual event took place.

To ensure that sessions ran smoothly, we complemented session chairs with additional support staff: each session had an OC member assigned as supervisor and a Q&A assistant who collected questions during the sessions. We also planned back-ups for each role, in case of connection problems or technical issues. Moreover, we organized technical assistance throughout the whole event.

Promotion of the event

The promotion e-Rum2020 was taken care of by a team of Organizing Committee members, who were supported in their tasks by a graphic designer. The first tasks involved the creation of a logo for the conference, and of the conference website. The logo was initially designed for the physical event that was to be held in Milano; for this reason, it included a stylized representation of the Duomo di Milano, one of the main landmarks of the city. When the decision to hold the event virtually was announced, we decided to redesign the logo by updating event name and dates, and adding a wifi symbol on top of the cathedral (Figure 1).

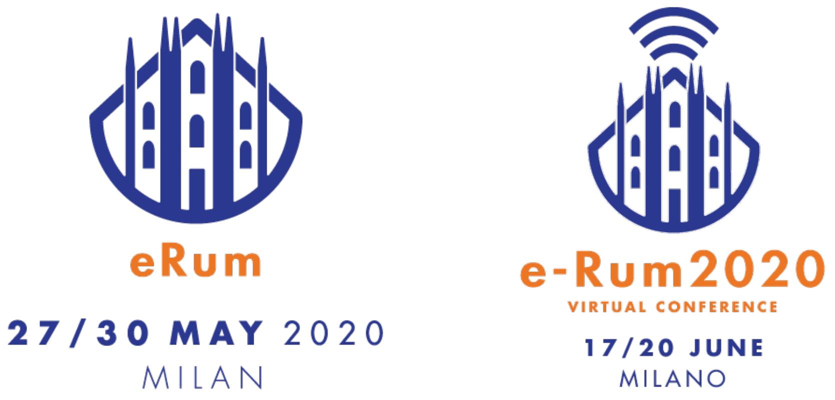


Figure 1: The conference logos designed for the physical conference (eRum2020) that should have been held in May 2020 (left) and for the virtual event (e-Rum2020) that was held in June 2020 (right).

The conference website (`https://2020.erum.io`) was developed using Wordpress, using a color palette and graphical style in line with those used in the conference logo. We opted for a multi-page design with sections and subsections, with the purpose of easing information retrieval. The website underwent frequent updates that reflected important milestones (e.g., opening of submission period, opening of registrations, announcement of virtual event, publication of the program, etc.), with a major reorganization carried out to adapt it to the virtual conference format.

The conference promotion strategy was designed to be fully virtual and costless; it mostly relied on the use of social media (Twitter, Facebook and LinkedIn), on the blog of the MilanoR foundation and on emails spread on target mailing lists. The social media communication strategy was organized using an editorial calendar, in which all announcements and posts for each week were scheduled from the start, thereby making sure that the style and content were uniform over different channels and over time. After the decision to transition to an online conference, we created a dedicated e-Rum2020 YouTube channel and we redrafted the editorial calendar, including additional contents (video interviews with keynote speakers, information about the CovidR pre-conference event, etc.). Among the social networks used, we found Twitter to be the one that generated the highest engagement; the number of followers of eRum2020's Twitter page doubled in a year (growing from 1108 in June 2019 to 2313 in June 2020), with a steep increase observed when the conference was turned into a virtual event (Figure 2).

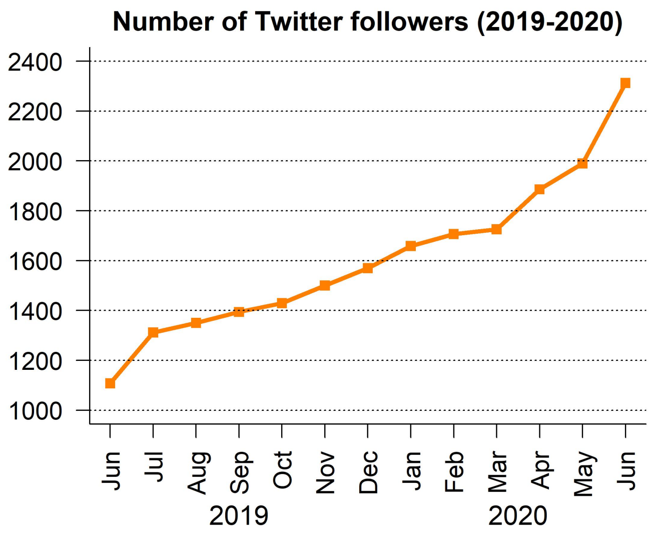


Figure 2: Evolution of the number of followers of the eRum Twitter page from June 2019 to June 2020.

The creation of the e-Rum2020 YouTube channel was instrumental to support the virtual event. To increase e-Rum2020's outreach, we recorded 5 interviews with the keynote speakers that were published prior to the conference. The interviews were intentionally informal and provided a relaxed narration of personal experiences and R-related stories of the speakers. The channel was also used to stream plenary sessions live during the event, and to publish recordings of all conference sessions after the end of the event. In total, the channel currently hosts 59 videos organized in 12 playlists, with a total of 36 hours of edited material uploaded.

Scientific Program

The Scientific Program of e-Rum2020 comprised several types of sessions, which ranged from keynote and invited sessions to different types of contributed sessions (workshops, regular talks, lightning talks, shiny demos and posters). All sessions were organized around a 4-days schedule, with the last day entirely dedicated to the workshops. Moreover, the conference was preceded by two satellite events: the CovidR pre-conference event, and a hackathon on spatial networks.

To ensure a balanced representation of the main fields of application of R, we identified 6 tracks that were used to guide the selection of both keynote and invited speakers, and of contributed sessions. The tracks were: Machine Learning and Modelling, R World, Applications, Life Sciences, Data Visualization and R in Production.

Keynote and invited sessions

The invited part of the program comprised six keynote sessions of 45 minutes, and three invited sessions of 90 minutes each. Six keynote speakers were selected to represent the 6 conference tracks. Of the 11 invited speakers, 9 were selected in representation of the 6 tracks, and two were invited to present their work at eRum2020 after winning the CovidR contest.

Contributed sessions

The call for contributed sessions was opened on December 11, 2019, and closed on January 29, 2020, well before the outbreak of the pandemic in Europe. Abstracts were submitted through the Sessionize system, which offered a number of features for handling mass communications with users, but was lacking a simple yet efficient mechanism for evaluation, with the widely used 1-5 notes for scoring each contribution. We therefore decided to complement it by developing assessR (`https://github.com/Milano-R/assessr`), a Shiny app that seamlessly displayed the most relevant information for each abstract. The app kept the author information hidden, as we believed that a blinded procedure would encourage reviewers to assess the content in a fairer way.

For the evaluation of the submitted abstracts, we created a Program Committee (PC) whose members were selected in representation of the conference tracks, and were asked to score contributions specific to their field of expertise. The PC members were: Aldo Solari, Enrico Deusebio, Charlotte Soneson, Branko Kovac, Andrie De Vries, Fulvia Pennoni, Goran Milovanović, Davide Cittaro, Hannah Frick, Adolfo Alvaro, Olga Mierzwa-Sulima, Gergely Daróczi, Piercesare Secchi, Diane Beldame, Xavier Adam, Davide Risso, Pier Luca Lanzi, Levi Waldron, Heather Turner, Martin Mächler, Stefano Maria Iacus.

We received a total of 230 contributions (Figure 3), of which 32 for workshops. Of these, 94 (including 11 workshops) were selected for presentation at e-Rum2020. All contributed sessions, with the exception of workshops, were organized in two parallel tracks (later assigned to specific rooms in the system provided by Hopin), spread over 3 days. A separate day was dedicated to the workshops.

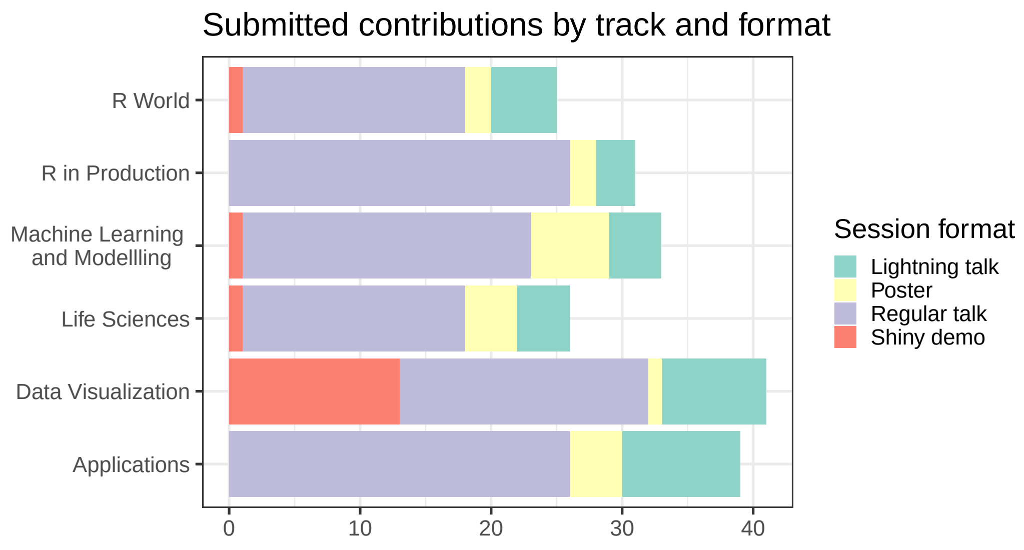


Figure 3: Submitted contributions by track and session type.

Definition of the final program

The final lineup of talks and contributions reflected a nice balance across the six tracks. As we expected that due to the online format participants might have been able to join and stay focused for shorter stretches of time, we decided to shorten the length of all talks.

Each session type was linked to a specific room configuration for Q&As: for example, invited speakers had the possibility to individually host people at a virtual table, while panel Q&A sessions were organized for regular and lightning talks. Considering the time zones of our speakers was another important aspect to take into account. Most contributed sessions were submitted from Europe, but we were also able to accommodate late afternoon slots from the Pacific Time Zone.

Titles, abstracts and authors of each contribution, arranged by format, were published as a booklet using **bookdown** (`https://2020.erum.io/program/contributed-sessions`). That choice made easy and flexible its updates, progressively following confirmations, renunciations and possible changes. The final program of e-Rum2020, comprehensive of the conference schedule, was then published as a brochure (`https://2020.erum.io/program`).

Hackathon on spatial networks

The beginning of e-Rum2020 was preceded by a satellite event on spatial networks that was held the day before the conference. The event comprised an online webinar and a hackathon, and it focused on introducing and testing **sfnetworks**, a new R package for the analysis of spatial networks. The satellite event was organized by Andrea Gilardi, Lorena Abad, Robin Lovelace and Lucas van der Meer.

CovidR pre-conference event

A significant addition to e-Rum2020, decided alongside with the transition to a virtual conference, was CovidR: a contest and pre-conference event featuring open-source R contributions around the topic of the COVID-19 pandemic. Since the beginning of the pandemic, the R community has been very active in using R for analyzing data, developing models and providing useful visualizations. The idea behind CovidR was to collect such contributions and motivate the community to share and spread their work through a contest.

The CovidR submission process was based on a GitHub repository (`https://github.com/Milano-R/erum2020-covidr-contest`) and a streamlined Pull Request mechanism, from which we constructed an R Markdown gallery website (`https://milano-r.github.io/erum2020-covidr-contest`). The website gallery was constructed using the R package **rmdgallery** (`https://riccardoporreca.github.io/rmdgallery`), which was developed alongside the contest. The gallery allowed community engagement through the possibility of thumb-up voting submitted contributions.

Out of the 35 submissions to the contest, 15 were selected to be presented at the CovidR pre-conference event, which was held on May 29, 2020. The event featured 5-minute-long presentations and virtual round tables for Q&As, with awards granted based on the overall quality of the contributions, as well as on community feedback. The two winners of the contest were invited to give an extended presentation of their work at e-Rum2020.

A secondary, but important aspect of this pre-conference was that it enabled us to test the technical tools and the overall process of running a large virtual event. The technical preparation and smoothness of the main conference owe a lot to the lessons learned and confidence gained during CovidR.

e-Rum2020 in figures

Conference tickets were sold out, with 2000 registered participants. After registration, 1379 participants actively attended the conference. The CovidR pre-conference event and the workshops were respectively attended by 171 and 513 participants. Figure 4 shows the geographic distribution of the conference attendees.

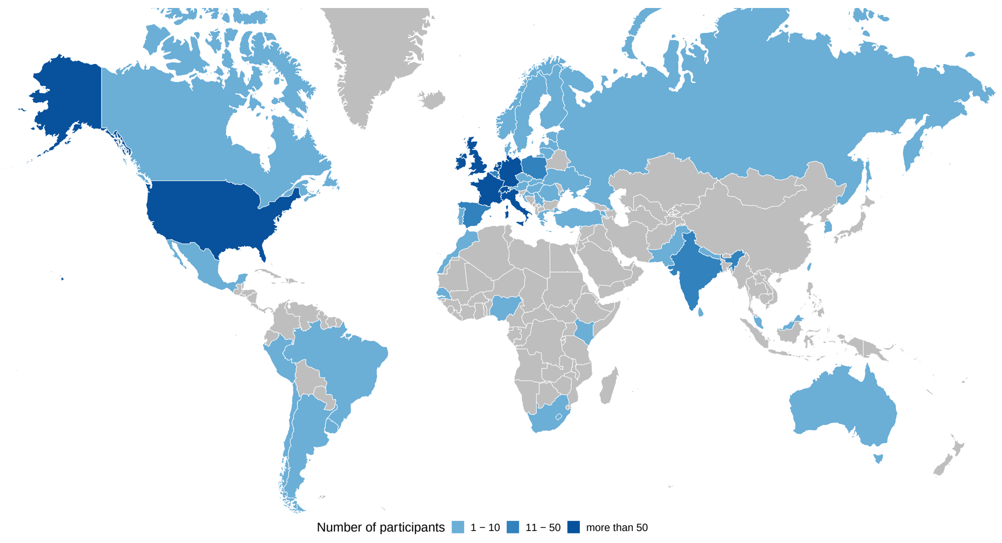


Figure 4: Geographic distribution of the attendees of e-Rum2020.

The program consisted of 104 speakers, among which 6 keynote speakers, 12 invited speakers, and 86 contributed talks. The contributed talk included 39 regular talks, 23 lightning talks, 11 Shiny demos and 13 poster presentations.

The conference staff was entirely composed by volunteers. It counted 21 Organizing Committee members, 21 Program Committee members, 23 session chairs, 7 Q&A assistants, 7 technical support assistants and 1 social media assistant.

Organizers

e-Rum2020 was jointly organized by the e-Rum2020 Organizing Committee, the MilanoR association, two universities (Università di Milano-Bicocca and Politecnico di Milano), and two data science firms (VanLog and Mirai Solutions). The event received the patronage of Comune di Milano.

The Organizing Committee of e-Rum2020 consisted of 21 volunteers based in 5 different European countries (Italy, Switzerland, Belgium, The Netherlands and Germany): Mariachiara Fortuna, Francesca Vitalini, Emanuela Furfaro, Mirko Signorelli, Federico Marini, Riccardo L. Rossi, Roberta Sirovich, Andrea Meloncelli, Lorenzo Salvi, Riccardo Porreca, Andrea Guzzo, Gert Janssenswillen, Serena Signorelli, Filippo Chiarello, Matteo Pelagatti, Gabriele Orlando, Matteo Borrotti, Laura Terzera, Marta Galvani, Matteo Fontana and Parvaneh Shafiei.

Sponsors

We would like to thank all the sponsors that supported the organization of e-Rum2020: RStudio, Open Analytics, cynkra, the R Consortium, the SDG group, Revelo Datalabs, SolidQ, Appsilon Data Science, datahouse and Kode, as well as our in-kind sponsors StickerMule and CRC Press.

Acknowledgments

As members of the e-Rum2020 Organizing Committee, we would like to thank all the volunteers that contributed to the success of e-Rum2020: the Program Committee members, keynote, invited and contributed speakers, and all the volunteers that helped as session chairs, or helped managing the Q&A, technical support for attendees and social media coverage during the event.

Links

Further information about e-Rum2020 and the contributions presented during the conference can be found at the following links:

1. conference website: `https://2020.erum.io`
2. e-Rum2020 Youtube channel: `https://www.youtube.com/c/eRum2020`
3. conference materials: `https://github.com/Milano-R/erum2020program#readme`
4. CovidR gallery with all submissions for the CovidR pre-conference event: `https://milano-r.github.io/erum2020-covidr-contest`
5. AssessR shiny app: `https://github.com/Milano-R/assessr`

Bibliography

M. Beresewicz, A. Alvarez, P. Biecek, M. K. Dyderski, M. Kosinski, J. Nowosad, K. Rotter, A. Szabelska-Beresewicz, M. Szymkowiak, Ł. Wawrowski, et al. Conference Report: European R Users Meeting 2016. *R Journal*, 9(1), 2017. [p417]

G. Daróczi. Conference Report: eRum 2018. *R Journal*, 10(1), 2018. [p417]

Mariachiara Fortuna
Vanlog
Via Amedeo Peyron 19, Torino

Italy
mariachiara.fortuna@vanlog.it

Francesca Vitalini
Mirai Solutions GmbH
Gotthardstrasse 56, Zurich
Switzerland
francesca.vitalini@mirai-solutions.com

Mirko Signorelli
Department of Biomedical Data Sciences, Leiden University Medical Center
Einthovenweg 20, Leiden
The Netherlands
ORCID: 0000-0002-8102-3356
m.signorelli@lumc.nl

Emanuela Furfaro
Department of Statistics, University of California
1 Shields Ave, Davis
United States of America
ORCID: 0000-0002-2440-841X
efurfaro@ucdavis.edu

Federico Marini
Institute of Medical Biostatistics, Epidemiology and Informatics, University Medical Center of the Johannes Gutenberg University Mainz
Obere Zahlbacher Str. 69, Mainz
Germany
ORCID: 0000-0003-3252-7758
marinif@uni-mainz.de

Gert Janssenswillen
Faculty of Business Economics, Hasselt University
Agoralaan, 3590 Diepenbeek
Belgium
0000-0002-7474-2088
gert.janssenswillen@uhasselt.be

Riccardo Porreca
Mirai Solutions GmbH
Gotthardstrasse 56, Zurich
Switzerland
riccardo.porreca@mirai-solutions.com

Riccardo L. Rossi
Istituto Nazionale Genetica Molecolare
Via F. Sforza 35, Milano
Italy
ORCID: 0000-0002-4964-3264
rossi@ingm.org

Andrea Guzzo
Moxoff
Via Schiaffino 11/19, Milano
Italy
ORCID: 0000-0001-7840-1179
andrea.guzzo92@gmail.com

Roberta Sirovich
Department of Mathematics G. Peano, University of Torino
Via Carlo Alberto 10, Torino
Italy

ORCID: 0000-0002-3189-8269
roberta.sirovich@unito.it

Andrea Melloncelli
Vanlog
Via Amedeo Peyron 19, Torino
Italy
andrea.melloncelli@vanlog.it

Lorenzo Salvi
Comune di Torino
Piazza Palazzo di Città 1, Torino
Italy
lore.salvi81@gmail.com

Serena Signorelli
UBI Banca SpA
via Cefalonia 74, Brescia
Italy
serena.signorelli.87@gmail.com

Filippo Chiarello
University of Pisa
Lungarno Antonio Pacinotti 43, Pisa
Italy
filippochiarello.90@gmail.com